A Gentle Introduction to the Boundary Element Method in Matlab/Freemat

Stephen Kirkup[1] and Javad Yazdani[1]

***Abstract*—**The Boundary Element Method is developed in its most simple form; for the solution of Laplace's equation in an interior domain with a straight line approximation to the boundary. The direct and indirect approaches to the boundary element method are included. The methods are developed in Freemat, a language similar to Matlab.

The codes for the solution of Laplace's equation in a general domain with a general (Robin) boundary condition are developed. The codes are applied to a typical test problem. The codes are made available as open source (BEMLAP-MAT package) and can be downloaded from this paper or from the websites listed below[2]. The overall package is expected to become a teaching aid in the MSc course[3] that is presently being developed by the authors.

I. Introduction

The boundary element method (BEM) is an important computational analysis technique that engineers and scientists can apply to a range of problems There are a number of textbooks on the BEM[1,2]. The purpose of this article and the accompanying software is to meet the needs of scientists and engineers who are somewhat unfamiliar with the BEM, but have an understanding of numerical methods and computer programming, or would like to apply the BEM to appropriate engineering problems with minimal fuss.

The application of the boundary element method to an appropriate scientific or engineering problem essentially requires a mesh of the boundary of the domain only, and the determination of the boundary condition on the surface. The computational solution then yields the approximate solution at selected points in the domain.

The BEM is generally more efficient to apply and execute than competing methods, such as the finite element or finite difference methods. Hence the application of the BEM presents an attractive option to scientists and engineers. The authors are developing an MSc and this package is expected to act as a teaching aid on one of the modules in that course

On the other hand the underlying mathematical derivation and numerical analysis is extremely involved. Unfortunately, most textbooks on the BEM concentrate on the mathematics underlying the BEM, considering the ranges of integral equation formulation, ranges of element types, ranges of method derivation. In this article the elegance of the solution method is not lost in a mass of mathematical derivation and analysis. A simple notation is used to assist in the understanding of the development of the BEM.

In this work, the so-called *direct* and *indirect* boundary element methods for the solution of the interior Laplace equation are developed. Laplace's equation is the most straightforward problem to which the BEM can be applied. Laplace's equation also models a number of physical phenomena, such as steady state heat conduction and electrostatics. There is substantial recent research on the application of the BEM to Laplace's equation[3-10].

Over recent decades, Matlab[11] has become an increasingly important language for scientific computation. Freemat[12] is a freely available alternative compiler for Matlab. All codes are developed in Freemat, but they can be also used in the Matlab environment. Matlab/Freemat is based on Matrix arithmetic, allowing an economy of coding and naturally allows parallel processing, if it is available.

II. The Boundary Element Method

Over recent decades, the boundary element method (BEM) has received much attention from researchers and has become an important technique in the computational solution of a number of physical

[1]School of Science and Technology, East Lancashire Institute of Higher Education, Blackburn College UK.
[2] www.east-lancashire-research.org.uk
[2]www.kirkup.info/opensource
[3]www.elihe.ac.uk

problems and has therefore become a widely-used technique in engineering analysis. In common with the better-known finite element method (FEM) and finite difference method (FDM), the boundary element method is essentially a method for solving partial differential equations (PDEs) and can only be employed when the physical problem can be expressed as such. As with the other methods mentioned, the boundary element method is a numerical method and hence it is an important subject of research amongst the numerical analysis community. However, the potential advantages of the BEM have seemed so considerable that the strongest impetus behind its development has come from the engineering community, in its enthusiasm to obtain flexible and efficient computer-based solutions to a range of engineering problems. The boundary element method has found application in such diverse topics as stress analysis, potential flow, electromagnetics, fracture mechanics and acoustics.

The boundary element method is derived through the discretisation of an integral equation that is mathematically equivalent to the original partial differential equation. The essential re-formulation of the PDE that underlies the BEM consists of an integral equation that is defined on the boundary of the domain and an integral that relates the boundary solution to the solution at points in the domain. The former is termed a boundary integral equation (BIE) and the BEM is often referred to as *the boundary integral equation method* or *boundary integral method*. Over the last four decades the term *boundary element method* has become more popular. The other terms are still used in the literature however, particularly when authors wish to refer to the overall derivation and analysis of the methods, rather than their implementation or application.

An integral equation re-formulation can only be derived for certain classes of PDE. Hence the BEM is not widely applicable when compared to the near-universal adaptability of the finite element and finite difference method. However, in the cases in which the boundary element method is applicable, it often results in a numerical method that is easier to use and more computationally efficient than the competing methods.

The advantages in the boundary element method arise from the fact that only the boundary (or boundaries) of the domain of the PDE requires sub-division. (In the finite element method or finite difference method the whole domain of the PDE requires discretisation.) Thus the dimension of the problem is effectively reduced by one. For example an equation governing a three-dimensional region is transformed into one over its surface.

In cases where the domain is exterior to the boundary, as it is in potential flow past an obstacle, or the electrostatic field produced by charged surfaces in the open, the extent of the domain is infinite and hence the advantages of the BEM are even more striking; the equation governing the infinite domain is reduced to an equation over the (finite) boundary.

It is through the subdivision or meshing of the boundary into *panels* of some standard shape, and the approximation of functions on the boundary by low degree polynomials that the integral equations are converted into matrix-vector form. The solution of the discrete form of the integral equation leads to the approximate determination of the otherwise unknown boundary functions. Following this, the solution at any point in the domain can be found through numerical integration over the boundary.

Returning to the comparison with the FEM, the dimensions of the matrices in the BEM are expected to be much smaller for the given expectation of accuracy, since the BEM requires a mesh of the surface only, whereas the FEM requires a mesh of the full domain; for a given problem the FEM needs more elements. However the matrices that arise in the FEM are sparse and structured, making them amenable to special methods of solution, whereas the matrices that arise in the BEM are dense and have no particular apparent structure.

Apart from using the simplest governing equation, we are also considering only the simplest dimensional space- 2D. We also use the simplest elements; flat (straight line) panels with the surface functions approximated by a constant on each panel. An empirical analysis of the boundary element method with respect to Laplace's equation is given in the first author's previous work[13].

III. THE INTERIOR LAPLACE EQUATION

The Laplace equation is the simplest elliptic partial differential equation. It is one of the equations of potential theory and they have been received extensive mathematical analysis. It also serves as model elliptic equations for learning, implementing and testing numerical methods. In this article we are using the Laplace equation in order to motivate our understanding of the properties and practice of the boundary element method.

The boundary element method in this paper is

developed to solve the two-dimensional Laplace Equation

$$\frac{\partial^2 \varphi(\mathbf{p})}{\partial x^2} + \frac{\partial^2 \varphi(\mathbf{p})}{\partial y^2} = 0 \quad (\mathbf{p} \in D)$$

or in the shorthand form:

$$\nabla^2 \varphi(\mathbf{p}) = 0 \qquad (\mathbf{p} \in D) \quad (1)$$

in an interior domain D, with an enclosing boundary S, as illustrated in figure 1.

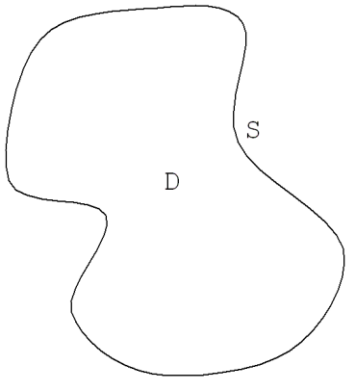

Fig 1. Illustration of the domain.

