Boundary Element Methods for Engineers: Part I

Potential Problems

Roger Fenner

Roger Fenner

Boundary Element Methods for Engineers

Part I: Potential Problems

Boundary Element Methods for Engineers: Part I: Potential Problems
1st edition
© 2014 Roger Fenner & bookboon.com
ISBN 978-87-403-0732-0

Contents

We believe that energy suppliers should be renewable, too. We are therefore looking for enthusiastic new colleagues with plenty of ideas who want to join RWE in changing the world. Visit us online to find out what we are offering and how we are working together to ensure the energy of the future.

Click on the ad to read more

Preface

A few decades ago, the advent of high-speed electronic digital computers gave tremendous impetus to all numerical methods for solving engineering problems, and made it possible to solve with good accuracy many problems which previously could only be solved approximately. Finite difference methods, applied manually, gave way to finite element methods, which are still one of the most versatile and widely used, particularly in structural and solid mechanics. In thermofluids, methods of the finite volume type tend to be preferred.

Slower to develop have been boundary element methods, based on boundary integral equations. Initial development was largely in the hands of mathematicians, as the underlying mathematics are relatively sophisticated. It was engineers, however, who turned boundary element methods into practically useful and powerful techniques.

The purpose of this book is to serve as a deliberately simple introduction to boundary element methods applicable to a wide range of engineering problems. The mathematics are kept as simple as reasonably possible. Computer programs form an integral part of the boundary element approach and they are treated as such in the text. Several programs suitable for use on desktops or laptops are presented and described in detail and their uses are illustrated with the aid of a number of practical examples. Problems, with solutions, are provided at the ends of the chapters, for readers to solve for themselves.

The programming language used in the main text is Fortran. Although it is somewhat unfashionable these days for general programming purposes, Fortran is still very widely used in engineering computation. Matlab versions of the programs are also provided in Appendices. Full listings of all the programs, both Fortran and Matlab, are available for download here.

A prior knowledge of either Fortran or Matlab is desirable. The level of continuum mechanics, numerical analysis, matrix algebra, vector analysis and other mathematics employed is that normally taught in undergraduate engineering courses. The book is therefore suitable for engineering undergraduates and other students at an equivalent level. Postgraduates and practising engineers may also find it useful if they are comparatively new to boundary element methods.

The book is presented in two Parts. This Part I starts with a brief review of the problems encountered in engineering, showing that they of two broad types. It then describes boundary element treatments of problems of the potential type, using both constant and quadratic boundary elements. Part II is concerned with elastic stress analysis problems of the plane strain and plane stress types.

Imperial College London Professor Roger Fenner

Notation

The mathematical symbols commonly used in the main text are defined in the following list. In some cases particular symbols have more than one meaning in different parts of the book, although this should not cause any serious ambiguity.

A	area of a solution domain
$[A]$	square matrix
A_{ij}	coefficient of matrix $[A]$
a_1, a_2, a_3	constants in general boundary condition Equation 1.83
$[B]$	square matrix
B_{ij}	coefficient of matrix $[B]$
$[b]$	column vector of known coefficients
C	torsional couple
C_p	specific heat
$C_{xx}, C_{xy}, C_{yx}, C_{yy}$	free term constants
D	flexural rigidity of a flat plate
E	Young's modulus
e	strain
$F_{\mathrm{D}}, F_{\mathrm{P}}$	drag and pressure flow shape factors for downstream flow
f_1	function of position in Poisson's equation
f_2	function of position in biharmonic equation
G	shear modulus
g	acceleration due to gravity
g	heat generated per unit volume
H	height of a channel or solution domain in general
H	total rate of heat conduction
h	surface heat transfer coefficient
i	nodal point number
$\boldsymbol{i}$	unit vector in the x co-ordinate direction
J	Jacobian of transformation (from global to intrinsic co-ordinates)
j	nodal point number
$\boldsymbol{j}$	unit vector in the y co-ordinate direction
K	ratio of outer to inner radius for a cylinder
k	thermal conductivity
k	nodal point number
$\boldsymbol{k}$	unit vector in the z co-ordinate direction
L	maximum dimension of the solution domain
M	total number of boundary elements in a mesh
m	element number
N	total number of nodes in a mesh
N_c	shape function
n	direction of outward normal to the boundary of a solution domain
$\boldsymbol{n}$	outward normal vector to boundary

n_x, n_y	components of $\boldsymbol{n}$ in the x and y directions
$\hat{\boldsymbol{n}}$	unit outward normal vector to boundary
$\hat{n}_x, \hat{n}_y$	components of $\hat{\boldsymbol{n}}$ in the x and y directions
P	source/force point on a boundary
P_z	pressure gradient in the $z$co-ordinate direction
p	pressure
p	source/force point
Q	volumetric flow rate
Q	field point on a boundary
q	field point
R	a function of intrinsic co-ordinate ξ
r	radial co-ordinate
r	distance between a source/force point and a field point
$\boldsymbol{r}$	vector distance between points
$\hat{\boldsymbol{r}}$	unitvector in the direction between points
S	boundary of a domain
S	distance along a boundary
$\boldsymbol{S}$	vector tangent to a boundary
S	ratio between lengths of successive elements on a boundary segment
S_m	part of a boundary forming element m
s	distance along a solution domain boundary
T	temperature
T	traction kernel function
T_∞	remote temperature of surroundings in thermal convection
t	time
t	traction
U	displacement kernel function
u	displacement or velocity in the$x$direction
u_r	displacement in the radial direction
u_θ	displacement in theθdirection
$\bar{u}$	mean value of u
V_z	velocity component of a boundary in the z co-ordinate direction
v	displacement or velocity in the$y$direction
$\bar{v}$	mean value of v
W	width of a channel or solution domain in general
w	displacement or velocity in the$z$direction
X, Y	global Cartesian co-ordinates
$\bar{X}, \bar{Y}, \bar{Z}$	components of body forces per unit volume in the Cartesian co-ordinate directions
x, y, z	Cartesian co-ordinates
$[x]$	column vector of unknown quantities
α	coefficient of linear thermal expansion
α	coefficient in mixed boundary condition, Equation 2.32
β	coefficient in mixed boundary condition, Equation 2.32

γ	an angle
ΔT	change in temperature
ε	a small distance
η	local intrinsic co-ordinate within an element
θ	angle of rotation per unit length of a bar in torsion
θ	angular co-ordinate
θ	an angle
κ	permeability of a porous medium
μ	viscosity
ν	Poisson's ratio
ξ	local intrinsic co-ordinate within an element
π_{P}	dimensionless pressure gradient
π_{Q}	dimensionless flow rate
ρ	density
σ	stress
ϕ	velocity potential
ϕ	fundamental solution for potential
χ	stress function
ψ	stream function
ψ	potential
∇	grad operator
∇^2	harmonic (Laplacean) operator
∇^4	biharmonic operator

Subscripts

c	counter for nodes within an element
e	von Mises equivalent (stress)
i, j, k	nodal point numbers
L	solution to Laplace's equation
m	element number
n	direction of the outward normal to a boundary
PI	particular integral solution satisfying Poisson's equation
r	radial direction in polar co-ordinates
s	direction along a boundary
s	segment
T	total solution (Laplace plus particular integral)
x, y, z	Cartesian co-ordinate directions
θ	angular direction in polar co-ordinates
$1, 2$	inner and outer of two concentric circular boundaries
1, 2, 3	nodes of a quadratic element

Superscripts

$*$	effective value under plane stress conditions
$*$	modified quantity

Some Program Variable Names

The Fortran computer program variable names widely used in the programs and main text are defined in alphabetical order in the following list.

A	Coefficients of matrix $[A]$
AII	Matrix diagonal coefficient A_{ii}(potential problems)
AIIXX, AIIXY, AIIYX, AIIYY	Coefficients at the diagonal of matrix (elastic problems)
AIJ	Matrix coefficient A_{ij}
AK	First kernel function contributing to the matrix $[A]$ (potential problems)
AKXX, AKXY, AKYX, AKYY	First kernel functions contributing to the $[A]$ matrix (elastic problems)
ALPERP	Perpendicular distance from centre of curvature to mid point of a segment chord
ALPERP2	Square of ALPERP
ALPHA	Element values of constants in mixed boundary conditions
ALPHAN	Nodal point values of constants in mixed boundary conditions (quadratic elements)
ALPHASEG	Boundary segment values of constants in mixed boundary conditions
ALSEG	Length of a segment chord measured between its end points
ANG	Angular position of current end point on a curved boundary segment
ANGFIR	Angular position of first end point on a curved boundary segment
ANGSEG	Angle subtended at centre of curvature by a curved boundary segment
ANGSTORE	Angular positions of end points on curved boundary segments
AROW	Array storing element node contributions to the $[A]$ matrix (potential problems)
AROWX	Array storing element node contributions to the $[A]$ matrix (elastic problems)
AROWY	Array storing element node contributions to the $[A]$ matrix (elastic problems)
BDPSI	Coefficient of right hand side vector (matrix $[B]$ times vector of knowns)
BETA	Element values of constants in mixed boundary conditions
BETAN	Nodal point values of constants in mixed boundary conditions (quadratic elements)
BETASEG	Boundary segment values of constants in mixed boundary conditions
BIJ	Matrix coefficient B_{ij}
BK	Second kernel function contributing to the $[B]$ matrix (potential problems)
BKXX, BKXY, BKYX, BKYY	Second kernel functions contributing to the $[B]$ matrix (elastic problems)
BK2	Non-singular part of second kernel function when P is the current element node (potential problems)
BK2XX, BK2XY, BK2YX, BK2YY	Non-singular parts of second kernel functions when P is the current element node

	(elastic problems)
BROW	Array storing element node contributions to the $[B]$ matrix (potential problems)
BROWX	Array storing element node contributions to the $[B]$ matrix (elastic problems)
BROWY	Array storing element node contributions to the $[B]$ matrix (elastic problems)
BTX, BTY	Coefficients of right hand side column vector (matrix $[B]$ times the vector of knowns)
CASE	Alphanumeric plane stress or strain problem type
D	Perpendicular distance from P to the element containing node Q
DPSI	Nodal point values of the potential gradient solution to Laplace's equation
DPSIPI	Nodal point values of the particular integral potential gradient function
DPSIPIM	Values of the particular integral potential gradient at the nodes of each element
DPSISEG	Values of potential gradient applied as boundary conditions to the boundary segments
DPSISTORE	Temporary store for potential gradient
DPSIT	Nodal point values of total potential gradient (Laplace plus particular integral)
DRDN	Rate of change of radius with distance along normal to boundary
DUDZ	Rate of change of displacement u with ξ along element
DVDZ	Rate of change of displacement v with ξ along element
DZDE	Jacobian of transformation from intrinsic co-ordinate ξ to η
E	Young's modulus
EGL	Values of the intrinsic co-ordinate at the Gauss points (logarithmic quadrature)
ELENGTH	Lengths of the elements

ESS	Direct strain along boundary
ESTORE	Stored value of Young's modulus
ETA	Intrinsic co-ordinate η
EVX	x component of the vector along an element
EVY	y component of the vector along an element
F1	Constant function f_1 in Poisson's equation
FLOWELEM	Potential flow across an element
FLOWIN	Total potential flow into the domain
FLOWOUT	Total potential flow out of the domain
FLOWSEG	Flows of potential across the boundary segments
FXELEM	Force on an element in x direction
FXSEG	Total force on a boundary segment in x direction
FYELEM	Force on an element in y direction
FYSEG	Total force on a boundary segment in y direction
HX	Interval between points in the x direction used in domain integration
HY	Interval between points in the y direction used in domain integration
I	Node counter
I1, I2, I3	Numbers of the three nodes of a quadratic element
IBC	Type number of boundary conditions applied to the (constant) elements
IBCD	Counter for segments subject to applied potential gradient boundary conditions
IBCE	Type number of boundary conditions applied to the (quadratic) elements
IBCM	Counter for segments subject to applied mixed boundary conditions
IBCN	Type number of boundary conditions applied to the nodes (of quadratic elements)
IBCP	Counter for segments subject to applied potential boundary conditions
IBCPC	Counter for point displacement constraints
IBCS	Counter for segments subject to applied stress boundary conditions
IBCU	Counter for segments subject to applied displacement boundary conditions
IBOUND	Counter for boundaries
IC	Case number for logarithmic Gaussian quadrature
IDIRPC	Direction numbers of point displacement constraints
IEEND	Counter for element end points
IEP1	Counter for first end point of an element
IEP2	Counter for second end point of an element
IFIRST	Numbers of first nodes on the segments
IFLAG	Flag for ill-conditioning of the $[A]$ matrix
IGAUSS	Counter for Gauss points
IINT	Counter for internal points
ILAST	Numbers of last nodes on the segments
IN	Counter for nodes within an element

IROW	Number of row in the $[A]$ matrix
ISEG	Segment counter
ISEGBC	Segment numbers for a particular type of boundary condition
ISEGELEM	Segment numbers for elements
ISEGEND	Segment numbers for element end points
ISEGMAX	Number of last segment on current boundary
ISEGMIN	Number of first segment on current boundary
ISEND	Counter for boundary segment end points
IT	Number indicating type of Gaussian quadrature (normal or logarithmic)
IX	Counter for points in the x direction used in domain integration
IXMAX	Maximum value of IX
IY	Counter for points in the y direction used in domain integration
IYMAX	Maximum value of IY
J	Node counter
JACOB	Jacobian of transformation from global to local intrinsic ξ co-ordinate
JMAX	Maximum number of columns in the extended $[A]$ matrix
M	Element counter
M1	Numbers of the elements adjacent to the first node of each element
M3	Numbers of the elements adjacent to the third node of each element
MAXL	Maximum dimension of the solution domain
MAXNB	Maximum number of boundaries allowed by the array dimensions
MAXNEL	Maximum number of elements
MAXNEQN	Maximum number of equations
MAXNNP	Maximum number of nodal points allowed by the array dimensions
MAXNPC	Maximum number of point displacement constraints
MFIRST	Numbers of the first elements on the segments
MLAST	Numbers of the last elements on the segments
MMAX	Number of last element on current boundary
MMIN	Number of first element on current boundary
NBCD	Number of segments subject to applied potential gradient boundary conditions
NBCM	Number of segments subject to applied mixed boundary conditions
NBCP	Number of segments subject to applied potential boundary conditions
NBCPC	Number of point displacement constraints
NBCS	Number of segments subject to applied stress boundary conditions
NBCT	Total number of segments subject to applied boundary conditions
NBCU	Number of segments subject to applied displacement boundary conditions
NBOUND	Number of boundaries
NEEND	Number of element end points
NEL	Number of elements

NELB	Numbers of elements on the boundaries
NELSEG	Number of elements on current boundary segment
NEP1	Numbers of the first end points of the elements
NEP2	Numbers of the second end points of the elements
NEQN	Number of equations
NGAUSS	Number of Gauss points
NINT	Number of internal points
NNP	Number of nodal points
NNPB	Numbers of nodal points on each of the boundaries
NODE	Numbers of the nodes of the elements
NODEPC	Numbers of nodes subjected to point displacement constraints
NSEGB	Numbers of boundary segments on each of the boundaries
NSEGTOT	Total number of boundary segments
NU	Poisson's ratio
NX	Number of internal points in the x direction used in domain integration
NY	Number of internal points in the y direction used in domain integration
PI	π
PSI	Nodal point values of the potential solution to Laplace's equation
PSIBOT	Value of potential on the bottom edge of a rectangular domain
PSIIP	Laplace equation potential at an internal point