A boundary condition is determined on S. For this work we assume that the boundary condition is of the following general *Robin* or mixed form

$$a(\mathbf{p})\varphi(\mathbf{p}) + b(\mathbf{p})\frac{\partial \varphi}{\partial n_p}(\mathbf{p}) = f(\mathbf{p}) \quad (\mathbf{p} \in S), \quad (2)$$

where $a(\mathbf{p})$, $b(\mathbf{p})$ and $f(\mathbf{p})$ are real-valued functions defined on S only and n_p is the unit outward normal to the boundary at **p** (assumed to be unique). The general boundary condition includes the Dirichlet (essential) boundary condition ($a(\mathbf{p}) = 1, b(\mathbf{p}) = 0$) and Neumann (derivative) boundary condition ($a(\mathbf{p}) = 0, b(\mathbf{p}) = 1$).

Together, the governing partial differential equation within a domain (eg (1)) and the boundary condition (2) is called a boundary value problem (BVP). The solution to such a problem is principally the determination of φ (at points) in the domain *D*, whether by analytic or numerical methods.

IV. Integral Equation Reformulation

The first stage in the development of a boundary element method from a boundary value problem (like (1)-(2)) is to rewrite the partial differential equation as an integral equation. Traditionally, there have been two ways of doing this; the *direct* method and the *indirect* method. In this section we will go through the stages for developing the integral equation formulations..

Green's function

In order to do this it is useful to introduce an *influence function*; a function that determines the effect at a point **q** of a unit source at a point **p**, this function is also often known as a *Green's function*. For the two-dimensional Laplace equation (1), the Green's function is known to be

$$G(\mathbf{p},\mathbf{q}) = \frac{-1}{2\pi}\ln(|\mathbf{p}-\mathbf{q}|). \quad (3)$$

The Green's function has the property

$$\nabla^2 G(\mathbf{p}-\mathbf{q}) = \delta(\mathbf{p}-\mathbf{q})$$

where δ is the Dirac delta function.

Laplace Integral Operators

As a further set of building blocks, it is also useful to define the set of *Laplace integral operators*:

$$\{L\zeta\}_\Gamma(\mathbf{p}) = \int_\Gamma G(\mathbf{p},\mathbf{q})\zeta(\mathbf{q})dS_q \quad (4a),$$

$$\{M\zeta\}_\Gamma(\mathbf{p}) = \int_\Gamma \frac{\partial G(\mathbf{p},\mathbf{q})}{\partial n_q}\zeta(\mathbf{q})dS_q \quad (4b),$$

$$\{M^t\zeta\}_\Gamma(\mathbf{p};\mathbf{w}) = \frac{\partial}{\partial w}\int_\Gamma G(\mathbf{p},\mathbf{q})\zeta(\mathbf{q})dS_q = \int_\Gamma \frac{\partial G(\mathbf{p},\mathbf{q})}{\partial w}\zeta(\mathbf{q})dS_q$$

(4c)

$$\{N\zeta\}_\Gamma(\mathbf{p};\mathbf{w}) = \frac{\partial}{\partial w}\int_\Gamma \frac{\partial G(\mathbf{p},\mathbf{q})}{\partial n_q}\zeta(\mathbf{q})dS_q$$

(4d)

where Γ is the whole or any part of *S*, ζ is any real-valued function, defined on Γ.

When **p**∈Γ, then we have the more particular form of M^t and *N*:

$$\{M^t\zeta\}_\Gamma(\mathbf{p};\mathbf{n}_p)=\frac{\partial}{\partial n_p}\int_\Gamma G(\mathbf{p},\mathbf{q})\zeta(\mathbf{q})dS_q=\int_\Gamma\frac{\partial G(\mathbf{p},\mathbf{q})}{\partial n_p}\zeta(\mathbf{q})dS_q \quad (4e)$$

$$\{N\zeta\}_\Gamma(\mathbf{p};\mathbf{n}_p)=\frac{\partial}{\partial n_p}\int_\Gamma\frac{\partial G(\mathbf{p},\mathbf{q})}{\partial n_q}\zeta(\mathbf{q})dS_q \quad (4f)$$

Note that the derivative $\frac{\partial}{\partial n_p}$ cannot always be taken directly inside the integral in (4f), if we did then the integral can be *hypersingular*, and therefore not defined in the normal sense, when **q**=**p**. *N* is therefore not a true integral operator, but belongs to the more general class of *pseudo-differential* operators. For convenience, we will continue to refer to *L*,*M*,*M*^{t}and *N* as integral operators, but we will also keep the special case on *N* in mind.

Applying *L* and *N* to any function ζ, any boundary Γ and any vector $\mathbf{n}_p$ gives rise to a continuous function in space. However the operators *M* and *M*^{t} have jump discontinuities at the boundary:

$$\lim_{\varepsilon\to 0}\{M\zeta\}_\Gamma(\mathbf{p}-\varepsilon\mathbf{n}_p)-\frac{1}{2}\zeta(\mathbf{p})=\{M\zeta\}_\Gamma(\mathbf{p}) \quad (5a)$$

$$\lim_{\varepsilon\to 0}\{M^t\zeta\}_\Gamma(\mathbf{p}-\varepsilon\mathbf{n}_p;\mathbf{n}_p)+\frac{1}{2}\zeta(\mathbf{p})=\{M^t\zeta\}_\Gamma(\mathbf{p};\mathbf{n}_p) \quad (5b)$$

where **p**εΓ and Γ is smooth at **p**.

Direct Method

The following equation arises as a result of Green's second theorem

$$\{M\varphi\}_S(\mathbf{p})-\{Lv\}_S(\mathbf{p})=-\varphi(\mathbf{p}) \quad (\mathbf{p}\varepsilon D). \quad (6a)$$

where $v=\frac{\partial\varphi(\mathbf{p})}{\partial n_p}$. For points on *S* we apply the limit (5a) in equation (6a):

$$\{M\varphi\}_S(\mathbf{p})-\{Lv\}_S(\mathbf{p})=-\frac{1}{2}\varphi(\mathbf{p}) \quad (\mathbf{p}\varepsilon S). \quad (6b)$$

Given these equations, the method of solution would involve solving (6b) with the boundary condition (2) in order to find approximations to φ and *v* on the boundary and then use equation (6a) to compute φ at any chosen points in the domain.

However, there is one notable case when this method will not work as well. In the case of a pure Dirichlet boundary condition, (6a) is effectively a Fredholm integral equation of the first kind. It is well known that the numerical solution of first kind equations is not as efficient as it is for the equivalent second kind equation (which equation (6b) would otherwise be) [14].

We can easily introduce another equation, using the Laplace integral operators, that will be useful to us. Differentiating equation (6a) with respect to a vector **w**, gives:

$$\{N\varphi\}_S(\mathbf{p};\mathbf{w})-\{M^tv\}_S(\mathbf{p};\mathbf{w})=-\frac{\partial\varphi}{\partial w}(\mathbf{p}) \quad (\mathbf{p}\varepsilon D). \quad (7a)$$

For points **p** near the boundary, with $\mathbf{n}_p$ being the unique unit outward normal at **p**, then (7a) becomes

$$\{N\varphi\}_S(\mathbf{p};\mathbf{n}_p)-\{M^tv\}_S(\mathbf{p};\mathbf{n}_p)=-\frac{\partial\varphi}{\partial n_p}(\mathbf{p}) \quad (\mathbf{p}\varepsilon D) \quad (7b)$$

Moving the point **p** to the surface and applying the limit (5b) gives rise to the following equation on the surface:

$$\{N\varphi\}_S(\mathbf{p};\mathbf{n}_p)-\{M^tv\}_S(\mathbf{p};\mathbf{n}_p)=-\frac{1}{2}v(\mathbf{p}) \quad (8)$$

One disadvantage in using equation (8) as a basis for solving the Laplace equation, is that it now contains the *hypersingular* operator *N*. The other disadvantage is that if we wish to solve the Neumann problem using equation (8) then we have to solve over the operator *N*, which leads to a similar loss of efficiency that is found in solving first kind equations.

In order to avoid the problems with the Dirichlet problem with equation (6b) and the Neumann problem with equation (8), a hybrid equation is proposed:

$$\{(M+\frac{1}{2}I+\mu N)\}\varphi(\mathbf{p})=\{(L+(M^t-\frac{1}{2}I))\}v(\mathbf{p}). \quad (9)$$

For suitable weighting parameter μ, equation (9) forms a suitable basis for solving the Robin BVP and the special cases of the Dirichlet and Neumann BVPs.
Once approximations to φ and v are found on the boundary from equation (8) with the boundary condition (2), we can use equation (6a) to determine and approximation to φ for any point (**p**) in the domain.

Indirect Method

The alternative or *indirect* approach to obtaining an integral reformulation of the PDE involves writing the solution φ as a layer potential. The most obvious way of doing this is to write

$$\varphi(\mathbf{p}) = \{L\sigma\}_S(\mathbf{p}) \quad (\mathbf{p}\varepsilon D \cup S) \tag{10a}$$

where σ is a density function defined on *S*.

It is possible to solve the Dirichlet problem from equation (10a). This would normally involve finding σ on *S* by solving the integral equation (10a). However, the same equation cannot be used for the Neumann problem and, what is more, it requires solution to be carried out over the first kind operator *L*.

By differentiating the equation (10a) with respect to any vector **w**, we obtain

$$\frac{\partial\varphi}{\partial w}(\mathbf{p}) = \frac{\partial}{\partial w}\{L\sigma\}(\mathbf{p}) = \{M^t\sigma\}(\mathbf{p};\mathbf{w}) \quad (\mathbf{p}\varepsilon D) \tag{10b}$$

As **p** approaches the boundary and we take $\mathbf{w}=\mathbf{n}_p$. and on the boundary equation (10b) becomes

$$\frac{\partial\varphi}{\partial n_p}(\mathbf{p}) = v(\mathbf{p}) = \frac{\partial}{\partial n_p}\{L\sigma\}(\mathbf{p}) = \{M^t\sigma\}(\mathbf{p};\mathbf{n}_p) + \frac{1}{2}\sigma(\mathbf{p}) \quad (\mathbf{p}\varepsilon S) \tag{10c}$$

where the jump discontinuity (5b) has been included.

Equation (10c) relates v on S to σ. Hence equation (10c) can be used as a basis for solving the Neumann problem. It is a second kind equation and so it is very suitable as a basis for solution.

We do not have a more general solution method. To do this let us introduce a hybrid single- and double-layer potential:

$$\varphi = \{L\sigma_\mu\}(\mathbf{p}) + \mu\{M\sigma_\mu\}(\mathbf{p}) \quad (\mathbf{p}\varepsilon D) \tag{11a}$$

where σ_μ is a density function that depends on the choice of μ.

By allowing the point **p** approach the boundary *S*, equation (11a) becomes:

$$\varphi = \{L\sigma_\mu\}(\mathbf{p}) + \mu\{M\sigma_\mu\}(\mathbf{p}) - \frac{\mu}{2}\sigma_\mu(\mathbf{p}) . \quad (\mathbf{p}\varepsilon S) \tag{11b}$$

For $\mu\neq 0$ equation (11b) is a suitable equation to solve the Dirichlet problem since it is always a second-kind integral equation. However, for the Neumann and more general Robin problem it is useful to introduce another equation that is the outcome of differentiating equation (11a), firstly with respect to any vector **w**:

$$\frac{\partial\varphi}{\partial w}(\mathbf{p}) = \frac{\partial}{\partial w}\{L\sigma_\mu\}(\mathbf{p}) + \mu\frac{\partial}{\partial w}\{M\sigma_\mu\}(\mathbf{p}) . \quad (\mathbf{p}\varepsilon D) \tag{11c}$$

Allowing **p** to approach the boundary and **w** becomes the unit outward normal to the boundary there gives the following equations:

$$\frac{\partial\varphi}{\partial n_p}(\mathbf{p}) = \frac{\partial}{\partial n_p}\{L\sigma_\mu\}(\mathbf{p}) + \mu\frac{\partial}{\partial n_p}\{M\sigma_\mu\}(\mathbf{p}) + \frac{\mu}{2}\sigma_\mu(\mathbf{p}) . \quad (\mathbf{p}\varepsilon S) \tag{11d}$$

Using the operator notation, this give

$$v(\mathbf{p}) = \{M^t\sigma_\mu\}(\mathbf{p}) + \mu\{N\sigma_\mu\}(\mathbf{p}) - \frac{\mu}{2}\sigma_\mu(\mathbf{p}) . \quad (\mathbf{p}\varepsilon S) \tag{11e}$$

Substituting the expressions for φ and v in equation (11a) and (11b) into the general Robin boundary condition (2) gives the following boundary integral equation:

$$\{\alpha(L + \mu(M - \frac{1}{2}I)) + \beta((M^t + \frac{1}{2}I) + \mu N)\}\sigma_\mu(\mathbf{p}) = f(\mathbf{p}) \tag{11e}$$

Equation (11e) is most suitable for the solution of the classes of boundary conditions considered.

The indirect boundary element method involves solving (11e) to return an approximation to σ_μ on the boundary. Equation (11a) can then be employed to compute an approximation to φ in the domain *D*.

V. THE DISCRETE OPERATORS

There is a variety of techniques for deriving the system of linear equations from a given integral equation. In general, a method can be derived by replacing the integrals in an integral equation by a quadrature formula or by a weighted residual method such as the Galerkin method. Many methods for solving integral equations can be used to develop a particular boundary element method [15]. Of all the methods the method of collocation is one of the most straightforward and most popular and it is the one that we will be using in this paper.

Collocation

The application of collocation to a boundary integral equation requires that the boundary is represented by a set of *panels*. For example a two dimensional boundary can be approximated by a set of straight lines, as illustrated in figure 2.

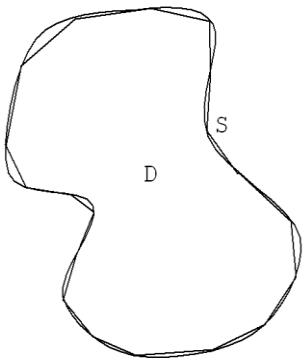

Fig 2. Illustration of the boundary divided into panels.

In order to complete the discretisation of the integral equations, the boundary functions also need to be approximated on each panel. It is the characteristics of the panel and the representation of the boundary function on the panel that together define the *element* in the boundary element method. By representing the boundary functions by a characteristic form on each panel, the boundary integral equations can be simplified into a linear system of equations. Most simply, the boundary functions can be approximated by a constant on each panel. The collocation (or representative) point is at the centre of the panel (C^{-1}collocation). The overall process is that of discretising the integral operators and the methods for carrying this out are covered in (for the more general Helmholtz equation) [16].