PSIIPT	Total potential at an internal point (Laplace plus particular integral)
PSILEFT	Value of potential on the left hand edge of a rectangular domain
PSIPI	Nodal point values of the particular integral potential function
PSIRIGHT	Value of potential on the right hand edge of a rectangular domain
PSISEG	Values of potential applied as boundary conditions to the boundary segments
PSIT	Nodal point values of total potential (Laplace plus particular integral)
PSITOP	Value of potential on the top edge of a rectangular domain
PSIVAL	Values of potential stored for domain integration
R1	Distance from point P to the first end of element containing node Q
R1X	x component of radius vector from P to the first end of element containing Q
R1Y	y component of radius vector from P to the first end of element containing Q
R2	Distance from point P to the second end of element containing node Q
R2X	x component of radius vector from P to the second end of element containing Q
R2Y	y component of radius vector from P to the second end of element containing Q
RATSEG	Ratio between successive element lengths on current boundary segment
RFN	Value of function $R(\xi)$
RSEG	Radius of curvature of current boundary segment
RX	Component in x direction of unit radius vector from P to Q
RY	Component in y direction of unit radius vector from P to Q
SD	Shape function derivatives for quadratic elements (normal quadrature)
SDL	Shape function derivative values for quadratic elements (logarithmic quadrature)
SF	Shape function values for quadratic elements (normal quadrature)
SFL	Shape function values for quadratic elements (logarithmic quadrature)
SFN	Shape function value at a Gauss point
SIGE	Nodal point values of von Mises equivalent stress
SIGNN	Nodal point values of direct stress normal to boundary
SIGNNSEG	Boundary segment values of direct stress normal to boundary
SIGSN	Nodal point values of shear stress along boundary
SIGSNSEG	Boundary segment values of shear stress along boundary
SIGSS	Nodal point values of direct stress along boundary
STORE	Stored values in the boundary condition application process
TITLE	Alphanumeric title for the problem (maximum 80 characters)
TMX	Element nodal point values of traction in x direction
TX	Nodal point values of traction in x direction
TMY	Element nodal point values of traction in y direction
TY	Nodal point values of traction in y direction
U	Nodal point values of displacement in x direction
UELEM	Element values of displacement in x direction
UNGX	x component of the unit normal at a Gauss point
UNGY	y component of the unit normal at a Gauss point

UNMX	x components of the unit normals at the nodes of each element
UNMY	y components of the unit normals at the nodes of each element
UNX	x components of the unit normals at the nodes
UNY	y components of the unit normals at the nodes
USEG	Boundary segment values of displacement in x direction
UV	Nodal point values of computed displacements (or tractions)
V	Nodal point values of displacement in y direction
VELEM	Element values of displacement in y direction
VSEG	Boundary segment values of displacement in y direction
WG	Values of the Gaussian weighting factors (normal quadrature)
WGL	Values of the Gaussian weighting factors (logarithmic quadrature)
XC	x co-ordinate of the origin for the particular integral function
XCENT	x co-ordinate of the centre of curvature of a curved boundary segment
XEEND	x co-ordinates of the element end points
XFIRST	x co-ordinate of first end point of current boundary segment
XINT	x co-ordinate of an internal point
XLAST	x co-ordinate of last end point of current boundary segment
XMID	x co-ordinate of the mid point between the ends of a curved segment
XNODE	x co-ordinates of the nodes
XP	x co-ordinate of point
XPOINT	Global x co-ordinate of an internal point
XQ	x co-ordinate of Gauss point
XSEND	x co-ordinates of the boundary segment end points
XX	x co-ordinate relative to the origin for the particular integral
YC	y co-ordinate of the origin for the particular integral function
YCENT	y co-ordinate of the centre of curvature of a curved boundary segment
YEEND	y co-ordinates of the element end points
YFIRST	y co-ordinate of first end point of current boundary segment
YINT	y co-ordinate of an internal point
YINTGL	Values of y direction integrals stored for domain integration
YLAST	y co-ordinate of last end point of current boundary segment
YMID	y co-ordinate of the mid point between the ends of a curved segment
YNODE	y co-ordinates of the nodes
YP	y co-ordinate of point P
YPOINT	Global co-ordinate of an internal point
YQ	y co-ordinate of Gauss point Q
YSEND	y co-ordinates of the boundary segment end points
YY	y co-ordinate relative to the origin for the particular integral
ZETA	Intrinsic co-ordinate ξ
ZG	Values of the intrinsic co-ordinate ξ at the Gauss points (normal quadrature)

1 Introduction

The aim of this chapter is to review many of the types of problems encountered in engineering, particularly those involving solid components or fluid flows. Problems from widely diverse branches of engineering are often governed by mathematically identical equations. This means that they can be solved using the same methods. Such methods may be either analytical where the problem is sufficiently simple, or computational in more general cases.

Traditional methods of solving engineering problems involved the use of algebraic formulae derived from approximate analyses of the physical quantities of interest. For example, the distributions of displacements and stresses in a solid component, or of velocities, pressure and temperature in a flowing fluid, might be required. Such a problem had to be simplified to the point where an analytical solution could be obtained which, it was hoped, was a sufficiently good approximation to the solution of the real problem. With the advent of modern high-speed digital computers, the focus of engineering analysis has moved towards more versatile and accurate numerical methods.

Well-established types of numerical technique include the finite difference, finite volume and finite element methods. In all these methods, the distributions of the variables are obtained as approximate values at a large, but finite, number of discrete points over the entire region of interest, the solution domain. Despite some significant differences these methods are very similar in this important respect, and may be described as domain methods. In boundary methods, on the other hand, the primary results are the distributions of the variables on the boundary of the solution domain only, again defined in terms of values at a finite number of points. The most widely used name for such boundary-based techniques is the boundary element method. Another is the boundary integral equation (BIE) method, reflecting the fact that the differential equation governing the physical process throughout the solution domain is transformed into an integral equation applied to the boundary alone.

The object of this book is to provide a simple treatment of the fundamentals, numerical implementation and some of the applications of boundary element methods to continuum mechanics problems in engineering. Part I is concerned with problems which fall into the category known as potential problems, while Part II deals with stress analysis of elastic solids.

1.1 Continuum Mechanics Problems

The concept of treating solids and fluids as though they are continuous media, rather than composed of discrete molecules, is one that is widely used in most branches of engineering. In this section, the basic equations of continuum mechanics are reviewed.

The mathematical equations for the deformation of solids are very similar in form to those for the flow of fluids. The essential difference is that solid behaviour is described in terms of displacements and strains, but for fluids velocities and strain rates are the corresponding variables. Equations are presented here for Cartesian co-ordinate systems, first in their full three-dimensional forms. Problems considered in this book are two-dimensional, allowing further simplification of the equations.

1.1.1 Stresses and strains

The symbol σ is used to indicate stress, and individual components are indicated by double subscripts

direct stresses: $\sigma_{xx}, \sigma_{yy}, \sigma_{zz}$

shear stresses: $\sigma_{xy}, \sigma_{yz}, \sigma_{zx}, \sigma_{yx}, \sigma_{zy}, \sigma_{xz}$

The first subscript defines the direction of the stress component, and the second one denotes the direction of the outward normal to the surface on which it acts, as shown in Figure 1.1. For any small element of material to be in equilibrium in a rotational sense, the shear stresses must be complementary

$$\sigma_{xy} = \sigma_{yx}\,, \qquad \sigma_{yz} = \sigma_{zy}\,, \qquad \sigma_{zx} = \sigma_{xz} \tag{1.1}$$

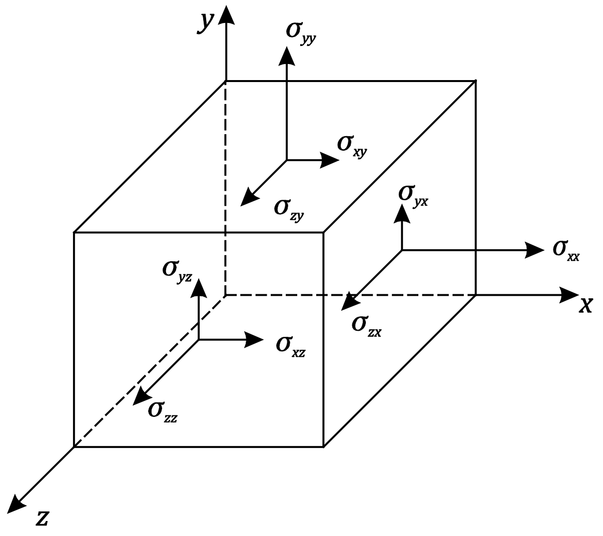


Figure 1.1 Cartesian stress components

The components of solid displacement or fluid velocity in the *x*, *y* and *z* co-ordinate directions are denoted by *u*, *v* and *w*. Following the same double subscript notation, the direct and shear components of strain or strain rate may be defined by

$$e_{xx} = \frac{\partial u}{\partial x}, \qquad e_{yy} = \frac{\partial v}{\partial y}, \qquad e_{zz} = \frac{\partial w}{\partial z} \tag{1.2}$$

$$e_{xy} = e_{yx} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \tag{1.3}$$

$$e_{yz} = e_{zy} = \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y} \tag{1.4}$$

$$e_{zx} = e_{xz} = \frac{\partial w}{\partial x} + \frac{\partial u}{\partial z} \tag{1.5}$$

1.1.2 Equilibrium equations

If inertia effects are sufficiently small to be neglected, the partial differential equations of equilibrium in the three Cartesian co-ordinate directions are

$$\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xy}}{\partial y} + \frac{\partial \sigma_{xz}}{\partial z} + \bar{X} = 0 \tag{1.6}$$

$$\frac{\partial \sigma_{yx}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} + \frac{\partial \sigma_{yz}}{\partial z} + \bar{Y} = 0 \tag{1.7}$$

$$\frac{\partial \sigma_{zx}}{\partial x} + \frac{\partial \sigma_{zy}}{\partial y} + \frac{\partial \sigma_{zz}}{\partial z} + \bar{Z} = 0 \tag{1.8}$$

where $\bar{X}$, $\bar{Y}$ and $\bar{Z}$ are the components at the relevant point in the material of any body forces per unit volume acting in the co-ordinate directions. The force of gravity is the most common example.

While inertia effects are not normally important in solid mechanics, they can only be ignored in fluids if the flows are 'slow' enough to be dominated by pressure and viscous forces. Otherwise, terms would have to be added to Equations 1.6 to 1.8. For example, Equation 1.6 becomes

$$\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xy}}{\partial y} + \frac{\partial \sigma_{xz}}{\partial z} + \bar{X} = \rho\left(\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z}\right) \tag{1.9}$$

where t is time ρ and is the density of the fluid. This, together with the equivalent equations for the other two directions, are known as the Navier-Stokes equations.

1.1.3 Energy equation

If thermal convection is ignored, the equation for conservation of energy within a solid or fluid material is

$$k\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2}\right) + g = \rho C_p \frac{\partial T}{\partial t} \tag{1.10}$$

where T is the temperature, g is the heat per unit volume generated at the point of interest, k is the thermal conductivity, and C_p is the specific heat of the material. Heat may be generated, for example, by mechanical work, and

$$g = \sigma_{xx}e_{xx} + \sigma_{yy}e_{yy} + \sigma_{zz}e_{zz} + \sigma_{xy}e_{xy} + \sigma_{yz}e_{yz} + \sigma_{zx}e_{zx} \tag{1.11}$$

Another example of heat generation in a material is when an electrical current is passed through a conductor. For a fluid in which heat is transferred by convection, the following term would have to be added to the right-hand side of Equation 1.10

$$\rho C_p \left(u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y} + w\frac{\partial T}{\partial z} \right) \tag{1.12}$$

1.1.4 Compatibility equations

Strains or strain rates should be compatible with each other. The physical interpretation of this statement is that no discontinuities such as holes or overlaps of material should exist, which in mathematical terms means that *u, v* and *w* should be continuous and differentiable functions of *x, y* and *z*. Using the six strain (rate) definitions from Equations 1.2 to 1.5, the following six relationships can be obtained by using differentiation to eliminate *u, v* and *w*.

$$\frac{\partial^2 e_{xx}}{\partial y^2} + \frac{\partial^2 e_{yy}}{\partial x^2} = \frac{\partial^2 e_{xy}}{\partial x \partial y} \tag{1.13}$$

$$\frac{\partial^2 e_{yy}}{\partial z^2} + \frac{\partial^2 e_{zz}}{\partial y^2} = \frac{\partial^2 e_{yz}}{\partial y \partial z} \tag{1.14}$$

$$\frac{\partial^2 e_{zz}}{\partial x^2} + \frac{\partial^2 e_{xx}}{\partial z^2} = \frac{\partial^2 e_{zx}}{\partial z \partial x} \tag{1.15}$$

$$2\frac{\partial^2 e_{xx}}{\partial y \partial z} = \frac{\partial}{\partial x}\left(-\frac{\partial e_{yz}}{\partial x} + \frac{\partial e_{zx}}{\partial y} + \frac{\partial e_{xy}}{\partial z} \right) \tag{1.16}$$

$$2\frac{\partial^2 e_{yy}}{\partial z \partial x} = \frac{\partial}{\partial y}\left(\frac{\partial e_{yz}}{\partial x} - \frac{\partial e_{zx}}{\partial y} + \frac{\partial e_{xy}}{\partial z} \right) \tag{1.17}$$

$$2\frac{\partial^2 e_{zz}}{\partial x \partial y} = \frac{\partial}{\partial z}\left(\frac{\partial e_{yz}}{\partial x} + \frac{\partial e_{zx}}{\partial y} - \frac{\partial e_{xy}}{\partial z} \right) \tag{1.18}$$

Such compatibility equations rarely appear in fluid mechanics analyses, because these are normally set up with velocities as the unknowns. Compatibility of strain rates is then satisfied automatically. On the other hand, problems of solid stress analysis are often set up with stresses as the unknowns, in which case it is necessary to ensure that the strains are compatible.

1.1.5 Continuity equation

Conservation of mass in a fluid flow is ensured by the continuity equation. For a constant density fluid it becomes the incompressibility condition

$$e_{xx} + e_{yy} + e_{zz} = 0 \tag{1.19}$$

1.1.6 Constitutive equations

The connections between stresses on the one hand and strains or strain rates on the other are defined by constitutive equations which involve the appropriate material properties. For present purposes solids are assumed to be purely elastic and fluids to be purely viscous. The existence of viscoelastic materials is ignored. In general the properties of a material may vary. If they are independent of position the material is said to be homogeneous. If they are independent of direction at all points it is said to be isotropic. If they are independent of the stress or strain (rate) applied the material is said to be linear.