Let the ΔSj (for j = 1, 2, .., n) be the n panels that represent an approximation to S in the boundary element method. We may write

$$S \approx \tilde{S} = \sum_{j=1}^{n} \Delta S_j \ . \tag{12}$$

Following from equation (4a), we may write:

$$\{L\zeta\}_S(\mathbf{p}) = \int_S G(\mathbf{p},\mathbf{q})\zeta(\mathbf{q})dS_q \approx \int_{\tilde{S}} G(\mathbf{p},\mathbf{q})\zeta(\mathbf{q})dS_q$$
$$= \sum_{j=1}^{n} \int_{\Delta\tilde{S}} G(\mathbf{p},\mathbf{q})\zeta(\mathbf{q})dS_q \approx \sum_{j=1}^{n} \zeta_j \int_{\Delta\tilde{S}_j} G(\mathbf{p},\mathbf{q})dS_q$$
$$= \sum_{j=1}^{n} \zeta_j \{Le\}_{\Delta S_j}(\mathbf{p}) \tag{13a}$$

where in the final expression we have made the approximation $\zeta(\mathbf{q}) \approx \zeta_j$ (a constant) on the j[th] panel and *e* is the unit function. A similar discretisation can be applied to the other integral operators:

$$\{M\zeta\}_S(\mathbf{p}) \approx \sum_{j=1}^{n} \zeta_j \{Me\}_{\Delta S_j}(\mathbf{p}) \tag{13b}$$

$$\{M^t\zeta\}_S(\mathbf{p}) \approx \sum_{j=1}^{n} \zeta_j \{M^t e\}_{\Delta S_j}(\mathbf{p}), \tag{13c}$$

and

$$\{N\zeta\}_S(\mathbf{p}) \approx \sum_{j=1}^{n} \zeta_j \{Ne\}_{\Delta S_j}(\mathbf{p}). \tag{13d}$$

For any point **p**, $\{Le\}_{\Delta S_j}(\mathbf{p})$, $\{Me\}_{\Delta S_j}(\mathbf{p})$, $\{M^t e\}_{\Delta S_j}(\mathbf{p})$ and $\{Ne\}_{\Delta S_j}(\mathbf{p})$ are termed the discrete Laplace integral operators.

Simplifying the integrands

Writing G(**p**,**q**) as G(r) where r=|**r**| and **r**=**q**-**p**, the Green's function (3) can be written as follows:

$$G(r) = \frac{-1}{2\pi}\ln(r). \tag{14}$$

The derivatives of *G* with respect to r:

$$\frac{\partial}{\partial r}G(r) = \frac{-1}{2\pi}\frac{1}{r}, \tag{15}$$

and

$$\frac{\partial^2}{\partial r^2}G(r) = \frac{1}{2\pi}\frac{1}{r^2}. \tag{16}$$

The normal derivatives of *G*:

$$\frac{\partial G}{\partial n_q} = \frac{\partial G}{\partial r}\frac{\partial r}{\partial n_q}, \tag{17}$$

$$\frac{\partial G}{\partial n_p} = \frac{\partial G}{\partial r}\frac{\partial r}{\partial n_p}, \tag{18}$$

and

$$\frac{\partial^2 G}{\partial n_p \partial n_q} = \left(\frac{\partial G}{\partial r}\frac{\partial^2 r}{\partial n_p \partial n_q} + \frac{\partial^2 G}{\partial r^2}\frac{\partial r}{\partial n_p}\frac{\partial r}{\partial n_q}\right). \tag{19}$$

The normal derivatives of *r*:

$$\frac{\partial r}{\partial n_q} = -\frac{\mathbf{r}.\mathbf{n}_q}{r}, \tag{20}$$

$$\frac{\partial r}{\partial n_p} = \frac{\mathbf{r}.\mathbf{n}_p}{r}, \tag{21}$$

and

$$\frac{\partial^2 r}{\partial n_p \partial n_q} = -\frac{1}{r}(\mathbf{n}_p.\mathbf{n}_q + \frac{\partial r}{\partial n_p}\frac{\partial r}{\partial n_q}). \tag{22}$$

Evaluating the integrals

In most cases of evaluating the discrete integrals in (13a-d), the integrand is continuous and can be approximated most efficiently by Gaussian quadrature. For the case in which **p** lies on the element of integration, the integral can be evaluated by a simple formulae. Let the element Δ have length *a*+*b* with the point **p** lying a distance *a* from one end and a distance *b* from the other, as illustrated in the figure 3.

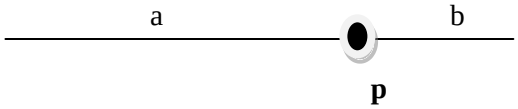

Figure 3. A general element with the collocation point **p** lying on the element.

In the case illustrated in figure 3 we can derive the following expressions:

$$\{Le\}_{\Delta}(\mathbf{p}) = \frac{1}{2\pi}(a+b-a\log a-b\log b) \tag{23a}$$

$$\{Me\}_{\Delta}(\mathbf{p}) = 0 \tag{23b}$$

$$\{M^t e\}_{\Delta}(\mathbf{p}) = 0 \tag{23c}$$

$$\{Ne\}_{\Delta}(\mathbf{p}) = \frac{1}{2\pi}(\frac{1}{a}+\frac{1}{b}) \tag{23d}$$

VI. Parallel Methods for Approximating the Discrete Laplace Operators

We develop a potentially parallel method by avoiding loops and using the in-built matrix/vector operations in Matlab or Freemat. By doing this we are effectively making each loop independent and therefore can be carried out in parallel.

Firstly, we need to set up a general purpose Gaussian quadrature rule (Gauss-Legendre). The routine gl.m sets up an 8 point rule for integration over the interval [0,1].

gl.m

```
 % function [wts,pts,n]=gl()
% Sets up a Gauss-Legendre Quadrature
%  for integration over [0,1]
%  with n=8 weights (wts) and points (pts)

      function [wts,pts,n]=gl()
      n=8;
      wts= [ 5.061426814519E-02
            0.111190517227
            0.156853322939
            0.181341891689
            0.181341891689
            0.156853322939
            0.111190517227
            5.061426814519E-02];
      pts= [ 1.985507175123E-02
            0.101666761293
            0.237233795042
            0.408282678752
            0.591717321248
            0.762766204958
            0.898333238707
            0.980144928249];
```

As a most useful building block, it would be helpful to have a routine for evaluating the four Laplace integral operators, That is for any point **p**, we wish to compute $\{Le\}_{\Delta S_j}(\mathbf{p})$, $\{Me\}_{\Delta S_j}(\mathbf{p})$, $\{M^t e\}_{\Delta S_j}(\mathbf{p})$ and $\{Ne\}_{\Delta S_j}(\mathbf{p})$. Given that many of the intermediate steps for computing the operators are similar, it is more efficient to compute them simultaneously. The operators needed depend on the formulation used hence the routine has the facility for setting the required operators.

The routine for doing this is l2lc.m. The method follows the formulas given in section V. Given the routine will be called many times, its efficiency is one of the critical factors in the efficiency of the overall boundary element method. In l2lc.m, loops are avoided completely (replaced by vector operations) so that any available parallelism can be taken advantage of.

l2lc.m

```
% function
[l,m,mt,n]=l2lc(p,vecp,qa,qb,lponq,needl,needm,needmt,needn)
% Returns the discrete Laplace operators for the observation point p,
%  the derivatve at the observation point (if applicable) vecp, the
%  coordinates of the edges of the element qa and qb, lponq states
whether
%  p lies on the element (true) or not (false), and
needl,needm,needmt,needn
%  state whether the discrete operators l,m,mt and n are needed (if
any
%  operator is not needed then a corresponding zero is returned.

function [l,m,mt,n]=
l2lc(p,vecp,qa,qb,lponq,needl,needm,needmt,needn)

     oo2pi=0.5/pi;
qbma=qb-qa;
qlen=norm(qbma);
pmqa=p-qa;
pmqb=p-qb;
normq(1)=-qbma(2)/qlen;
normq(2)=qbma(1)/qlen;
pqalen=norm(pmqa);
pqblen=norm(pmqb);
dnpdq=vecp*normq';

l=0;
m=0;
mt=0;
n=0;


if (lponq)
  l=(qlen-(pqalen*log(pqalen)+pqblen*log(pqblen)))*oo2pi;
  m=0;
  mt=0;
  n=-(1/pqalen+1/pqblen)*oo2pi;
  continue
else

  [w,x,npoints]=gl();

  onesnp=ones(1,npoints);

  delta=qbma'*x;

  qasame=[qa(1).*onesnp; qa(2).*onesnp];
  psame=[p(1)*onesnp; p(2)*onesnp];

  q=qasame+delta;
  rr=psame-q;
  srr=rr.^2;

  srr1=srr(1,1:npoints);
  srr2=srr(2,1:npoints);
  sr=srr1+srr2;
  r=sqrt(sr);

  if (needl)
    g=-oo2pi*log(r);
    l=qlen*(w*g');
  end

  if (needm|needmt|needn)
    rnqr=-normq*rr;
    rnq=rnqr./r;
    rnpr=vecp*rr;
    rnp=rnpr./r;
    gr=-oo2pi/r;
    wgr=(w.*gr);
  end

  if (needm)
    m=qlen*(wgr*rnq');
  end

  if (needmt)
     mt=qlen*(wgr*rnp');
  end

  if (needn)
   rnprnq=rnp.*rnq;
   dnpnq=vecp*normq';
   dnpnqsame=dnpnq*onesnp;
   rnpnq=-(dnpnqsame+rnprnq)./r;
   grr=oo2pi/sr;
   wgrr=w.*grr;
   n=qlen*(wgr*rnpnq'+wgrr*rnprnq');
end
```

VII. Defining the Boundary

A method is required for defining the boundary as a set of panels. The easiest way of showing how this may be done is by example. Figure 4 shows a square of dimension 0.1 divided into 32 panels.

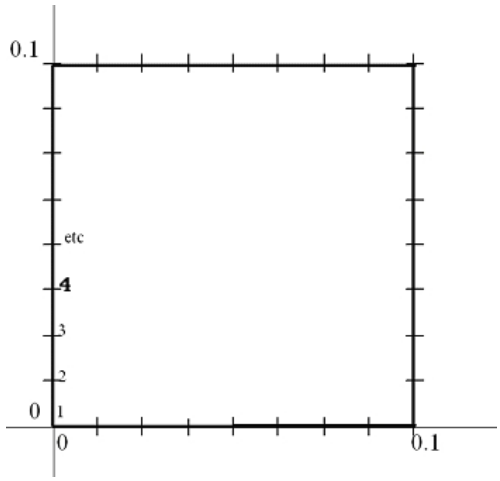

Figure 4. A square divided into 32 panels.

The description in the following Matlab/Freemat code (square.m) involves listing the coordinates of the vertices of the panels in vertpts and lists the indices of the edges of each element in elemvert.

square.m

```
% function [vertpts,elemvert]=square
% Returns a list of edges of the panels that make up the square
%  in vertpts and the list of elements that make up the boundary
%  are defined by the indices of the two vertices in elemvert.
% Note that the vertices of each element in elemvert must be
%  defined in the clockwise direction.

function [vertpts,elemvert]=square

 vertpts=[0.00000000000 0.00000000000;
        0.00000000000 0.01250000000;
        0.00000000000 0.02500000000;
        0.00000000000 0.03750000000;
        0.00000000000 0.05000000000;
        0.00000000000 0.06250000000;
        0.00000000000 0.07500000000;
        0.00000000000 0.08750000000;

        0.00000000000 0.10000000000;
        0.01250000000 0.10000000000;
        0.02500000000 0.10000000000;
        0.03750000000 0.10000000000;
        0.05000000000 0.10000000000;
        0.06250000000 0.10000000000;
        0.07500000000 0.10000000000;
        0.08750000000 0.10000000000;

        0.10000000000 0.10000000000;
        0.10000000000 0.08750000000;
        0.10000000000 0.07500000000;
        0.10000000000 0.06250000000;
        0.10000000000 0.05000000000;
        0.10000000000 0.03750000000;
        0.10000000000 0.02500000000;
        0.10000000000 0.01250000000;

        0.10000000000 0.00000000000;
        0.08750000000 0.00000000000;
        0.07500000000 0.00000000000;
        0.06250000000 0.00000000000;
        0.05000000000 0.00000000000;
        0.03750000000 0.00000000000;
        0.02500000000 0.00000000000;
        0.01250000000 0.00000000000];

elemvert=[1:32; 2:33]
elemvert(2,32)=1
```

VIII. Matrices in the Boundary Element Method

The boundary element method is derived by applying an integral equation method to the appropriate boundary integral equation. The most straightforward method to apply is that of collocation.

The initial development of the application of collocation to give expressions for the discrete Laplace operators is given in Section V. To continue the development, approximations to the (unknown) boundary functions – whether that be φ and/or *v* in the direct method, or σ in the indirect method – are obtained by replacing the boundary integral equation by a matrix-vector equation and then solving it. Approximations to φ in the domain can then be found by direct integration.

In the collocation method, the general point **p** in the boundary integral equation (e.g. equations (9) for the direct method or equation (11e) for the indirect method), takes the value of every central point on each panel; the collocation points: $\mathbf{p}=\mathbf{p}_{S1}, \mathbf{p}_{S2},\ldots, \mathbf{p}_{Sn}$.

For illustration, let us apply the collocation method to boundary integral equation (10a) (for $\mathbf{p}\varepsilon S$**):**

$$\varphi(\mathbf{p}) = \{L\sigma\}_S(\mathbf{p}) \ (\mathbf{p}\varepsilon S) \tag{24}$$

Following the development in equation (13a), the following discrete form of equation (24) is obtained:

$$\varphi(\mathbf{p}) = \{L\sigma\}_S(\mathbf{p}) \approx \sum_{j=1}^{n} \sigma_j \{Le\}_{\Delta S_j}(\mathbf{p}) .$$

Allowing **p** to take the value of $\mathbf{p}_{Si}$, a collocation points gives the following:

$$\varphi(\mathbf{p}_{Si}) = \{L\sigma\}_S(\mathbf{p}_{Si}) \approx \sum_{j=1}^{n} \sigma_j \{Le\}_{\Delta S_j}(\mathbf{p}_{Si}).$$

Let us now introduce the notation:

$$\varphi_{Si} = \varphi(\mathbf{p}_{Si}), v_{Si} = v(\mathbf{p}_{Si}), \sigma_{Si} = \sigma(\mathbf{p}_{Si}),$$

$$\underline{\varphi}_S = \begin{bmatrix} \varphi_{S1} \\ \varphi_{S2} \\ \vdots \\ \vdots \\ \varphi_{Sn} \end{bmatrix}, \quad \underline{v}_S = \begin{bmatrix} v_{S1} \\ v_{S2} \\ \vdots \\ \vdots \\ v_{Sn} \end{bmatrix}, \quad \underline{\sigma}_S = \begin{bmatrix} \sigma_{S1} \\ \sigma_{S2} \\ \vdots \\ \vdots \\ \sigma_{Sn} \end{bmatrix},$$

$$[\mathbf{L}_{SS}]_{ij} = \{Le\}_{\Delta S_j}(\mathbf{p}_{Si}),$$

$$[\mathbf{M}_{SS}]_{ij} = \{Me\}_{\Delta S_j}(\mathbf{p}_{Si}),$$

$$[\mathbf{M}^{\mathbf{t}}{}_{SS}]_{ij} = \{M^t e\}_{\Delta S_j}(\mathbf{p}_{Si}),$$

$$[\mathbf{N}_{SS}]_{ij} = \{Ne\}_{\Delta S_j}(\mathbf{p}_{Si}),$$

Returning to the integral equation (24): by applying the collocation method it is then replaced by the following equation:

$$\underline{\varphi}_S \approx \mathbf{L}_{SS}\underline{\sigma}_S,$$

Which can be solved for the Dirichlet case to return an approximation to σ on S. (Although basing the BEM on this first kind equation is not advised; this is meant to be purely illustrative.)