Strains in an elastic solid may be produced both by applying stresses and by changing the material temperature. For a homogeneous, isotropic and linearly elastic solid the constitutive equations are

$$e_{xx} = \frac{1}{E}\left[\sigma_{xx} - \nu\left(\sigma_{yy} + \sigma_{zz}\right)\right] + \alpha\Delta T \tag{1.20}$$

$$e_{yy} = \frac{1}{E}\left[\sigma_{yy} - \nu(\sigma_{zz} + \sigma_{xx})\right] + \alpha\Delta T \tag{1.21}$$

$$e_{zz} = \frac{1}{E}\left[\sigma_{zz} - \nu\left(\sigma_{xx} + \sigma_{yy}\right)\right] + \alpha\Delta T \tag{1.22}$$

$$e_{xy} = \frac{\sigma_{xy}}{G} = \frac{2(1+\nu)}{E}\sigma_{xy} \tag{1.23}$$

$$e_{yz} = \frac{\sigma_{yz}}{G} = \frac{2(1+\nu)}{E}\sigma_{yz} \tag{1.24}$$

$$e_{zx} = \frac{\sigma_{zx}}{G} = \frac{2(1+\nu)}{E}\sigma_{zx} \tag{1.25}$$

In these equations E is Young's modulus, G is the shear modulus, ν is Poisson's ratio, α is the coefficient of linear thermal expansion and ΔT is the temperature change.

A Newtonian fluid is homogeneous, isotropic and linear, and under laminar flow conditions the constitutive equations for its flow are

$$\sigma_{xx} = -p + 2\mu e_{xx}, \quad \sigma_{yy} = -p + 2\mu e_{yy}, \quad \sigma_{zz} = -p + 2\mu e_{zz} \tag{1.26}$$

$$\sigma_{xy} = \mu e_{xy}, \quad \sigma_{yz} = \mu e_{yz}, \quad \sigma_{zx} = \mu e_{zx} \tag{1.27}$$

where p is the hydrostatic pressure in the fluid and μ is the viscosity.

1.2 Some Practical Engineering Problems

The following examples demonstrate the application of the above equations to practical problems. It is not the details of all the examples that are important, but rather the similarities between the resulting differential equations.

1.2.1 Downstream viscous flow in a uniform channel

Figure 1.2 shows the cross-section of a uniform flow channel, which for present purposes is taken to be rectangular in shape. The lower three sides of the channel are assumed to be stationary while the top surface moves with a velocity V_z in the downstream z-direction normal to the cross-section shown. The velocity of flow, w, is also in this direction. Because the channel is uniform, there are no variations of either geometry or physical variables in the downstream direction, with the exception that pressure is a linear function of z, but constant across the cross-section. Substituting the expressions for stresses given in Equations 1.26 and 1.27 into equilibrium Equation 1.8 and ignoring body forces, the governing differential equation for downstream velocity w becomes

$$\frac{\partial}{\partial x}\left(\mu\frac{\partial w}{\partial x}\right)+\frac{\partial}{\partial y}\left(\mu\frac{\partial w}{\partial y}\right)=\frac{\partial p}{\partial z}=P_z \tag{1.28}$$

Because the viscosity is constant

$$\frac{\partial^2 w}{\partial x^2}+\frac{\partial^2 w}{\partial y^2}=\nabla^2 w=\frac{P_z}{\mu} \tag{1.29}$$

The mathematical symbol ∇^2 is a partial differential operator, and is sometimes referred to as the Laplacian. Problems governed by this type of equation may be referred to as either potential or harmonic.

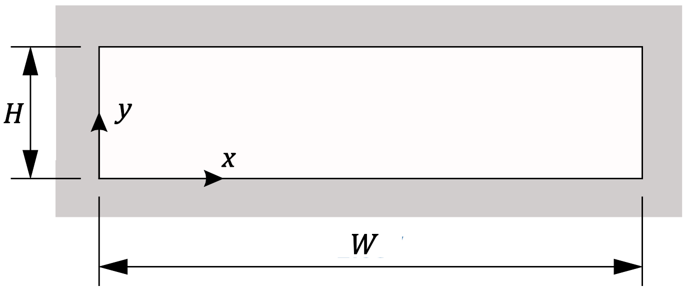


Figure 1.2 Rectangular channel geometry and co-ordinates

If there is no slip between the fluid and the solid walls of the channel, the boundary conditions for velocity are

$$w = 0 \text{ on } x = 0,\ x = W \text{ and } y = 0; \qquad w = V_z \text{ on } y = H \tag{1.30}$$

The downstream volumetric flow rate is found by integration of the velocity profile over the cross-section

$$Q = \int_0^H \int_0^W w \,\mathrm{d}x\,\mathrm{d}y \tag{1.31}$$

1.2.2 Torsion of a prismatic bar

Figure 1.3 shows the cross-section of a prismatic bar, which for present purposes is elliptical in shape. The ends of the bar are twisted relative to each other about the *z*-axis normal to the cross-section shown. The St. Venant theory of torsion assumes that the deformation of the bar is a combination of a rotation of the cross-section and a warping in the *z*-direction, although there is no warping in the particular case of a bar of circular cross-section. The angle of rotation, in the anticlockwise direction, is assumed to be θ per unit length of the bar and the cross-section illustrated is assumed to be at a distance *z* from the position of zero rotation. Hence, the displacements of the typical point P shown in the figure are

$$u = -\theta z y\,, \qquad v = \theta z x\,, \qquad w = w(x, y) \tag{1.32}$$

x and *y* being the co-ordinates of P. The displacement *w* describes the warping of the cross-section.

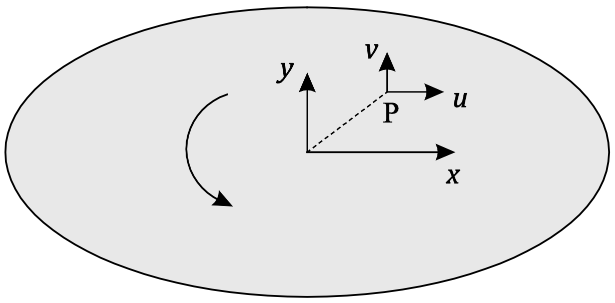


Figure 1.3 Elliptical cross-section of a prismatic bar

Using the definitions of strains given in Equations 1.2 to 1.5

$$e_{xx} = e_{yy} = e_{zz} = e_{xy} = 0$$

$$e_{yz} = \theta x + \frac{\partial w}{\partial y}, \qquad e_{zx} = \frac{\partial w}{\partial x} - \theta y \tag{1.33}$$

With displacements being chosen as the unknowns, compatibility Equations 1.13 to 1.18 are automatically satisfied. In the absence of temperature change, constitutive Equations 1.20 to 1.25, give the only nonzero stress components as $\sigma_{yz} = Ge_{yz}$ and $\sigma_{zx} = Ge_{zx}$. Among the equilibrium Equations 1.6 to 1.8, only the last is relevant, and ignoring body forces becomes

$$\frac{\partial \sigma_{zx}}{\partial x} + \frac{\partial \sigma_{zy}}{\partial y} = 0 \tag{1.34}$$

Consequently

$$\frac{\partial}{\partial x}\left[G\left(\frac{\partial w}{\partial x} - \theta y\right)\right] + \frac{\partial}{\partial y}\left[G\left(\theta x + \frac{\partial w}{\partial y}\right)\right] = 0$$

$$\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} = \nabla^2 w = 0 \tag{1.35}$$

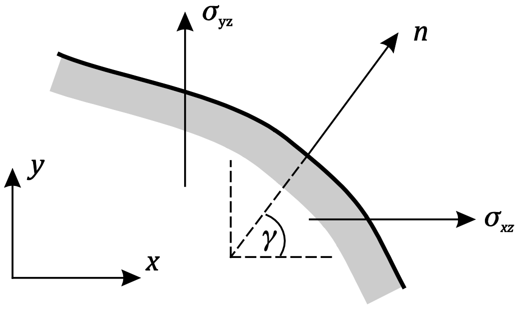


Figure 1.4 Part of the boundary of a cross-section of a prismatic bar

The boundary conditions for this torsion problem are for the shear stresses acting on the outer surface of the bar to be zero. At a particular point on the surface let n be the direction of the outward normal there, in which case the surface stress σ_{zn} is zero, and so is the complementary stress σ_{nz}. Consequently

$$\sigma_{nz} = \sigma_{xz}\cos\gamma + \sigma_{yz}\sin\gamma = 0 \tag{1.36}$$

$$\left(\frac{\partial w}{\partial x} - \theta y\right)\cos\gamma + \left(\frac{\partial w}{\partial y} + \theta x\right)\sin\gamma = 0 \tag{1.37}$$

where γ is the angle between the direction of n and the x-axis. Boundary conditions for displacement of this type are not straightforward to apply, particularly for asymmetrical bar cross-sections where the position of the axis of rotation is not obvious.

This example provides a good illustration of the difficulties of formulating a problem in terms of displacements when the boundary conditions are defined in terms of stresses. Consequently, solid mechanics problems are sometimes more conveniently formulated in using stress functions. In this torsion problem stress function χ can be defined to automatically satisfy the equilibrium Equation 1.34

$$\sigma_{zx} = \frac{\partial \chi}{\partial y}, \qquad \sigma_{yz} = -\frac{\partial \chi}{\partial x} \tag{1.38}$$

The relevant shear strains are

$$e_{zx} = \frac{1}{G}\frac{\partial \chi}{\partial y}, \qquad e_{yz} = -\frac{1}{G}\frac{\partial \chi}{\partial x} \tag{1.39}$$

which must the satisfy compatibility Equations 1.16 and 1.17, meaning that

$$\frac{\partial e_{zx}}{\partial y} - \frac{\partial e_{yz}}{\partial x} = constant \tag{1.40}$$

The value of the constant can be found from Equations 1.33 as -2θ. Then, substituting the strains defined by Equations 1.39 into Equation 1.40, the final differential equation for χ is obtained as

$$\frac{\partial^2 \chi}{\partial x^2} + \frac{\partial^2 \chi}{\partial y^2} = \nabla^2 \chi = -2G\theta \tag{1.41}$$

Because achieving your dreams is your greatest challenge. IE Business School's Master in Management taught in English, Spanish or bilingually, trains young high performance professionals at the beginning of their career through an innovative and stimulating program that will help them reach their full potential.

- Choose your area of specialization.
- Customize your master through the different options offered.
- Global Immersion Weeks in locations such as Rio de Janeiro, Shanghai or San Francisco.

Because you change, we change with you.

www.ie.edu/master-management | mim.admissions@ie.edu

Click on the ad to read more

In mathematical terms this is very similar to Equation 1.35. The zero shear stress boundary condition $\sigma_{nz} = 0$ is obtained when the gradient of the stress function along the boundary is zero, in other words when the value of is constant along the boundary. Because it is not the absolute value of the stress function, but only its derivatives, which determine the stress distribution, $\chi = 0$ is a suitable boundary condition, and much simpler to apply than Equation 1.37 for warping displacement.

Given the shear stress distributions over the cross-section, the magnitude of the couple required to twist the bar may be found by integration

$$C = \iint \left(x\sigma_{yz} - y\sigma_{zx}\right) \, \mathrm{d}x \, \mathrm{d}y \tag{1.42}$$

Introducing the stress function already defined

$$\begin{aligned} C &= -\iint \left(x\frac{\partial \chi}{\partial x} + y\frac{\partial \chi}{\partial y}\right) \, \mathrm{d}x \, \mathrm{d}y \\ &= -\iint \left[\frac{\partial}{\partial x}(x\chi) + \frac{\partial}{\partial y}(y\chi)\right] \, \mathrm{d}x \, \mathrm{d}y + 2\iint \chi \, \mathrm{d}x \, \mathrm{d}y \end{aligned}$$

and applying Green's theorem to the first of these two integrals

$$C = -\oint (x\chi \, \mathrm{d}y - y\chi \, \mathrm{d}x) + 2\iint \chi \, \mathrm{d}x \, \mathrm{d}y$$

The line integration in this new integral is carried out around the boundary of the bar. Because $\chi = 0$ on this boundary, the expression for torsional couple becomes

$$C = 2\iint \chi \, \mathrm{d}x \, \mathrm{d}y \tag{1.43}$$

1.2.3 Ideal fluid flow

The 'ideal' fluid model of flow can be used to describe the motion of real fluids in areas away from the wakes and boundary layers that occur near solid boundaries. Close to such boundaries viscous effects are significant, but an ideal fluid, which is homogeneous, isotropic and incompressible, is also assumed to have negligible viscosity. The flow of an ideal fluid is governed by pressure forces and fluid inertia. For two-dimensional flow in the $x - y$ plane (with $w = 0$ everywhere) the equilibrium equations in the form of Equation 1.9 for steady (time-independent) flow with negligible external body forces are

$$-\frac{\partial p}{\partial x} = \rho\left(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right) \tag{1.44}$$

$$-\frac{\partial p}{\partial y} = \rho\left(u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y}\right) \tag{1.45}$$

The two-dimensional form of the continuity Equation 1.19 is

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0 \tag{1.46}$$

which is automatically satisfied by stream function ψ, defined by

$$u=\frac{\partial \psi}{\partial y}\,, \qquad v=-\frac{\partial \psi}{\partial x} \tag{1.47}$$

Equations 1.44 and 1.45 can be rearranged as

$$-\frac{\partial}{\partial x}\left(\frac{p}{\rho}+\frac{u^2+v^2}{2}\right)=v\left(\frac{\partial u}{\partial y}-\frac{\partial v}{\partial x}\right) \tag{1.48}$$

$$-\frac{\partial}{\partial y}\left(\frac{p}{\rho}+\frac{u^2+v^2}{2}\right)=u\left(\frac{\partial v}{\partial x}-\frac{\partial u}{\partial y}\right) \tag{1.49}$$

and these in general require that

$$\frac{\partial u}{\partial y}-\frac{\partial v}{\partial x}=0 \tag{1.50}$$

Using Equations 1.47 for velocities expressed in terms of stream function, this becomes

$$\nabla^2\psi=0 \tag{1.51}$$

Given the stream function distribution, the pressure distribution can be determined from

$$\frac{p}{\rho}+\frac{u^2+v^2}{2}=constant \tag{1.52}$$

An alternative approach to ideal fluid flow problems is to define a velocity potential $\emptyset$

$$u=-\frac{\partial \emptyset}{\partial x}\,, \qquad v=-\frac{\partial \emptyset}{\partial y} \tag{1.53}$$

which automatically satisfies Equation 1.50. In order to satisfy continuity Equation 1.46 it is necessary that

$$\nabla^2\emptyset=0 \tag{1.54}$$

The same mathematical form of equation is obtained, and the fact that it is for velocity potential helps to explain why problems governed by this type of equation may be referred to as potential problems.

1.2.4 Diffusion and potential problems

Many physical processes involve diffusion and variables which are effectively a type of potential (including velocity potential in the ideal fluid flow problem above). An example is thermal conduction in solid or fluid materials, governed by the energy Equation 1.10. For steady heat conduction in the $x - y$ plane this reduces to

$$\nabla^2 T = -\frac{g}{k} \tag{1.55}$$

Boundary conditions for the temperature may be of various types. The simplest is when the temperature at the boundary or part of the boundary is known. On the other hand, a boundary which is thermally insulated, with no heat transfer across it, is subject to the derivative condition

$$\frac{\partial T}{\partial n} = 0 \tag{1.56}$$

As usual n is the direction of the outward normal to the boundary. A more general type of condition is applicable when the heat conducted towards the boundary is convected away from the surface into a fluid in contact with it

$$-k\frac{\partial T}{\partial n} = h(T - T_\infty) \tag{1.57}$$

where h is the heat transfer coefficient and T_∞ is the remote temperature of the surrounding fluid.