Usually, the objective is to find the solution φ in the domain. Returning to our illustrative boundary integral equation (10a) (for **p**εD**):**

$$\varphi(\mathbf{p}) = \{L\sigma\}_S(\mathbf{p}) \ (\mathbf{p}\varepsilon D) \tag{25}$$

For points **p** in the domain, we can approximate φ(**p**) as before

$$\varphi(\mathbf{p}) = \{L\sigma\}_S(\mathbf{p}) \approx \sum_{j=1}^{n} \sigma_j \{Le\}_{\Delta S_j}(\mathbf{p}),$$

except in its application, an approximation to σ (the values of the σ_i) has been obtained.

Let the solution be sought at the *m* domain points **p** = $\mathbf{p}_{D1}, \mathbf{p}_{D2}, \ldots, \mathbf{p}_{Dm}$

$$\varphi(\mathbf{p}_{Di}) = \{L\sigma\}_S(\mathbf{p}_{Di}) \approx \sum_{j=1}^{n} \sigma_j \{Le\}_{\Delta S_j}(\mathbf{p}_{Di}).$$

Let us now introduce the notation:

$$\varphi_{Di} = \varphi(\mathbf{p}_{Di}), v_{Di} = v(\mathbf{p}_{Di}), \sigma_{Di} = \sigma(\mathbf{p}_{Di}),$$

$$\underline{\varphi}_D = \begin{bmatrix} \varphi_{D1} \\ \varphi_{D2} \\ \vdots \\ \vdots \\ \varphi_{Dm} \end{bmatrix}, \quad \underline{v}_D = \begin{bmatrix} v_{D1} \\ v_{D2} \\ \vdots \\ \vdots \\ v_{Dn} \end{bmatrix},$$

$$[\mathbf{L}_{DS}]_{ij} = \{Le\}_{\Delta S_j}(\mathbf{p}_{Di}),$$

$$[\mathbf{M}_{DS}]_{ij} = \{Me\}_{\Delta S_j}(\mathbf{p}_{Di}),$$

$$[\mathbf{M}^{\mathbf{t}}{}_{DS}]_{ij} = \{M^t e\}_{\Delta S_j}(\mathbf{p}_{Di}),$$

$$[\mathbf{N}_{DS}]_{ij} = \{Ne\}_{\Delta S_j}(\mathbf{p}_{Di}).$$

Returning back to the example integral equation (24), using this notation, approximations to the solution at the domain points can be determined by the following matrix-vector multiplication:

$$\underline{\varphi}_D \approx \mathbf{L}_{DS}\underline{\sigma}_S.$$

It can be observed that there is a general case in evaluating the required matrices; it is given a set of points and a description of the boundary, a method for evaluating the matrix components is required. The routine for doing this is lbem2.m.

lbem2.m

```
% function [L,M,Mt,N] = lbem2(m,p,vecp,n,vertpts,elemvert,p_on,
needL,needM,needMt,needN)
% Sets up the matrices required in the boundary element method
% for m (observation) points p with a vector (derivative) direction of
% vecp at the points p. The boundary is made up of n elements;
vertpts lists the coordinates
% of the edges of the elements and elemvert lists the pairs of vertices
% that define each element. p_on idicates whether the point p(i) lies
on the ith panel.
% needL,needM,needMt,needN indicate which matrices are needed
by being set to true.

function [L,M,Mt,N] =lbem2(m,p,vecp,n,vertpts,elemvert,p_on,
needL,needM,needMt,needN)

 [qa,qb]=vertices(n,vertpts,elemvert);

 for (i=1:n)
   for (j=1:n)
```

```
[L(i,j),M(i,j),Mt(i,j),N(i,j)]=l2lc(p(i,:),vecp(i,:),qa(j,:),qb(j,:),p_on&i==j,nee
dL,needM,needMt,needN);
    end
  end
```

Drawn from this general case is the special case in which the points lie at the centres of the elements that define the boundary. This special case is implemented by the routine lbem2_on.m.

lbem2_on.m

```
% function [L,M,Mt,N]
=lbem2(n,vertpts,elemvert,p_on,needL,needM,needMt,needN)
% Sets up the matrices required in the boundary element method
%  for a boundary made up of n elements. vertpts lists the coordinates
%  of the edges of the elements and elemvert lists the pairs of vertices
%  that define each element.needL,needM,needMt,needN indicate
which matrices
%  are needed by being set to true.

function [L,M,Mt,N] =lbem2_on(n,vertpts,elemvert,
needL,needM,needMt,needN)

  [qa,qb]=vertices(n,vertpts,elemvert);

  p=(qa+qb)/2;

  qbma=qb-qa;

  for(i=1:n)
    qlen(i)=norm(qbma(i,:));
  end

  for (i=1:n)
    normp(i,1)=-qbma(i,2)/qlen(i);
    normp(i,2)=qbma(i,1)/qlen(i);
  end

  [L,M,Mt,N]=lbem2(n,p,normp,n,vertpts,elemvert,true,
needL,needM,needMt,needN)
```

IX. Boundary Element Method

Having, developed all the building blocks we may now complete the coding for the boundary element methods. In section IV, two classes of boundary integral equation were introduced and these lead to two classes of boundary element method; the direct method and the indirect method.

Direct Method

For the chosen integral equation (9), the discrete analogue for collocation points on S is as follows:

$$[\ \mathbf{M}_{SS}+\tfrac{1}{2}\mathbf{I}+\mu\mathbf{N}_{SS}\ ]\ \underline{\varphi_S} = [\mathbf{L}_{SS}+\mu(\mathbf{M}^t_{SS}-\tfrac{1}{2}\mathbf{I})]\ \underline{v}_S \qquad (25)$$

with the general Robin boundary condition

$$\alpha_i\varphi_i + \beta_i v_i = f_i \ \text{ for } i=1..n\ . \qquad (26)$$

The solution of this kind of system is not a standard utility in Matlab/Freemat. Hence we introduce the routing gls.m that carries out row operations to rewrite a system like the above as a standard linear system and then solve by the inbuilt Matlab/Freemat matrix-vector solution method.

gls.m

```
% gls returns the solution x,y to a problem of the form
%                          A x = B y +c
% where A and B are n by n real matrices and c is a n-vector
% under the condition(s)
%
%          {\alpha}_i x_i + {\beta}_i y_i = f_i   for i=1..n.
%
% Clearly only one of {\alpha}_i or {\beta}_i can be zero for each i.
%
% The method employed involves forming a linear system of the form
Cz=d
% where the n by n matrix C and the vector d can be determined from
A,B
% and the {\alpha}_i and {\beta}_i. A standard LU factorisation
% solution method is then employed to return a solution. From this the
actual
% solutions x,y can be determined.

function [x,y]=gls(A,B,c,n,alpha,beta,F)
  gamma=norm(B,inf)/norm(A,inf);
  test=abs(beta)>abs(gamma*alpha);

  for (i=1:n)
    if (test(i))
      Fob=F(i)/beta(i);
      aob=alpha(i)/beta(i);
      for (j=1:n)
        c(j)=c(j)+Fob*B(j,i);
        B(j,i)=-aob*B(j,i);
      end
    else
      Foa=F(i)/alpha(i);
      boa=beta(i)/alpha(i);
      for (j=1:n)
        c(j)=c(j)-Foa*A(j,i);
        A(j,i)=-boa*A(j,i);
      end
    end
  end

  A=A-B;
  y=A\c;

  for (i=1:n)
    if (test(i))
      x(i)=(F(i)-alpha(i)*y(i))/beta(i);
    else
      x(i)=(F(i)-beta(i)*y(i))/alpha(i);
    end
  end

  for (i=1:n)
    if (test(i))
```

```
    temp=x(i);
    x(i)=y(i);
    y(i)=temp;
  end
end
x=x';
```

Once the solution on the boundary is found to (25), (26), the solution at any domain point can be found through integrating over the boundary using equation (6a).

Matlab/Freemat routine libem2.m returns the solution in the domain and on the boundary using the direct method.

libem2.m

```
% function [phi_D,phi_S,v_S] =
libem2(n_D,p_D,n_S,vertpts_S,elemvert_S,alpha_S,beta_S,f_S)
% Returns the solution phi_D at the n_D domain points p_D and at the
collocation points.
% The boundary is made up of n elements; vertpts lists the
coordinates
% of the edges of the elements and elemvert lists the pairs of vertices
% that define each element. alpha_S, beta_S and f_S determine the
Robin boundary condition.

function [phi_D,phi_S,v_S]
=libem2(n_D,p_D,n_S,vertpts_S,elemvert_S,alpha_S,beta_S,f_S)

% calculate phi_S, v_S, phi and v on the boundary
% calculate L_SS, M_SS, Mt_SS and N_SS, the discrete form of the
operators for the collocation points
 [L_SS,M_SS,Mt_SS,N_SS]
=lbem2_on(n_S,vertpts_S,elemvert_S,true,true,true,true);
 M_SSplus=M_SS+eye(n_S)/2;
 Mt_SSminus=Mt_SS-eye(n_S)/2;
 mu=norm(M_SSplus)/norm(N_SS);
 for (i=1:n_S)
   zero_S(i)=0.0;
 end
  [phi_S,v_S]=
 gls(M_SSplus+mu*N_SS,L_SS+mu*Mt_SSminus,zero_S,n_S,
alpha_S,beta_S,f_S);

% calculate phi_S, v_S, phi and v on the boundary
% calculate L_DS, M_DS, the discrete form of the operators for the
domain points
% dummy values set to vecp_D, since it is not used
 for (i=1:n_D)
   vecp_D(i,1)=1;
   vecp_D(i,2)=0;
 end
 [L_DS,M_DS,Mt_DS,N_DS]
=lbem2(n_D,p_D,vecp_D,n_S,vertpts_S,elemvert_S,false,true,true,fal
se,false);

 phi_D=L_DS*v_S-M_DS*phi_S;
```

Indirect Method

For the chosen integral equation (11e), the discrete analogue for collocation points on S is as follows:

$$\{(\mathbf{L}_{SS}+\mu(\mathbf{M}_{SS}-\tfrac{1}{2}\mathbf{I}))D_{\alpha}+((\mathbf{M}^{t}_{SS}+\tfrac{1}{2}\mathbf{I})+\mu\mathbf{N}_{SS})\mathbf{D}_{\beta}\}\underline{\sigma}_{\mu}=\underline{f} \quad (27)$$

where the general Robin boundary condition is included such that $\mathbf{D\alpha}$ and $\mathbf{D}_{\beta}$ are diagonal matrices with diagonal components α_i and β_i respectively.

The solution of equation (27) gives $\underline{\sigma}_{\mu}$, the discrete equivalent of the layer potential σ_{μ} on the boundary. The solution in the domain and on the boundary can then be found by direct integration using equations (11a) and (11b). The Matlab/Freemat routine libem2_indirect.m carries out the same function as libem2, but uses the direct method.

libem2_indirect.m

```
% function [phi_D,phi_S,v_S] =
libem2_indirect(n_D,p_D,n_S,vertpts_S,elemvert_S,alpha_S,beta_S,f
_S)
% Returns the solution phi_D at the n_D domain points p_D and at the
collocation points.
% The boundary is made up of n elements; vertpts lists the
coordinates
% of the edges of the elements and elemvert lists the pairs of vertices
% that define each element. alpha_S, beta_S and f_S determine the
Robin boundary condition.

function [phi_D,phi_S,v_S]
=libem2_indirect(n_D,p_D,n_S,vertpts_S,elemvert_S,alpha_S,beta_S,
f_S)

% calculate the boundary density function sigma
% calculate L_SS, M_SS, Mt_SS and N_SS, the discrete form of the
operators for the collocation points
 [L_SS,M_SS,Mt_SS,N_SS]
=lbem2_on(n_S,vertpts_S,elemvert_S,true,true,true,true);
 Mt_SSplus=Mt_SS+eye(n_S)/2;
 M_SSminus=M_SS-eye(n_S)/2;
 mu=norm(Mt_SSplus)/norm(N_SS);
 matrix1=L_SS+mu*M_SSminus;
 matrix2=Mt_SSplus+mu*N_SS;
 for (i=1:n_S)
   matrix(i,:)=alpha_S(i)*matrix1(i,:)+beta_S(i)*matrix2(i,:);
 end
 sigma=matrix\f_S';

% calculate the solution on the boundary (often not necessary)
 phi_S= (L_SS+mu*(M_SS-eye(n_S)/2))*sigma;
 v_S=(Mt_SS+eye(n_S)/2+mu*N_SS)*sigma;

% calculate L_DS, M_DS, the discrete form of the operators for the
domain points
% dummy values set to vecp_D, since it is not used
 for (i=1:n_D)
   vecp_D(i,1)=1;
   vecp_D(i,2)=0;
 end
 [L_DS,M_DS,Mt_DS,N_DS]
=lbem2(n_D,p_D,vecp_D,n_S,vertpts_S,elemvert_S,false,true,true,fal
se,false);

% calculate the solution at the domain points
```

```
phi_D=(L_DS+mu*M_DS)*sigma;
```

Weighting parameter

The value of the weighting parameter μ is arbitrary from the mathematical point of view. However, from the computational point of view, we need to avoid the equation becoming close to a first kind equation (that is if μ is small for the Dirichlet boundary condition) and we also want to avoid similar issues arising if we solve over something close to the *N* operator (that is if μ is large for the Neumann boundary condition).

The "size" or norm of the relevant matrices must also be taken into account when choosing a value for μ. A reasonable choice would therefore seem to be to choose a value of μ that balances the relevant matrices and therefore the relative contribution from the two underlying formulations. Out of the four matrices in the boundary solution, only $\mathbf{N}_{SS}$ has the property such that its norm is inversely proportional to the size of the panels; the norms of the other matrices stay approximately the same as the boundary panels become smaller. Hence in libem2 and libem2_indirect the underlying contributions from the two integral equations are balanced through applying the following values for μ:

$$\mu = \frac{\left\| \mathbf{M}^t_{SS} + \frac{1}{2}\mathbf{I} \right\|}{\left\| \mathbf{N}_{SS} \right\|} \quad \text{for the direct method and}$$

$$\mu = \frac{\left\| \mathbf{M}_{SS} + \frac{1}{2}\mathbf{I} \right\|}{\left\| \mathbf{N}_{SS} \right\|} \quad \text{for the indirect method.}$$

X. TEST PROBLEM AND RESULTS

Finally the direct and indirect boundary element methods are applied to a test problems and the results are observed. The test problem consists of a square with vertices (0,0), (0.1,0), (0.1, 0.1) and (0.1,0), as illustrated already in figure 4. The boundary condition is defined as illustrated in figure 5. A computational solution is sought at the points (0.025,0.025), (0.025,0.075), (0.075,0.075), (0.075,0.025) and (0.05,0.05), also illustrated in Figure 5.