Other examples of diffusion and potential problems are provided by electrical conduction, electrostatic potential in insulating media, fluid flow in porous media, and neutron diffusion in a nuclear reactor. Take the flow of a fluid in a porous medium, such as water seeping through rock or soil. The flow is Newtonian and at a microscopic level the volumetric flow rate along an individual passage in the soil is proportional to the local pressure gradient along the passage. Then at a macroscopic level the mean velocity in a given direction at a particular position, which is the sum of the flow rates along individual passages, is also proportional to the pressure gradient in that direction at the chosen position. Therefore, for two-dimensional flow in the $x - y$ plane

$$\bar{u} = -\kappa \frac{\partial p}{\partial x}, \qquad \bar{v} = -\kappa \frac{\partial p}{\partial y} \tag{1.58}$$

where k is the permeability of the soil, and pressure is effectively a form of velocity potential. As in the case of ideal fluid flow considered above, a stream function ψ may be defined which automatically satisfies continuity

$$\bar{u} = \frac{\partial \psi}{\partial y}, \qquad \bar{v} = -\frac{\partial \psi}{\partial x} \tag{1.59}$$

Differentiation can be used to eliminate pressure from Equations 1.58 to give

$$\frac{\partial \bar{u}}{\partial y} - \frac{\partial \bar{v}}{\partial x} = \nabla^2 \psi = 0 \tag{1.60}$$

Boundary conditions are generally of the form of either prescribed constant values of for boundaries impervious to flow, or zero derivatives normal to boundaries where the pressure is constant.

An alternative approach to porous media flow problems is to work in terms of pressure as a velocity potential. Substituting Equations 1.58 into continuity Equation 1.46 gives

$$\nabla^2 p = 0 \tag{1.61}$$

1.2.5 Plane strain

Both plane strain and plane stress are important types of two-dimensional solid loading and deformation. Figure 1.5 shows a solid body whose cross-section is does not vary in the -direction. If its length in this direction is large, the typical cross-section OABC shown can be treated as being far from the ends. Provided that the surface loadings applied to the body are in the $x - y$ plane, the state of strain created at OABC is two-dimensional, independent of and with displacement $w = 0$.

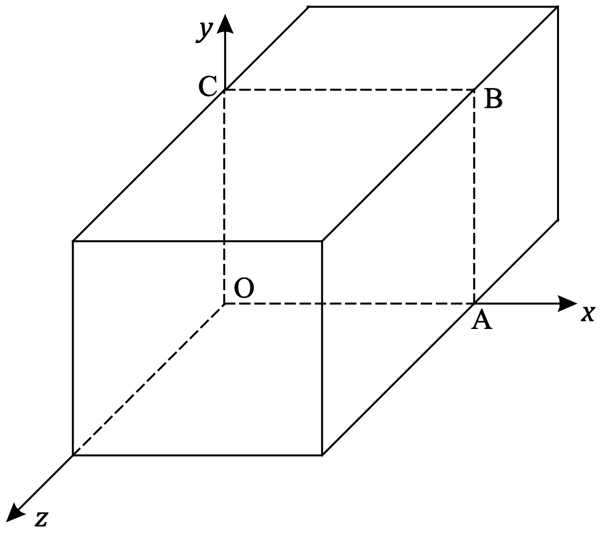


Figure 1.5 The plane strain approximation for a prismatic solid body

Ignoring body forces, equilibrium Equations 1.6 to 1.8 become

$$\frac{\partial\sigma_{xx}}{\partial x}+\frac{\partial\sigma_{xy}}{\partial y}=0 \tag{1.62}$$

$$\frac{\partial\sigma_{yx}}{\partial x}+\frac{\partial\sigma_{yy}}{\partial y}=0 \tag{1.63}$$

$$\frac{\partial\sigma_{zz}}{\partial z}=0 \tag{1.64}$$

Using the strain definitions given in Equations 1.2 to 1.5, $e_{zz}=e_{yz}=e_{zx}=0$ and with no temperature change constitutive Equation 1.22 becomes

$$\sigma_{zz}=\nu\left(\sigma_{xx}+\sigma_{yy}\right) \tag{1.65}$$

One method of continuing the analysis is to define a stress function χ, known as Airy's stress function, which automatically satisfies equilibrium Equations 1.62 and 1.63

$$\sigma_{xx}=\frac{\partial^2\chi}{\partial y^2}, \qquad \sigma_{yy}=\frac{\partial^2\chi}{\partial x^2}, \qquad \sigma_{xy}=-\frac{\partial^2\chi}{\partial x\partial y} \tag{1.66}$$

Using Equation 1.65

$$\sigma_{zz}=\nu\nabla^2\chi \tag{1.67}$$

and constitutive Equations 1.20, 1.21 and 1.23 give the nonzero strain components as

$$e_{xx} = \frac{1}{E}\left[\frac{\partial^2\chi}{\partial y^2} - \nu\left(\frac{\partial^2\chi}{\partial x^2} + \nu\nabla^2\chi\right)\right] \tag{1.68}$$

$$e_{yy} = \frac{1}{E}\left[\frac{\partial^2\chi}{\partial x^2} - \nu\left(\frac{\partial^2\chi}{\partial y^2} + \nu\nabla^2\chi\right)\right] \tag{1.69}$$

$$e_{xy} = -\frac{2(1+\nu)}{E}\frac{\partial^2\chi}{\partial x\partial y} \tag{1.70}$$

These strains must be compatible. Equations 1.14 to 1.18 are automatically satisfied by the plane strain assumptions, and substituting the expressions for strains into Equation 1.13 gives

$$\frac{\partial^4\chi}{\partial x^4} + \frac{\partial^4\chi}{\partial y^4} - \nu\left[2\frac{\partial^4\chi}{\partial x^2\partial y^2} + \nu\nabla^2(\nabla^2\chi)\right] = -2(1+\nu)\frac{\partial^4\chi}{\partial x^2\partial y^2}$$

which can be simplified to

$$\nabla^4\chi = 0 \tag{1.71}$$

irrespective of the value of Poisson's ratio. The biharmonic partial differential operator ∇^4 is defined by

$$\nabla^4\chi = \nabla^2(\nabla^2\chi) = \frac{\partial^4\chi}{\partial x^4} + \frac{\partial^4\chi}{\partial y^4} + 2\frac{\partial^4\chi}{\partial x^2\partial y^2} \tag{1.72}$$

Equation 1.71 may be subject to various types of boundary conditions, including prescribed stresses or displacements.

1.2.6 Plane stress

Figure 1.6 shows a flat solid plate lying parallel to the $x - y$ plane. Provided the applied loadings are in plane of the plate as shown, the direct and shear stress components on the faces of the plate are zero. If the plate is thin then it may be assumed that throughout the material the plane stress approximation $\sigma_{zz} = \sigma_{yz} = \sigma_{xz} = 0$ is applicable. Ignoring body forces, the equilibrium conditions again reduce to Equations 1.62 and 1.63, allowing the use of Airy's stress function defined in Equations 1.66.

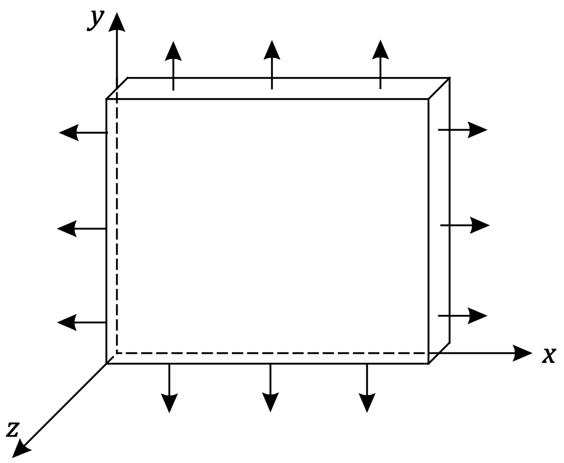


Figure 1.6 The plane stress approximation for a thin solid body

Using constitutive Equations 1.20 to 1.25, $e_{yz} = e_{zx} = 0$ and with no temperature change

$$e_{xx} = \frac{1}{E}\left(\sigma_{xx} - \nu\sigma_{yy}\right) = \frac{1}{E}\left(\frac{\partial^2\chi}{\partial y^2} - \nu\frac{\partial^2\chi}{\partial x^2}\right) \tag{1.73}$$

$$e_{yy} = \frac{1}{E}\left(\sigma_{yy} - \nu\sigma_{xx}\right) = \frac{1}{E}\left(\frac{\partial^2\chi}{\partial x^2} - \nu\frac{\partial^2\chi}{\partial y^2}\right) \tag{1.74}$$

$$e_{zz} = -\frac{\nu}{E}\left(\sigma_{xx} + \sigma_{yy}\right) = -\frac{\nu}{E}\nabla^2\chi \tag{1.75}$$

$$e_{xy} = \frac{2(1+\nu)}{E}\sigma_{xy} = -\frac{2(1+\nu)}{E}\frac{\partial^2\chi}{\partial x\partial y} \tag{1.76}$$

If these expressions for strains are substituted into compatibility Equation 1.13 then

$$\frac{\partial^4\chi}{\partial x^4} + \frac{\partial^4\chi}{\partial y^4} - 2\nu\frac{\partial^4\chi}{\partial x^2\partial y^2} = -2(1+\nu)\frac{\partial^4\chi}{\partial x^2\partial y^2}$$

$$\nabla^4\chi = 0 \tag{1.77}$$

irrespective of the value of Poisson's ratio. The remaining compatibility Equations 1.14 to 1.18 are not all automatically satisfied but, it can be shown that they are satisfied approximately when the plate is thin enough for variations through the thickness of the stresses and strains are small enough to be ignored. The possible boundary conditions for plane stress problems are similar to those for plane strain.

1.2.7 Recirculating viscous flow

In Section 1.2.1 the governing equation for downstream viscous flow in a uniform channel was derived. If, for example, the top surface of this channel, shown in Figure 1.2, moves with a velocity V_x in the x-direction relative to the remaining stationary sides, a recirculating viscous flow is created in the plane of the cross-section. A stream function as defined in Equations 1.47 automatically satisfies continuity for this two-dimensional flow. Ignoring body forces, the equilibrium equations are Equations 1.62 and 1.63. Introducing constitutive Equations 1.26 and 1.27, these equilibrium conditions may be expressed as

$$\frac{\partial p}{\partial x} = \mu\left[2\frac{\partial^2 u}{\partial x^2} + \frac{\partial}{\partial y}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\right] \tag{1.78}$$

$$\frac{\partial p}{\partial y} = \mu\left[2\frac{\partial^2 v}{\partial y^2} + \frac{\partial}{\partial x}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\right] \tag{1.79}$$

Using differentiation to remove the pressure from these equations, and introducing the stream function, the governing equation is obtained as

$$\nabla^4\psi = 0 \tag{1.80}$$

If there is no slip between the fluid and the solid walls of the channel, the boundary conditions are

$$\psi = 0\,,\quad v = -\frac{\partial\psi}{\partial x} = 0 \qquad \text{on } x = 0\,,\ x = W$$

$$\psi = 0\,,\quad u = \frac{\partial\psi}{\partial y} = 0 \qquad \text{on } y = 0 \tag{1.81}$$

$$\psi = Q_L\,,\quad u = \frac{\partial\psi}{\partial y} = V_x \qquad \text{on } y = H$$

where Q_L is the volumetric flow rate per unit length of channel in the downstream direction leaking into and out of the channel in the small gaps between the moving and fixed channel surfaces. The absolute values of the stream function defined on these surfaces are not important, provided a difference of Q_L is maintained between them, because it is only the derivatives of ψ that define velocities in the flow.

In this problem of recirculating viscous flow, the direct strain rate in the z-direction is zero, and there is a close physical analogy with the plane strain deformation of solid bodies described in Section 1.2.5.

1.2.8 Laterally loaded flat plate

The last example concerns a flat plate, of uniform thickness , lying parallel to the $x - y$ plane and subjected to a lateral pressure $p(x, y)$. Without going into the details, the equation governing the displacement normal to the surface of the plate is

$$\nabla^4 w = \frac{p}{D} \tag{1.82}$$

The parameter $D = Eh^3/12(1-\nu^2)$ is known the flexural rigidity of the plate. Applicable boundary conditions include $w = 0$ for a rigidly supported edge, and $\partial w/\partial n = 0$ (n being the direction of the outward normal to the boundary in the $x-y$ plane) for an edge of the plate which is rigidly clamped and therefore not able to rotate. It is interesting to note the familiar mathematical form of the governing equation, although such plate bending problems are outside the scope of this book.

1.2.9 General comments

In the above examples there are obvious similarities between the governing partial differential equations. There are also a number of general observations that can be made. Problems involving solid bodies tend to be formulated in terms of stress variables (including stress functions) which automatically satisfy the conditions of equilibrium, and solutions are obtained which satisfy the equations of strain compatibility. On the other hand, problems in fluid flow tend to be formulated in terms of velocity variables (including stream functions) which automatically satisfy the continuity equation, and solutions are obtained which satisfy the equations of equilibrium. These approaches are guided by the forms of the boundary conditions, with stresses being more commonly prescribed for solids, and velocities for fluids. Such formulations are not necessarily relevant when numerical methods of solution are employed, particularly when more general mixed types of boundary conditions have to be satisfied.

Simple boundary conditions define either the value of the dependent variable on a boundary, or the value of its first derivative in the direction normal to the boundary. A more general boundary condition, of which these are special cases, is typified by Equation 1.57 which takes the form

$$a_1 \frac{\partial \psi}{\partial n} + a_2 \psi + a_3 = 0 \tag{1.83}$$

where ψ is the dependent variable and and are known constants. The order of the differential equation governing a problem, which is the order of the highest derivative in the equation, determines the number of boundary conditions required for a solution to be found. The second-order equations arising in Sections 1.2.1 to 1.2.4 need two conditions per co-ordinate direction, one on each boundary. The fourth-order equations arising in Sections 1.2.5 to 1.2.8 need four conditions, two on each boundary (for example, Equations 1.81).

Solving the governing equations with the appropriate boundary conditions for a particular problem gives the dependent variable as a function of position within the region of interest, the solution domain. Some problems call for further analysis. Flow rate in Section 1.2.1 and torsional couple in Section 1.2.2, for example, are obtained by integration of the dependent variables over the solution domains. While in many of the problems described, body forces and temperature changes are ignored, if they were included the fundamental types of the differential equations obtained would not be affected. These equation types have a large impact on the methods of solution to be employed.

1.1.10 Harmonic and biharmonic equations

The governing equations for the problems outlined are essentially of two types.