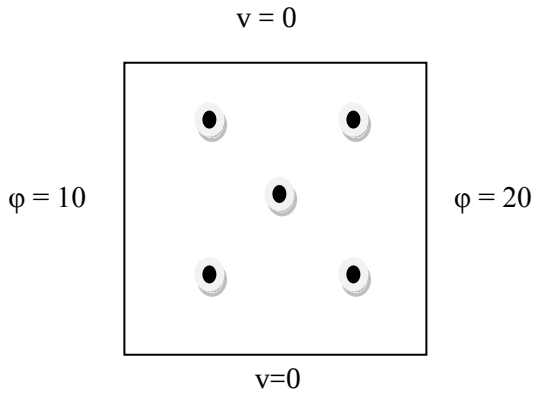

Figure 5. Boundary conditions and points where the solution is sought.

The exact solution to Laplace's equation in the domain for the problem described in figures 4 and 5 is φ=10+100*x*, so at the two points on the left φ=12.5, at the point in the middle φ=15 and at the two points on the right φ=17.5.

The program for computing the solution using the direct method is listed.

interiorsquaretest.m

```
% function [phi_D,phi_S,v_S] = interiorsquaretest
% Returns the solution phi_D, the solution at the domain points, phi_S and v_S,
% the solution on the boundary. The boundary is defined internally by
% calling 'square' and the points are defined by calling 'points'.
function [phi_D,phi_S,v_S]= interiorsquaretest
[vertpts,elemvert]=square
% boundary condition
a(1:8)=1;
a(9:16)=0;
a(17:24)=1;
a(25:32)=0;
b(1:8)=0;
b(9:16)=1;
b(17:24)=0;
b(25:32)=1;
F(1:8)=10;
F(9:16)=0;
F(17:24)=20;
F(25:32)=0;

% interior points
points=[ 0.025, 0.025; 0.025, 0.075; 0.05, 0.05; 0.075, 0.025; 0.075, 0.075];

% computer phi_D, phi at the interior points and phi_S,v_S, phi and v on the boundary
```

```
[phi_D, phi_S,v_S] =libem2(5,points,32,vertpts,elemvert,a,b,F);
```

For the indirect method we have the following program.

interiorsquaretest_indirect.m

```
% function [phi_D,phi_S,v_S] = interiorsquaretest_indirect
% Returns the solution phi_D, the solution at the domain points, phi_S and v_S,
% the solution on the boundary. The boundary is defined internally by
% calling 'square' and the points are defined by calling 'points'.
function [phi_D,phi_S,v_S]= interiorsquaretest_indirect
[vertpts,elemvert]=square
% boundary condition
a(1:8)=1;
a(9:16)=0;
a(17:24)=1;
a(25:32)=0;
b(1:8)=0;
b(9:16)=1;
b(17:24)=0;
b(25:32)=1;
F(1:8)=10;
F(9:16)=0;
F(17:24)=20;
F(25:32)=0;

% interior points
points=[ 0.025, 0.025; 0.025, 0.075; 0.05, 0.05; 0.075, 0.025; 0.075, 0.075];

% computer phi_D, phi at the interior points and phi_S,v_S, phi and v on the boundary
[phi_D,phi_S,v_S]
=libem2_indirect(5,points,32,vertpts,elemvert,a,b,F);
```

The results from the two methods are given in table 1.

TABLE I
COMPUTED AND EXACT RESULTS FOR SQUARE

point	exact	direct	indirect
(0.025,0.025)	12.5	12.4709	12.4891
(0.025,0.075)	12.5	12.4709	12.4891
(0.05,0.05)	15	15.0008	14.9939
(0.075,0.025)	17.5	17.5306	17.4927
(0.075,0.075)	17.5	17.5306	17.4927

REFERENCES

[1] L Wrobel. *The Boundary Element Method: Applications in Thermo-Fluids & Acoustics Vol. 1*. John Wiley and Sons, 2002.

[2] M.H. Aliabadi, *The Boundary Element Method: Vol. 2. Applications in Solids and Structures*, John Wiley and Sons Ltd, 2002

[3] R. Cunderlik , K. Mikula, and M. Mojzes, 3D BEM Application to Neumann Geodetic BVP using the collocation with linear basis functions. Proceedings of ALGORITMY, Conference on Scienti c Computing, pp. 268-275, 2002.
Available:
http://www.iam.fmph.uniba.sk/amuc/_contributed/algo2002/cunderlik/cunderlik.pdf

[4] R. Cunderlik , K. Mikula, AND M. Mojzes, A comparison of the variational solution to the Neumann geodetic boundary value problem with the geopotential model EGM-96. Contributions to Geophysics and Geodesy 34(3), 2004,
Available: http://www.math.sk/mikula/cmm.pdf

[5] Guror R, Olsen B, Kroese A, Cook F, Kumar S (2004). Evaluation of critical components of non-ceramic insulators in-service: role of defective interfaces, Power Systems Engineering Research Center, Arizona State University.
Available:
http://www.pserc.org/cgi-pserc/getbig/publicatio/reports/2004report/psercreport.pdf

[6] P. Lazic, H. Stefancic, H. Abraham ,The Robin Hood method: a novel numerical method for electrostatic problems based on a non-local charge transfer , *Journal of Computational Physics* Volume 213 , Issue 1 Pages: 117 - 140 ,2006
Available:
http://arxiv.org/PS_cache/physics/pdf/0411/0411192v1.pdf

[7] B. Lim, K, L. Venkatachalam, A, Jahangir, S. B. Johnson, S. J. Asirvatham, Concurrent Application of Charge Using a Novel Circuit Prevents Heat-Related Coagulum Formation During Radiofrequency Ablation, Journal of Cardiovascular Electrophysiology , pp1-8.

[8] F. Seydou, T. Seppanen, O. Ramahi, O. Numerical solution of 3D Laplace and Helmholtz equations for parallel scatterers , *Antennas and Propagation Society International Symposium*,IEEE Volume 2B, Issue , 3-8 July 2005 Page(s): 97 - 100 vol. 2B , 2005.

[9] A. VanderWyst, A. Christlieb, M. Sussman, I. D.

Boyd, Simulation of Charge and Mass Distributions of Indium Droplets Created by Field Emission, *American Institute of Aeronautics and Astronautics* Available: http://hpcc.engin.umich.edu/CFD/research/NGPD/Publications/AIAA-06-3560.pdf

[10] S. M. Kirkup, DC capacitor simulation by the boundary element method, *Communications in Numerical Methods in Engineering* **23**(9), 855 - 869, (2007). Available: http://www.east-lancashire-research.org.uk/AR/07/AR0702.pdf

[11] Matlab Available: http://www.mathworks.com

[12] Freemat Available : http://freemat.sourceforge.net

[13] S. M. Kirkup and D. J. Henwood, An empirical error analysis of the boundary element method applied to Laplace's equation, *Applied Mathematical Modelling*, Volume 18, Issue 1 , pp 32-38. 1994. Available: http://www.kirkup.info/papers

[14] L. M. Delves and J. L. Mohamed (1985). *Computational Methods for Integral Equations*, Cambridge University Press.

[15] C. A. Brebbia. *The Boundary Element Method for Engineers*. John Wiley and Sons, 1978.

[16] S. M. Kirkup. Fortran Codes for Computing the Discrete Helmholtz Integral Operators, Advances in Computational Mathematics, 9, pp391-409, 1998. Available: www.kirkup.info/papers

---

Received May 2008, published June 2008.

This article is published electronically on the www.east-lancashire-research.org.uk website.

Contact email: Stephen Kirkup: s.kirkup@blackburn.ac.uk