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = \nabla^2 \psi = f_1(x, y) \tag{1.84}$$

$$\frac{\partial^4 \psi}{\partial x^4} + \frac{\partial^4 \psi}{\partial y^4} + 2\frac{\partial^4 \psi}{\partial x^2 \partial y^2} = \nabla^2(\nabla^2 \chi) = \nabla^4 \chi = f_2(x, y) \tag{1.85}$$

Second-order Equation 1.84 is often referred to as Poisson's equation, reducing to Laplace's equation as a particular case when $f_1 = 0$. Equation 1.85, on the other hand, is fourth-order. One way of distinguishing between the two types is by referring to them as harmonic (Equation 1.84) and biharmonic (Equation 1.85), both somewhat mathematical descriptions. For practical engineering purposes it is perhaps more convenient to refer to problems governed by Equation 1.84 as potential problems. In this book, the only types of problems governed by Equation 1.85 which are considered in detail are elastic stress analysis problems, treated in Part 2.

It is worth noting that all the problems considered are equilibrium problems, involving either steady fluid flows or static stresses in solids. If, for example, the time derivative term in Equation 1.10 is retained in Equation 1.55, the resulting unsteady thermal conduction problem is of the propagation type, with the solution changing with time.

1.3 Methods for Solving Harmonic and Biharmonic Equations

Harmonic and biharmonic partial differential equations can only be solved analytically in few simple cases. Although many mathematical functions satisfy the differential equations, in general they cannot also satisfy the imposed boundary conditions, and it is the boundary conditions as well as the differential equations that define the solutions.

Traditional numerical methods of solution are based on discretisation, with continuous functions such as stresses, displacements or velocities being represented approximately by discrete values at a finite number of points both within the solution domain and on its boundary. These values are computed from sets of simultaneous, normally linear, algebraic equations. The accuracies of the approximations increase with the numbers of points employed, especially if the extra points are concentrated in regions where the functions change their values rapidly. Various methods of analysis are used to assemble the required sets of linear algebraic equations, including finite difference, finite volume and finite element methods.

The purpose of this book is to introduce a fundamentally different class of methods known as boundary element methods for solving potential and elastic stress analysis problems. As their name suggests, they require only the boundary of the solution domain to be discretised, although values of the variables can also be found at any point of interest within the domain.

Problems

1.1 The downstream viscous flow problem described in Section 1.2.1 has a plane of symmetry. If advantage is to be taken of this symmetry to analyse only one half of the channel, what are the boundary conditions that should be applied to the reduced domain?

1.2 A thin membrane is subject to a uniform tensile force of S per unit length in all in-plane directions. Under a uniform pressure of magnitude p the deflection w in the z-direction of the membrane which lies in the $x-y$ plane is governed by the equation

$$\frac{\partial^2 w}{\partial x^2}+\frac{\partial^2 w}{\partial y^2}=-\frac{p}{S}$$

Explain how such a membrane can be used experimentally as an analogue for the torsion problem.

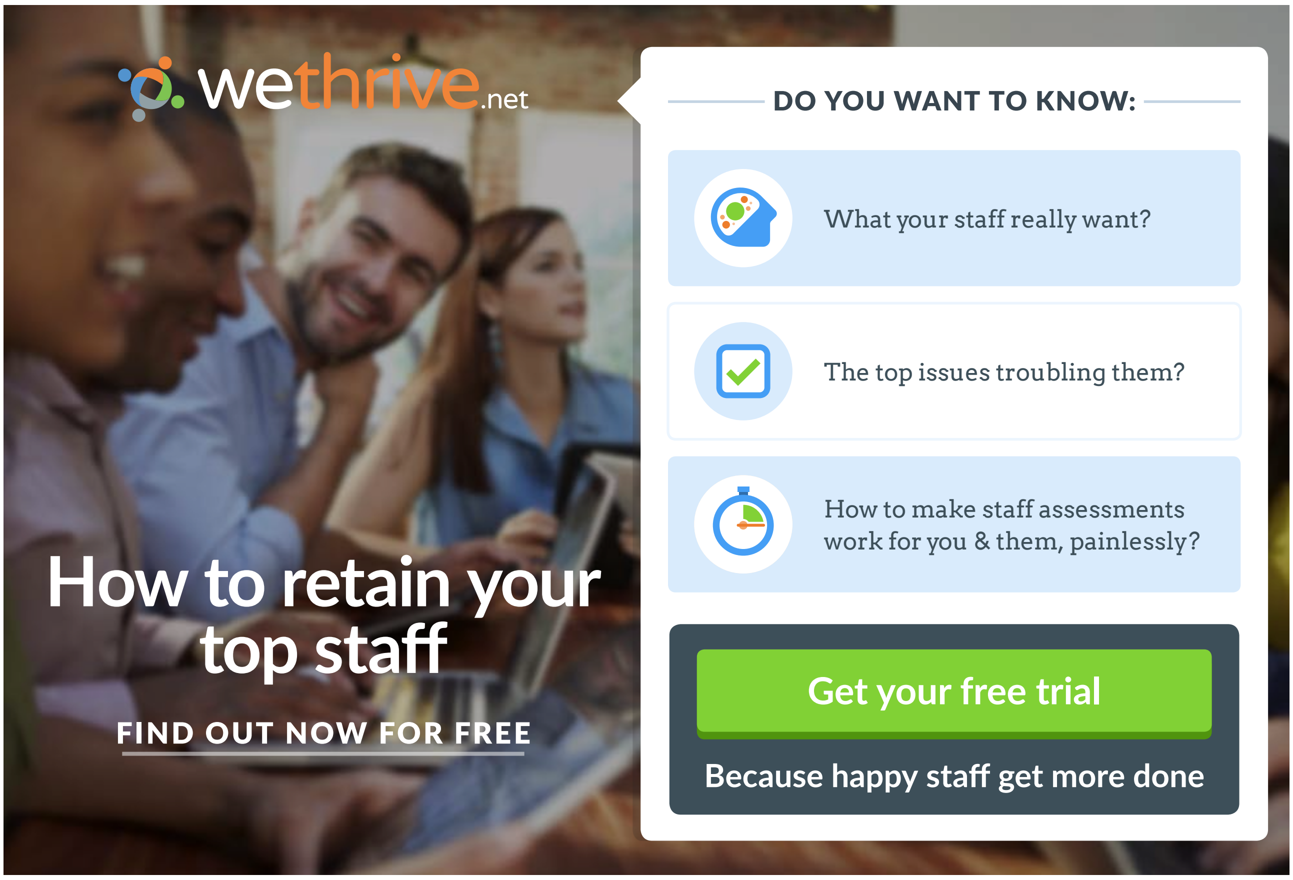
1.3 Water seeps through the porous rock underneath a concrete dam. Equation 1.61 for pressure governs this flow. What are the appropriate boundary conditions?

1.4 In the following problems a two-dimensional approximation is to be used for analysis. In each case, which is the more appropriate approximation, plane strain or plane stress?

a) An elastic membrane under in-plane loading.
b) The cross-section of a dam holding back water.
c) The cross-section of a billet of metal undergoing forging.
d) The cross-section of a thick-walled cylinder.
e) A tension test specimen.

1.5 What physical problems are represented by the following distributions of Airy's stress function: (a) $\chi = Ax^2$, (b) $\chi = By^3$, where A and B are constants?

1.6 Starting from Equations 1.20 to 1.22 under plane strain conditions in the absence of temperature changes, determine the direct stresses and in terms of the corresponding direct strains σ_{xx} and σ_{yy}. What happens when the value of Poisson's ratio is ½? Explain the physical significance.



2 Boundary Element Analysis of Potential Problems

In this chapter a form of boundary element analysis for two-dimensional potential problems such as those outlined in Chapter 1 is presented. Two types of elements are considered: constant and quadratic. Three-dimensional problems are discussed briefly in Section 7.3.

From Equation 1.84, potential problems are governed by Poisson's equation

$$\frac{\partial^2\psi}{\partial x^2}+\frac{\partial^2\psi}{\partial y^2}=\nabla^2\psi=f_1(x,y) \tag{2.1}$$

where $f_1(x,y)$ is a known function of position within the solution domain. If $f_1(x,y)=0$ everywhere then Equation 2.1 becomes Laplace's equation, which it is convenient to consider first

$$\frac{\partial^2\psi}{\partial x^2}+\frac{\partial^2\psi}{\partial y^2}=\nabla^2\psi=0 \tag{2.2}$$

Possible boundary conditions at the edges of the domain are prescribed values of potential ψ, prescribed values of the potential gradient in the direction normal to the boundary, or in general a linear relationship between potential and potential gradient

$$a_1\frac{\partial\psi}{\partial n}+a_2\psi+a_3=0 \tag{2.3}$$

where a_1, a_2 and a_3 are known constants.

2.1 Fundamental Solution

A substantial amount of fairly sophisticated mathematics has gone into the development of boundary element methods. In this book, however, only a minimum of mathematics is introduced, and the emphasis is on developing methods which are straightforward to understand and apply to practical engineering problems.

In order to proceed, a special form of solution to Laplace's equation is required, which is known as the fundamental solution. In practical terms, the fundamental solution is that due to a source of potential concentrated at a point in a solution domain of infinite extent in all directions. A mathematical requirement of the fundamental solution is that its value is singular (goes to infinity) at a point – the point where the source is located. The effects of a point source in terms of potential distribution will be the same in all directions moving away from the point. In two dimensions, if r and θ are plane polar co-ordinates centred at the source point, the potential distribution will be a function only of r. In fact it is

$$\phi=\ln\left(\frac{1}{r}\right) \tag{2.4}$$

where "ln" is the natural logarithm function. If co-ordinates x and y also have their origin at the source point then

$$r = +(x^2 + y^2)^{½} \tag{2.5}$$

and $\phi = \ln\left(\frac{1}{r}\right) = \ln(x^2 + y^2)^{-½} = -½\ln(x^2 + y^2)$ (2.6)

Differentiating partially with respect to x

$$\frac{\partial\phi}{\partial x} = -½\frac{2x}{(x^2+y^2)} = -\frac{x}{(x^2+y^2)} \tag{2.7}$$

and again $\frac{\partial^2\phi}{\partial x^2} = -\frac{1}{(x^2+y^2)} + \frac{x}{(x^2+y^2)^2}2x = -\frac{1}{(x^2+y^2)} + \frac{2x^2}{(x^2+y^2)^2}$ (2.8)

Similarly, differentiating partially with respect to y

$$\frac{\partial\phi}{\partial y} = -\frac{y}{(x^2+y^2)} \tag{2.9}$$

$$\frac{\partial^2\phi}{\partial y^2} = -\frac{1}{(x^2+y^2)} + \frac{2y^2}{(x^2+y^2)^2} \tag{2.10}$$

Adding the second derivatives defined in Equations 2.8 and 2.10

$$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = \nabla^2\phi = -\frac{2}{(x^2+y^2)} + \frac{2(x^2+y^2)}{(x^2+y^2)^2} = 0 \tag{2.11}$$

and the fundamental solution satisfies Laplace's equation, Equation 2.2.

The physical significance of Equation 2.4 can be explored by regarding ϕ as the temperature in a medium of infinite extent in all directions, and having a thermal conductivity of k. Consider a point Z with co-ordinates (x, y) at distance r from a point heat source at O in Figure 2.1.

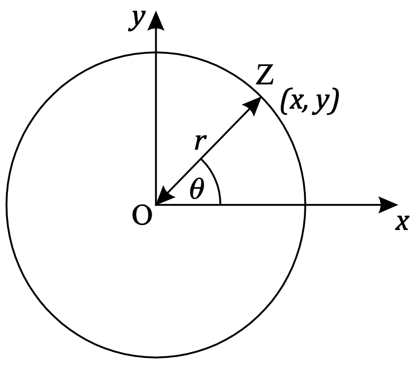


Figure 2.1 Co-ordinates and geometry near a point heat source

The temperature gradient in the radial direction at Z is

$$\frac{\partial\phi}{\partial r} = \frac{\partial}{\partial r}(-\ln r) = -\frac{1}{r}$$

Now consider the circle passing through point Z of radius centred at point O. The total rate of heat conduction outwards across this circle is

$$H = \oint k\left(-\frac{\partial\phi}{\partial r}\right) r\,\mathrm{d}\theta = \oint k\,\mathrm{d}\theta = 2\pi k \tag{2.12}$$

where the line integration is carried out around the circumference. The negative sign is present because a positive outflow of heat is associated with a negative temperature gradient in the outward direction. The magnitude of quantity H is the strength of the heat source, which is a constant. Under steady conditions the total heat conducted outwards from a point heat source is independent of the distance from the source.

If p and q are any two points within the two-dimensional solution domain for a potential problem, the fundamental solution at q referred to an origin at p (or vice versa), may be expressed as

$$\phi = \ln\left[\frac{1}{r(p,q)}\right] \qquad r(p,q) \neq 0 \tag{2.13}$$

As $r(p,q)$ approaches zero the solution tends to infinity, which is not permitted. Point p is referred to as the source point, q as the field point.

2.2 Boundary Integral Equation

The aim now is to relate what is happening within a solution domain of a potential problem to what is happening on the boundary, with a view to only analysing the latter. The above examination of the fundamental solution provides some useful guidance. Equation 2.12 was the result of summing the local heat transfer rates across the circle surrounding the heat source to give the total net heat flow, which under steady conditions must be equal to the magnitude of the source. It involved in effect an integral around the circle or boundary. This idea can be generalised: a boundary, S, of any size and shape could be drawn around the source. The total rate of heat conduction out of the domain can be found from a boundary integral along the path of the boundary

$$H = \oint_S k\left(-\frac{\partial\phi}{\partial n}\right)\mathrm{d}S \tag{2.14}$$

where n is the outward normal direction to boundary S.

The line integral symbol in Equation 2.14 means integration in the anticlockwise direction along the boundary enclosing the solution domain. In general, it means integration along the boundary in the direction which always keeps the domain on the left. If the domain contains a hole, for example, which is perfectly permissible in the present context, the solution domain would be defined by two boundaries: the outer one and the edge of the inner hole. Integration around the hole boundary would be in the clockwise direction to keep the domain always on the left. The sum of the integrals around the two boundaries would be required to find the total rate of heat conduction from the domain.

Equation 1.55, a simplified form of the general energy equation, Equation 1.10, relates heat generation per unit area in the two-dimensional domain to the Laplace operator applied to temperature

$$g = -k\nabla^2\phi \tag{2.15}$$

For a point source of heat, the magnitude of g can be thought of as tending to infinity at the point, but with the area over which it acts tending to zero. The summation of heat generated can, however, be expressed as

$$H = \iint_A g\,\mathrm{d}x\,\mathrm{d}y = \iint_A (-k\nabla^2\phi)\,\mathrm{d}x\,\mathrm{d}y \tag{2.16}$$

where A is the area of the domain enclosed by boundary S. Combining Equations 2.14 and 2.16, and removing the constant common factor of $-k$

$$\iint_A \nabla^2\phi \, \mathrm{d}x \, \mathrm{d}y = \oint_S \frac{\partial\phi}{\partial n} \mathrm{d}S \tag{2.17}$$

In this more general form, the sum of potential sources within a solution domain is equated to the net flow of potential across the boundary or boundaries.

Comparing Equations 2.12 and 2.16 it is clear that for the fundamental solution associated with a single point source within the domain

$$\iint_A (\nabla^2\phi) \, \mathrm{d}x \, \mathrm{d}y = -2\pi \tag{2.18}$$

In mathematical terms, the magnitude of a source at a particular point is given by the divergence of the vector field. The vector field is the potential gradient

$$\nabla\phi = \left(\boldsymbol{i}\frac{\partial}{\partial x} + \boldsymbol{j}\frac{\partial}{\partial y}\right)\phi = \boldsymbol{i}\frac{\partial\phi}{\partial x} + \boldsymbol{j}\frac{\partial\phi}{\partial y} \tag{2.19}$$

where $\boldsymbol{i}$ and $\boldsymbol{j}$ are the unit vectors in the x and y directions, respectively, and its divergence is

$$\nabla.(\nabla\phi) = \left(\boldsymbol{i}\frac{\partial}{\partial x} + \boldsymbol{j}\frac{\partial}{\partial y}\right).\left(\boldsymbol{i}\frac{\partial\phi}{\partial x} + \boldsymbol{j}\frac{\partial\phi}{\partial y}\right) = \nabla^2\phi \tag{2.20}$$

Equation 2.17 can be written as

$$\iint_A \nabla.(\nabla\phi) \, \mathrm{d}x \, \mathrm{d}y = \oint_S \frac{\partial\phi}{\partial n} \mathrm{d}S \tag{2.21}$$

and is sometimes referred to as the divergence theorem. It is a general result which holds for any continuous and differentiable function, and could therefore be written for not just potential gradient $\nabla.\phi$ but for $\psi\nabla.\phi$ where ψ is any (continuous and differentiable) function.

$$\iint_A \nabla.(\psi\nabla\phi) \, \mathrm{d}x \, \mathrm{d}y = \oint_S \psi\frac{\partial\phi}{\partial n} \mathrm{d}S \tag{2.22}$$

Now $\nabla.(\psi\nabla\phi) = \left(\boldsymbol{i}\frac{\partial}{\partial x} + \boldsymbol{j}\frac{\partial}{\partial y}\right).\left(\boldsymbol{i}\,\psi\frac{\partial\phi}{\partial x} + \boldsymbol{j}\,\psi\frac{\partial\phi}{\partial y}\right)$

$$= \frac{\partial}{\partial x}\left(\psi\frac{\partial\phi}{\partial x}\right) + \frac{\partial}{\partial y}\left(\psi\frac{\partial\phi}{\partial y}\right) = \psi\nabla^2\phi + \frac{\partial\psi}{\partial x}\frac{\partial\phi}{\partial x} + \frac{\partial\psi}{\partial y}\frac{\partial\phi}{\partial y}$$

and Equation 2.22 becomes

$$\iint_A \left[\psi\nabla^2\phi + \frac{\partial\psi}{\partial x}\frac{\partial\phi}{\partial x} + \frac{\partial\psi}{\partial y}\frac{\partial\phi}{\partial y}\right] \mathrm{d}x \, \mathrm{d}y = \oint_S \psi\frac{\partial\phi}{\partial n} \mathrm{d}S \tag{2.23}$$

Interchanging functions ψ and ϕ

$$\iint_A \left[\phi\nabla^2\psi + \frac{\partial\phi}{\partial x}\frac{\partial\psi}{\partial x} + \frac{\partial\phi}{\partial y}\frac{\partial\psi}{\partial y}\right] \mathrm{d}x\, \mathrm{d}y = \oint_S \phi\frac{\partial\psi}{\partial n}\mathrm{d}S \tag{2.24}$$

and subtracting Equation 2.24 from 2.23

$$\iint_A (\psi\nabla^2\phi - \phi\nabla^2\psi)\, \mathrm{d}x\, \mathrm{d}y = \oint_S \left(\psi\frac{\partial\phi}{\partial n} - \phi\frac{\partial\psi}{\partial n}\right)\mathrm{d}S \tag{2.25}$$

which is a result known as Green's symmetric identity (symmetric because it is symmetric in functions $\psi$and ϕ).

Equation 2.25 looks promising in that the left hand side integral over the area of the solution domain involves $\nabla^2\phi$ and $\nabla^2\psi$, which will both be zero if ϕ and ψ satisfy Laplace's equation. Hence the area integral will also be zero, leaving only the boundary integral on the right hand side. Suppose that ψ is the required solution to Equation 2.2, and that ϕ is the fundamental solution. Now ϕ does satisfy $\nabla^2\phi = 0$, except at the source point for the fundamental solution if this is located within the solution domain. With a little care this can be dealt with.

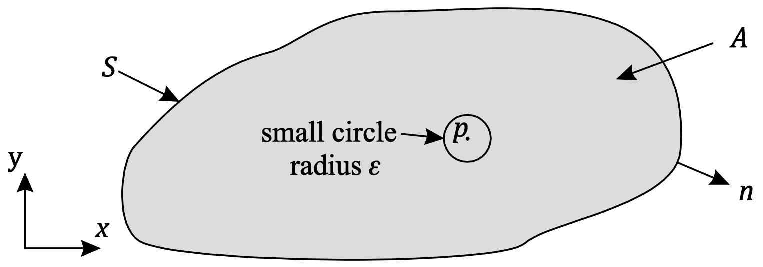


Figure 2.2 A two-dimensional solution domain, including a small circular region of radius ε surrounding the source point p

Surround the source point p with a small circle as shown in Figure 2.2. If the circle is very small, the value of ψ can be taken as constant within it, equal to the value at point p, that is $\psi(p)$. The integral over the circle of $\nabla^2\phi$ is already known from Equation 2.18 (which evolved from Equation 2.12) as -2π. Equation 2.25 becomes

$$2\pi\psi(p) + \oint_S \left(\psi\frac{\partial\phi}{\partial n} - \phi\frac{\partial\psi}{\partial n}\right) \mathrm{d}S = 0 \tag{2.26}$$

The function ϕ is known, so provided the values of both ψ and its gradient normal to the boundary are known at every point along the boundary, this provides a means of calculating the value of the required solution ψ at any point within the solution domain. The challenge now is to find these boundary values.

It can be anticipated that if point p is taken to the boundary at a point P, the resulting equation will involve only quantities on the boundary.

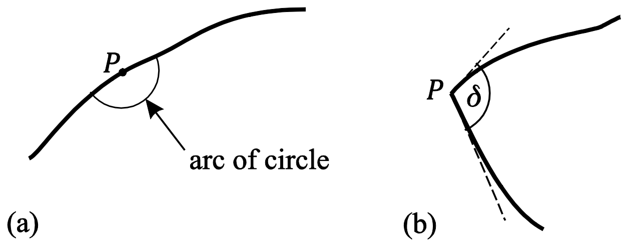


Figure 2.3 Typical point *P* on the boundary
(a) *P* on a smooth part of the boundary (b) *P* at a corner

Around the source point P on the boundary, which is assumed to be locally smooth as in Figure 2.3a, consider an arc of a circle centred at P, the ends of which are on the boundary. "Smooth" in this context means that the direction of the tangent to the boundary does not undergo an abrupt change at point P. If the radius of the arc is very small, the value of ψ can again be taken as constant within it, equal to the value at point P, that is $\psi(P)$. The integral over the region enclosed by the arc of $\nabla^2\phi$ can be inferred from Equation 2.18 (which again evolved from Equation 2.12) as $-\pi$, because as the radius of the arc shrinks to zero the region of integration becomes a semi-circle, containing just half of the effects of the point source (the other half being outside the boundary). Equation 2.25 becomes

$$\pi\psi(P) + \oint_S \left(\psi\frac{\partial\phi}{\partial n} - \phi\frac{\partial\psi}{\partial n}\right) \mathrm{d}S = 0 \tag{2.27}$$

which is now a truly boundary integral equation. The first term in this equation, which does not involve integration, is referred to as the "free term". If boundary point *P* is not on a smooth part of the boundary but at a corner such as that shown in Figure 2.3b, then the free term becomes $\delta\psi(P)$, where δ is the angle between the tangents at the corner measured through the domain. In practice, the precise magnitude of the constant in the free term is not important because it can be found in another way.

Equation 2.27 is the boundary integral equation for point *P* as the source point of the fundamental solution. To improve the clarity of what it means it can be written as

$$\pi\psi(P) + \oint_S \psi(Q)\frac{\mathrm{d}\phi(P,Q)}{\mathrm{d}n(Q)}\mathrm{d}S(Q) = \oint_S \phi(P,Q)\left(\frac{\mathrm{d}\psi}{\mathrm{d}n}\right)_Q \mathrm{d}S(Q) \tag{2.28}$$

Point Q is on the boundary and moves around the boundary as the integration with respect to *S*, the distance along the boundary, proceeds. Function $\phi(P,Q)$ is the fundamental solution for field point Q due to a source point at *P*, and n is the direction of the normal to the boundary at the field point. At any point Q either the value of ψ or its normal gradient will be known from the boundary conditions of the particular problem (or there will be a known linear relationship between them). Care will have to be taken when Q passes through *P* during the integration, because there the fundamental solution is not defined.

2.3 Discretisation of the Boundary Integral Equation

In general, Equation 2.28 cannot be solved analytically, and some form of numerical method must be employed. Solution variable ψ is found not as a continuous algebraic function of position along the boundary, but as numerical values at a finite number of discrete points on the boundary. The boundary may be subdivided into small pieces or boundary elements. Associated with each element are one or more of these points, known as nodes or nodal points. The distributions of ψ and its normal gradient over the elements are defined in terms of nodal point values by suitable interpolation functions. For example, the simplest type of boundary element is the constant element, having a single node at its centre and over which the variables are assumed to be constant. Next comes the linear element with a node at each end and assumed linear variations between them. Then the quadratic element with an additional node at its centre and assumed quadratic variations between the three nodes.

Whichever type of boundary element is used, boundary integral Equation 2.28 is applied to each of the N nodal points *P* in turn, in the discretised form

$$\pi\psi(P) + \sum_{m=1}^{M}\int_{S_m}\psi(Q)\frac{\mathrm{d}}{\mathrm{d}n(Q)}\left\{\ln\left[\frac{1}{r(P,Q)}\right]\right\}\mathrm{d}S(Q) = \sum_{m=1}^{M}\int_{S_m}\ln\left[\frac{1}{r(P,Q)}\right]\left(\frac{\mathrm{d}\psi}{\mathrm{d}n}\right)_Q\mathrm{d}S(Q) \tag{2.29}$$

The total number of boundary elements is M, m is an element counter, and S_m is the piece of the boundary occupied by element number m. The details of how integration is carried out over an individual element depend on the type of element involved. Equation 2.29 represents a set of N linear equations, where N is the number of nodal points ($N = M$ for constant or linear elements, $N = 2M$ for quadratic elements). They can be expressed as

$$[A][\psi] = [B]\left[\frac{\mathrm{d}\psi}{\mathrm{d}n}\right] \tag{2.30}$$

where $[A]$ and $[B]$ are square matrices containing known constant coefficients, and in general all will be non-zero. The column vectors $[\psi]$ and $\left[\frac{\mathrm{d}\psi}{\mathrm{d}n}\right]$ contain the nodal point values of potential and potential gradient normal to the boundary. At each node one of these will be unknown and one of them known, or there will be a linear relationship between them. The equations can be rearranged, taking all unknown quantities to the left hand side, known quantities to the right hand side, giving a set of linear equations in a familiar form

$$[A^*][x] = [B^*][y] = [b] \tag{2.31}$$

where $[A^*]$ and $[B^*]$ are modified coefficient matrices and $[b]$ is a column vector of known coefficients. This set can be solved for the N unknowns x at the nodes, meaning that the potential and potential gradient are known at every nodal point on the boundary. Given this information, values of potential at points within the solution domain can be found from Equation 2.26.

The case of a "mixed" boundary condition where there is a linear relationship between potential and potential gradient deserves a little more explanation. The general form is given by Equation 2.3, which for present purposes can be re-expressed as

$$\frac{\mathrm{d}\psi}{\mathrm{d}n} = \alpha\psi + \beta \tag{2.32}$$

where α and β are known constants. In Equations 2.30 the potential gradient for the relevant node is replaced by the expression $\alpha\psi + \beta$, where the potential is unknown. While β remains on the right hand side and contributes to known vector $[b]$, $\alpha\psi$ is taken to the left hand side by subtracting α times the relevant coefficient of $[B]$ from the corresponding coefficient of matrix $[A]$. The details of this process should become clearer when the programming of the analysis is considered.

2.4 Constant Boundary Elements

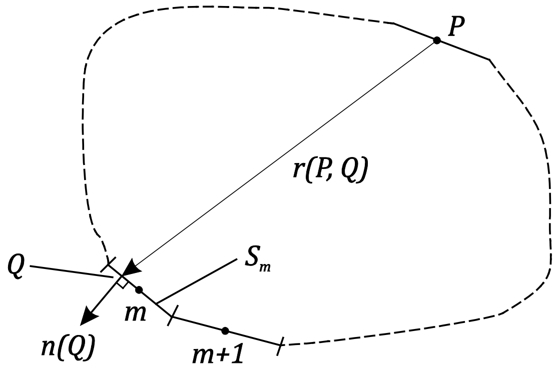


Figure 2.4 Constant element discretisation of a two-dimensional solution domain boundary

A constant boundary element for a two-dimensional problem takes the form of a straight line representing a piece of the boundary of the solution domain. The potential and potential gradient are assumed not to vary along an element, their constant values being assigned to the central nodal point of the element. Figure 2.4 shows a typical arrangement of constant elements on a boundary. Since the number of nodes is equal to the number of elements it is convenient to number both nodes and elements in the same sequence, taken from an arbitrary starting point and keeping the domain to the left of the direction of numbering. Element number m forming piece S_m of the boundary therefore contains node number j, where $j = m$. Equation 2.29 becomes

$$\pi\psi(P) + \sum_{m=1}^{M} \psi_m \int_{S_m} \frac{\mathrm{d}}{\mathrm{d}n(Q)}\left\{\ln\left[\frac{1}{r(P,Q)}\right]\right\}\mathrm{d}S(Q) = \sum_{m=1}^{M}\left(\frac{\mathrm{d}\psi}{\mathrm{d}n}\right)_m \int_{S_m} \ln\left[\frac{1}{r(P,Q)}\right]\mathrm{d}S(Q) \qquad (2.33)$$

The functions which have to be integrated in this equation are often referred to as kernels. The forms of these functions and the methods of evaluating the integrals depend on whether or not the node P and the point Q are in the same element.

2.4.1 Points *P* and *Q* not in the same element

Taking the general case shown in Figure 2.4, in which P and Q are not in the same element, the first kernel is

$$\frac{\mathrm{d}}{\mathrm{d}n(Q)}\left\{\ln\left[\frac{1}{r(P,Q)}\right]\right\} = \frac{\mathrm{d}}{\mathrm{d}n}(-\ln r) = -\frac{1}{r}\frac{\mathrm{d}r}{\mathrm{d}n} \qquad (2.34)$$

Radius r is simply the scalar distance between P and Q, and $\frac{\mathrm{d}r}{\mathrm{d}n}$ the rate of change of distance r with movement in the direction of the normal n. This function has to be integrated over each and every boundary element. While the potential and potential gradient are assumed to be constant over an individual element, the fundamental solution and hence kernel function associated with a particular point P varies over the element.

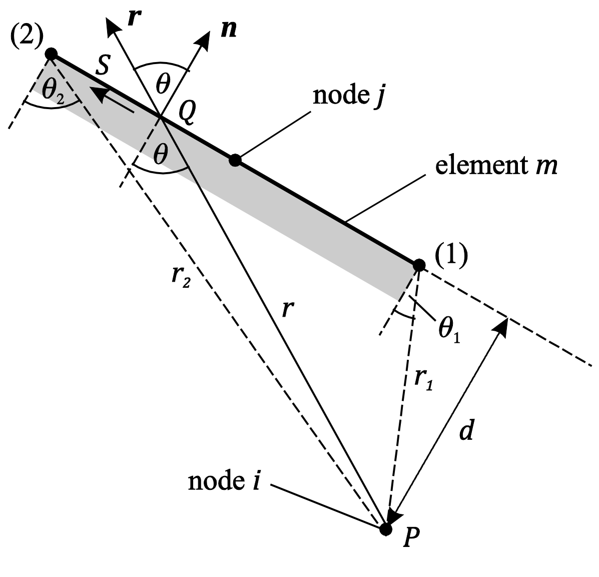


Figure 2.5 Integration over a typical constant element

Figure 2.5 illustrates the process for a typical element m with node j as its node. Point P is also a node, say the node numbered $i(i \neq j)$ The coefficient A_{ij} of matrix $[A]$ in Equation 2.30 is therefore

$$A_{ij} = \int_{S_m} \frac{\mathrm{d}}{\mathrm{d}n(Q)}\left\{\ln\left[\frac{1}{r(P,Q)}\right]\right\}\mathrm{d}S(Q) = \int_{S_m}\left(-\frac{1}{r}\frac{\mathrm{d}r}{\mathrm{d}n}\right)\mathrm{d}S \qquad (2.35)$$

This is the coefficient in the i th row of the matrix, forming the equation for node i as the source point P, and in the jth column, corresponding to node j in element m. Angle θ is that between the radius vector $\boldsymbol{r}$ (from p in the direction of Q) and the outward normal vector $\boldsymbol{n}$, and $\frac{\mathrm{d}r}{\mathrm{d}n} = \cos\theta$. For a small incremental distance $\mathrm{d}S$ along the element from Q, the components along and perpendicular to vector $\boldsymbol{r}$ are $\mathrm{d}S\ \sin\theta$ and $\mathrm{d}S\ \cos\theta$, respectively, as shown in Figure 2.6. The latter is the same as $r\ \mathrm{d}\theta$, which means that

$$A_{ij} = \int_{S_m} \left(-\frac{1}{r}\frac{\mathrm{d}r}{\mathrm{d}n}\right)\mathrm{d}S = \int_{\theta_1}^{\theta_2}\left(-\frac{\cos\theta}{r}\right)\frac{r\ \mathrm{d}\theta}{\cos\theta} = \int_{\theta_1}^{\theta_2}(-\mathrm{d}\theta) = \theta_1 - \theta_2 \tag{2.36}$$

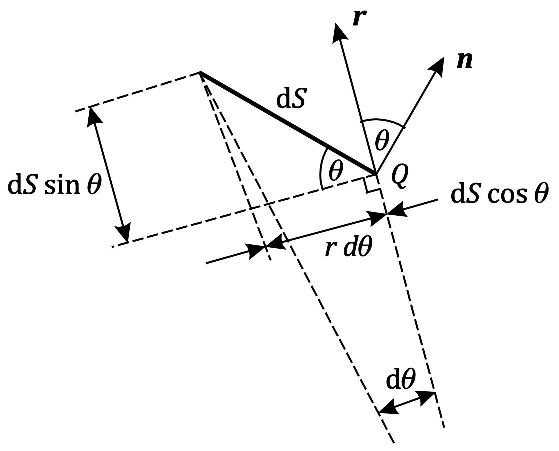


Figure 2.6 Incremental distances at point Q

The subscripts 1 and 2 refer to the ends of the element, taken in order along the boundary, always keeping the domain to the left. So the matrix coefficient is simply the difference between the angles at the first and second element ends (Figure 2.5), measured in radians. The required angles have to be computed with care, because inverse trigonometric functions have to be used, and these are multi-valued. A more robust approach is to find the angular difference $(\theta_1 - \theta_2)$ directly. Let $\hat{\boldsymbol{r}}_1$ be the unit vector in the direction from point P to the first end of the boundary element, with components $\hat{r}_{1x}$ and $\hat{r}_{1y}$ in the x and directions, respectively. Similarly, let $\hat{\boldsymbol{r}}_2$ be the unit vector in the direction from P to the second end, with components $\hat{r}_{2x}$ and $\hat{r}_{2y}$. Then the sine of the angle between them can be found from the vector product

$$\hat{\boldsymbol{r}}_1 \wedge \hat{\boldsymbol{r}}_2 = \begin{bmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ \hat{r}_{1x} & \hat{r}_{1y} & 0 \\ \hat{r}_{2x} & \hat{r}_{2y} & 0 \end{bmatrix} = \boldsymbol{k}\ \times 1\ \times 1\ \times \sin(\theta_2 - \theta_1) \tag{2.37}$$

$$\sin(\theta_1 - \theta_2) = \hat{r}_{1y}\hat{r}_{2x} - \hat{r}_{1x}\hat{r}_{2y} \tag{2.38}$$

where $\boldsymbol{k}$ is the unit vector in the direction, normal to the $x - y$ plane. The angular difference can be found from

$$\theta_1 - \theta_2 = \sin^{-1}(\hat{r}_{1y}\hat{r}_{2x} - \hat{r}_{1x}\hat{r}_{2y}) \tag{2.39}$$

It is worth noting that this result still holds when P is on the same side of the boundary as the outward pointing normal, which can happen with more complicated boundary shapes, and particularly when there are holes in the domain and therefore more than one boundary. In mathematical terms, such a domain is said to be multiply-connected.

The integral of the second kernel function gives the corresponding coefficient in the $[B]$ matrix in Equation 2.30

$$B_{ij} = \int_{S_m} \ln\left[\frac{1}{r(P,Q)}\right] \mathrm{d}S(Q) = \int_{\theta_1}^{\theta_2} \ln\left[\frac{1}{r(P,Q)}\right] \frac{r\,\mathrm{d}\theta}{\cos\theta} \tag{2.40}$$

If d is the distance from point P to the foot of the perpendicular on the line of the element (Figure 2.5), then $d = r\cos\theta$ and

$$B_{ij} = \int_{\theta_1}^{\theta_2} \ln\left(\frac{\cos\theta}{d}\right) \frac{d\,\mathrm{d}\theta}{\cos^2\theta} = \int_{\theta_1}^{\theta_2} \ln\left(\frac{\cos\theta}{d}\right) \sec^2\theta\, d\,\mathrm{d}\theta = d\int_{\theta_1}^{\theta_2} \ln\left(\frac{\cos\theta}{d}\right) \mathrm{d}(\tan\theta)$$

Integrating by parts

$$B_{ij} = d\left[\ln\left(\frac{\cos\theta}{d}\right)\tan\theta\right]_{\theta_1}^{\theta_2} - d\int_{\theta_1}^{\theta_2}\tan\theta\left(\frac{d}{\cos\theta}\right)\left(\frac{-\sin\theta}{d}\right)\mathrm{d}\theta$$

$$= d\left[\ln\left(\frac{\cos\theta}{d}\right)\tan\theta\right]_{\theta_1}^{\theta_2} + d\int_{\theta_1}^{\theta_2}(\sec^2\theta - 1)\mathrm{d}\theta$$

$$= d\left[\ln\left(\frac{\cos\theta}{d}\right)\tan\theta + \tan\theta - \theta\right]_{\theta_1}^{\theta_2}$$

which, given that $d = r_1\cos\theta_1 = r_2\cos\theta_2$, becomes

$$B_{ij} = r_2\sin\theta_2\left(\ln\left(\frac{1}{r_2}\right)+1\right) - r_1\sin\theta_1\left(\ln\left(\frac{1}{r_1}\right)+1\right) + d(\theta_1 - \theta_2) \tag{2.41}$$

Dimension d can be found as the component of either $\boldsymbol{r}_1$ or $\boldsymbol{r}_2$ in the normal direction $\boldsymbol{n}$, in other words from the scalar product of, say, vector $\boldsymbol{r}_1$ with unit vector $\hat{\boldsymbol{n}}$

$$d = \boldsymbol{r}_1.\hat{\boldsymbol{n}} = r_{1x}\hat{n}_x + r_{1y}\hat{n}_y \tag{2.42}$$

where $\hat{n}_x$ and $\hat{n}_y$ are the components of $\hat{\boldsymbol{n}}$ in the x and y directions. The two sines in Equation 2.41 can be found from the relevant vector products

$$\boldsymbol{k}\sin\theta_1 = \hat{\boldsymbol{n}} \wedge \hat{\boldsymbol{r}}_1$$

$$\sin\theta_1 = \hat{n}_x\hat{r}_{1y} - \hat{n}_y\hat{r}_{1x} \tag{2.43}$$

Similarly $\sin\theta_2 = \hat{n}_x\hat{r}_{2y} - \hat{n}_y\hat{r}_{2x}$ (2.44)

2.4.2 Points *P* and *Q* in the same element

Integration along the element containing the source point *P* involves integration through the singularity of the fundamental solution. Radius *r* and normal n are orthogonal, $\frac{\mathrm{d}r}{\mathrm{d}n} = 0$ in Equation 2.35, and there is no contribution to the term on diagonal of matrix $[A]$. However, due to the free term in Equation 2.33

$$A_{ii} = \pi \tag{2.45}$$

Using constant elements with nodes at their centres, the boundary is locally smooth at every node. It is worth noting, however, that the coefficients in any row $[A]$ of can be checked. If the potential is uniform over the entire boundary of the domain, it is also uniform over the domain itself. This means that the normal gradients of potential must be zero everywhere along the boundary. Hence

$$[A][\psi] = [0] \tag{2.46}$$

and, since the ψ values are all identical, the sum of the coefficients in any row of $[A]$ should be zero. Therefore

$$A_{ii} = -\sum_{\substack{j=1 \\ j\neq i}}^{M} A_{ij} \tag{2.47}$$

In the case of constant elements this merely provides a useful check on the coefficients. When higher-order elements such as quadratic are used, however, it provides an important means of computing the diagonal coefficients.

Figure 2.7 shows the situation when P and Q are in the same element, and the integral of the second kernel over this element is required.

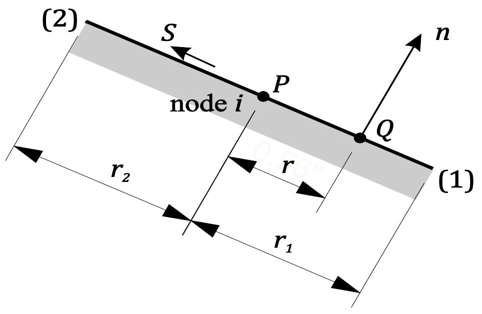


Figure 2.7 Special case when *P* and *Q* are in the same element

Point *P* is at the centre of the element, so that $r_1 = r_2$ and the integrals over the two halves of the element are identical. But they are also improper integrals, because of the singularity of the fundamental solution at *P*, and must therefore be treated as limits. Considering the integral over the half of the element between the first end, labelled 1, and the centre of the element

$$B_{ii} = \int_{S_m} \ln\left[\frac{1}{r(P,Q)}\right] \mathrm{d}S(Q) = 2\lim_{\varepsilon\to 0}\left\{\int_{r_1}^{\varepsilon} \ln\left(\frac{1}{r}\right)(-\mathrm{d}r)\right\}$$

$$= 2\lim_{\varepsilon\to 0}\left\{\int_{\varepsilon}^{r_1} \ln\left(\frac{1}{r}\right)\mathrm{d}r\right\}$$

$$= 2\lim\nolimits_{\varepsilon\to 0}\left\{\left[r\ln\left(\frac{1}{r}\right)\right]_{\varepsilon}^{r_1} + \int_{\varepsilon}^{r_1} r\frac{1}{r}\mathrm{d}r\right\}$$

$$= 2\lim_{\varepsilon\to 0}\left\{\left[r\ln\left(\frac{1}{r}\right)+r\right]_{\varepsilon}^{r_1}\right\}$$

and because $\lim_{\varepsilon \to 0} \left[\varepsilon \ln \left(\frac{1}{\varepsilon} \right) \right] = 0$ this becomes

$$B_{ii} = 2r_1 \left(\ln \left(\frac{1}{r_1} \right) + 1 \right) \tag{2.48}$$

While the kernel function is singular at P, its integral over the element is finite.

The constant boundary element treatment of potential problems has the merit that the integrations of the kernel functions can be carried out analytically, thereby eliminating one source of inaccuracy. On the other hand, the use of straight line elements with one node at the centre of each provides only a relatively crude discretisation of the problem.

2.5 Quadratic Boundary Elements

After the constant element, the next simplest is the linear element. Although in two dimensions this still takes the form of a straight line, it has a node at each end and accommodates linear variations of potential and potential gradient along its length. Much of the simplicity of the constant element treatment is thereby lost, but without any improvement in the ability to represent curved boundary shapes. Consequently, if higher-order elements are to be used, quadratic elements are generally preferred to linear elements.

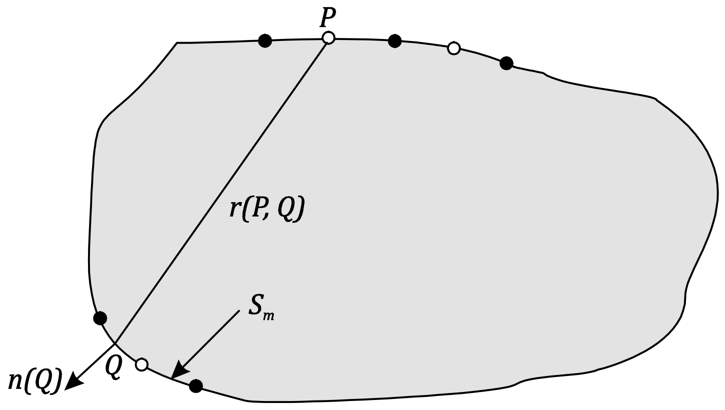


Figure 2.8 Quadratic element discretisation of a two-dimensional solution domain boundary

Figure 2.8 shows a typical arrangement of quadratic line elements on parts of the boundary of a two-dimensional solution domain. Each element has three nodes, one at each end and one at its centre. While the end nodes are shown as solid circles, those at the centres of elements are shown as open circles. The relevant boundary integral equation is Equation 2.29. Note that, because the total number of nodal points is twice the total number of elements, $2M$ equations are generated, M being the total number of elements. The integrations involved in the evaluation of the equation coefficients must in general be carried out numerically.

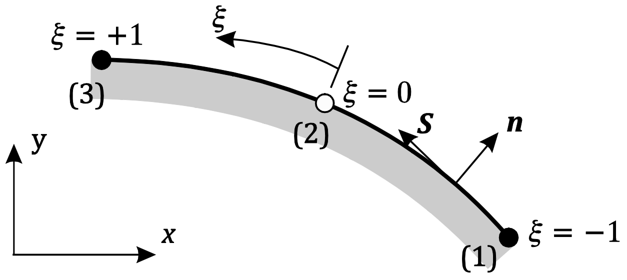


Figure 2.9 A typical quadratic line element

Figure 2.9 shows a typical curved element. Its three nodes are numbered in the direction of integration from 1 to 3, these numbers being local to the particular element. The second node is at the centre of the element as measured along its curved length. A local co-ordinate, ξ, which is often referred to as an intrinsic co-ordinate, follows the curved shape of the element, and is said to be a curvilinear (curved line) co-ordinate. It has its origin at the centre node: that is, $\xi = 0$ at node 2, and takes the values -1 and $+1$ at nodes 1 and 3, respectively. The reason for choosing this particular definition of local co-ordinate is to anticipate the use of Gaussian quadrature (Appendix A) for integration along the elements. The global co-ordinates describing the problem as a whole, rather than just the one element, remain as x and y.

With a quadratic element, the potential ψ, which is defined by its discrete values at the three nodes, ψ_1, ψ_2, and ψ_3 is allowed to vary quadratically with the intrinsic co-ordinate along the element. The distribution can be expressed as

$$\psi(\xi) = N_1(\xi)\psi_1 + N_2(\xi)\psi_2 + N_3(\xi)\psi_3 = \sum_{c=1}^{3} N_c(\xi)\psi_c \tag{2.49}$$

Functions N_1, N_2 and N_3 which interpolate ψ between the nodes are each quadratic, and must be such that

$$\psi(-1) = \psi_1 = N_1(-1)\psi_1 + N_2(-1)\psi_2 + N_3(-1)\psi_3$$

$$\psi(0) = \psi_2 = N_1(0)\psi_1 + N_2(0)\psi_2 + N_3(0)\psi_3$$

$$\psi(+1) = \psi_3 = N_1(+1)\psi_1 + N_2(+1)\psi_2 + N_3(+1)\psi_3$$

For example, N_1 is required to take the values 1, 0 and 0 at the three nodes. So, if

$$N_1(\xi) = a + b\xi + c\xi^2$$

$$1 = a - b + c, \quad 0 = a, \quad 0 = a + b + c$$

from which $a = 0,\ b = -\frac{1}{2},\ c = +\frac{1}{2}$, and

$$N_1(\xi) = \frac{1}{2}\xi(\xi - 1) \tag{2.50}$$

Similarly, N_2 is required to take the values 0, 1 and 0 at the nodes, and N_3 the values 0, 0 and 1, and

$$N_2(\xi) = 1 - \xi^2 \tag{2.51}$$

$$N_3(\xi) = \frac{1}{2}\xi(\xi + 1) \tag{2.52}$$

Functions N_1, N_2 and N_3 are referred to as shape functions. They can also be used to define the variations of the potential gradient normal to the boundary in terms of the nodal point values

$$\frac{\mathrm{d}\psi}{\mathrm{d}n}(\xi) = N_1(\xi)\left(\frac{\mathrm{d}\psi}{\mathrm{d}n}\right)_1 + N_2(\xi)\left(\frac{\mathrm{d}\psi}{\mathrm{d}n}\right)_2 + N_3(\xi)\left(\frac{\mathrm{d}\psi}{\mathrm{d}n}\right)_3 = \sum_{c=1}^{3} N_c\left(\frac{\mathrm{d}\psi}{\mathrm{d}n}\right)_c \tag{2.53}$$

and the variations of the global co-ordinates

$$x(\xi) = N_1(\xi)x_1 + N_2(\xi)x_2 + N_3(\xi)x_3 = \sum_{c=1}^{3} N_c(\xi)x_c \tag{2.54}$$

$$y(\xi) = N_1(\xi)y_1 + N_2(\xi)y_2 + N_3(\xi)y_3 = \sum_{c=1}^{3} N_c(\xi)y_c \tag{2.55}$$

Elements that use the same (shape) functions to interpolate both the geometry and the variations of the problem variables are known as isoparametric elements. Representation of the boundary geometry is now by means of a series of parabolas, each fitted through the three nodes of an element, rather than the straight lines associated with constant and linear elements. The parabolas reduce to straight lines wherever the boundary is genuinely straight.

In order to find the vector normal, $\boldsymbol{n}$, to the boundary at a point along an element, it is convenient to first find the tangent vector $\boldsymbol{S}$ (Figure 2.9). Moving along the boundary as the intrinsic co-ordinate ξ is varied, the rates of change of the global co-ordinates are $\frac{\mathrm{d}x}{\mathrm{d}\xi}$ and $\frac{\mathrm{d}y}{\mathrm{d}\xi}$, and the tangent vector is

$$\boldsymbol{S} = \frac{\mathrm{d}x}{\mathrm{d}\xi}\boldsymbol{i} + \frac{\mathrm{d}y}{\mathrm{d}\xi}\boldsymbol{j} \tag{2.56}$$

Given Equations 2.54 and 2.55, the rates of change are

$$\frac{\mathrm{d}x}{\mathrm{d}\xi} = \sum_{c=1}^{3} \frac{\mathrm{d}N_c(\xi)}{\mathrm{d}\xi} x_c, \qquad \frac{\mathrm{d}y}{\mathrm{d}\xi} = \sum_{c=1}^{3} \frac{\mathrm{d}N_c(\xi)}{\mathrm{d}\xi} y_c \tag{2.57}$$

and the derivatives of the shape functions are

$$\frac{\mathrm{d}N_1(\xi)}{\mathrm{d}\xi} = \xi - \frac{1}{2}, \qquad \frac{\mathrm{d}N_2(\xi)}{\mathrm{d}\xi} = -2\xi, \qquad \frac{\mathrm{d}N_3(\xi)}{\mathrm{d}\xi} = \xi + \frac{1}{2} \tag{2.58}$$

The vector normal can be found as the vector product of $\boldsymbol{S}$ and the unit vector in the third global co-ordinate direction, out of the plane of the solution domain

$$\boldsymbol{n} = n_x\boldsymbol{i} + n_y\boldsymbol{j} = \boldsymbol{S} \wedge \boldsymbol{k} = \frac{\mathrm{d}y}{\mathrm{d}\xi}\boldsymbol{i} - \frac{\mathrm{d}x}{\mathrm{d}\xi}\boldsymbol{j} \tag{2.59}$$

where n_x and n_y are the components of $\boldsymbol{n}$ in the x and y directions.

In order to integrate round the boundary it is necessary to change from global co-ordinate S to local intrinsic co-ordinate ξ within each element, by means of the Jacobian of transformation

$$J(\xi) = \frac{\mathrm{d}S}{\mathrm{d}\xi} = \frac{\sqrt{(\mathrm{d}x)^2+(\mathrm{d}y)^2}}{\mathrm{d}\xi} = \sqrt{\left(\frac{\mathrm{d}x}{\mathrm{d}\xi}\right)^2 + \left(\frac{\mathrm{d}y}{\mathrm{d}\xi}\right)^2} = \sqrt{n_x^2 + n_y^2} \tag{2.60}$$

This Jacobian can also be used to define the components of the unit normal

$$\hat{\boldsymbol{n}} = \frac{\boldsymbol{n}}{\sqrt{n_x^2+n_y^2}} = \frac{n_x}{J(\xi)}\boldsymbol{i} + \frac{n_y}{J(\xi)}\boldsymbol{j} = \hat{n}_x\boldsymbol{i} + \hat{n}_y\boldsymbol{j} \tag{2.61}$$

Introducing the transformation from global to intrinsic co-ordinate, the boundary integral equation, Equation 2.29, becomes

$$\pi\psi(P) + \sum_{m=1}^{M} \int_{-1}^{+1} \psi(Q) \frac{\mathrm{d}}{\mathrm{d}n(Q)} \left\{ \ln\left[\frac{1}{r(P,Q)}\right] \right\} J(\xi)\mathrm{d}\xi$$

$$= \sum_{m=1}^{M} \int_{-1}^{+1} \ln\left[\frac{1}{r(P,Q)}\right] \left(\frac{\mathrm{d}\psi}{\mathrm{d}n}\right)_Q J(\xi)\mathrm{d}\xi \tag{2.62}$$

Then introducing the parametric representations, Equations 2.49 and 2.53, for the potential and potential gradient distributions

$$\pi\psi(P) + \sum_{m=1}^{M}\sum_{c=1}^{3} \psi_c \int_{-1}^{+1} \frac{\mathrm{d}}{\mathrm{d}n(Q)} \left\{ \ln\left[\frac{1}{r(P,Q)}\right] \right\} N_c(\xi)\, J(\xi)\, \mathrm{d}\xi$$

$$= \sum_{m=1}^{M}\sum_{c=1}^{3} \left(\frac{\mathrm{d}\psi}{\mathrm{d}n}\right)_c \int_{-1}^{+1} \ln\left[\frac{1}{r(P,Q)}\right] N_c(\xi)\, J(\xi)\, \mathrm{d}\xi \tag{2.63}$$

where c is the number (1, 2 or 3) of the node in element number m.

Of the two kernel functions in Equation 2.63, the second one, on the right hand side of the equation, is straightforward to evaluate, except when points P and Q are in the same element. As in the constant element case, the first kernel is given by

$$\frac{\mathrm{d}}{\mathrm{d}n(Q)} \left\{ \ln\left[\frac{1}{r(P,Q)}\right] \right\} = \frac{\mathrm{d}}{\mathrm{d}n}(-\ln r) = -\frac{1}{r}\frac{\mathrm{d}r}{\mathrm{d}n} = -\frac{\cos\theta}{r} \tag{2.64}$$

where θ is the angle shown in Figure 2.5. Now

$$\cos\theta = \frac{\boldsymbol{r}.\hat{\boldsymbol{n}}}{r} = \frac{(r_x\hat{n}_x + r_y\hat{n}_y)}{r} \tag{2.65}$$

where r_x and r_y are the components of the radius vector, $\boldsymbol{r}$, joining P and Q.

2.5.1 Points *P* and *Q* not in the same element

In general, the required integrations cannot be performed analytically, but may be carried out numerically using a method known as Gaussian quadrature, which is described in Appendix A. Provided point P is not in the element over which integration is required the process is straightforward.

2.5.2 Points *P* and *Q* in the same element

The case where P is in the relevant element requires some care, because when P and Q are coincident both of the kernel functions are singular. The orders of the singularities are r^{-1} and $\ln(r^{-1})$, respectively. Provided that P is not the c th node of the element, however, the shape function $N_c(\xi)$ goes to zero at P, and the products of kernels and shape functions (in Equation 2.63) are not singular at P, and may be integrated normally.

If P is the c th node of the element then, provided that the boundary at this point is smooth, $\boldsymbol{r}$ and $\boldsymbol{n}$ are orthogonal and the first kernel is zero. The diagonal coefficient of $[A]$ is given by Equation 2.45. If the boundary is not smooth at P, however, then the first kernel is not zero and is difficult to evaluate directly. The diagonal coefficient of $[A]$, which includes the free term, is always obtainable from a summation condition equivalent to Equation 2.47.

$$A_{ii} = -\sum_{\substack{j=1 \\ j\neq i}}^{2M} A_{ij} \tag{2.66}$$

In other words, the sum of the coefficients in any row of $[A]$ should again be zero.

Still considering P to be the c th node of the element in Equation 2.63, the second kernel requires special treatment to deal with the $\ln(r^{-1})$ singularity, using a modified form of Gaussian quadrature. The radial distance from P to a point Q at (x, y) can be arranged in the form

$$r(P,Q) = \eta R(\xi) \tag{2.67}$$

where $R(\xi)$ is a known function and η is a modified intrinsic co-ordinate with its origin at P. The form of $R(\xi)$ depends on which of the three element nodes P is located at. Co-ordinate η is chosen in each case to conform to the requirements of Gaussian quadrature involving a logarithmic function (Appendix A).

P at the first node

If P is at the first node of the element

$$r(P,Q) = [(x-x_1)^2 + (y-y_1)^2]^{\frac{1}{2}} \tag{2.68}$$

and x and y are defined in terms of intrinsic co-ordinate ξ by Equations 2.54 and 2.55. So

$$x - x_1 = [N_1(\xi) - 1]x_1 + N_2(\xi)x_2 + N_3(\xi)x_3 \tag{2.69}$$

$$y - y_1 = [N_1(\xi) - 1]y_1 + N_2(\xi)y_2 + N_3(\xi)y_3 \tag{2.70}$$

Intrinsic co-ordinate η is chosen to range from 0 at the first node $(\xi = -1)$ to 1 at the third node $(\xi = +1)$, so that

$$\eta = \tfrac{1}{2}(\xi + 1) \qquad \left|\frac{d\xi}{d\eta}\right| = 2 \tag{2.71}$$

$$[N_1(\xi) - 1] = \tfrac{1}{2}(\xi^2 - \xi - 2) = \tfrac{1}{2}(\xi + 1)(\xi - 2) = \eta(\xi - 2) \tag{2.72}$$

$$N_2(\xi) = 1 - \xi^2 = 2\eta(1 - \xi) \tag{2.73}$$

$$N_3(\xi) = \tfrac{1}{2}\xi(\xi + 1) = \eta\xi \tag{2.74}$$

Therefore

$$x - x_1 = \eta[(\xi - 2)x_1 + 2(1 - \xi)x_2 + \xi x_3] \tag{2.75}$$

$$y - y_1 = \eta[(\xi - 2)y_1 + 2(1 - \xi)y_2 + \xi y_3] \tag{2.76}$$

$$R(\xi) = \{[(\xi - 2)x_1 + 2(1 - \xi)x_2 + \xi x_3]^2 + [(\xi - 2)y_1 + 2(1 - \xi)y_2 + \xi y_3]^2\}^{\frac{1}{2}} \tag{2.77}$$

The integral of the second kernel in Equation 2.63 can be expressed as

$$\int_{-1}^{+1} \ln\left[\frac{1}{r(P,Q)}\right] N_c(\xi)\, J(\xi)\, \mathrm{d}\xi = \int_0^1 \ln\left(\frac{1}{\eta}\right) N_c(\xi) J(\xi) \left|\frac{\mathrm{d}\xi}{\mathrm{d}\eta}\right| \mathrm{d}\eta + \int_{-1}^{+1} \ln\left(\frac{1}{R(\xi)}\right) N_c(\xi) J(\xi) \mathrm{d}\xi \tag{2.78}$$

Because $R(\xi)$ is not zero within the range of integration, the second integral on the right hand side can be evaluated by normal Gaussian quadrature. The first integral on the right hand side, however, involves the singular logarithmic function, but can nevertheless be evaluated using the appropriate quadrature formula described in Appendix A.

P at the second node

If *P* is at the second node of the element

$$r(P,Q) = [(x - x_2)^2 + (y - y_2)^2]^{\frac{1}{2}} \tag{2.79}$$

$$x - x_2 = N_1(\xi)x_1 + [N_2(\xi) - 1]x_2 + N_3(\xi)x_3 \tag{2.80}$$

$$y - y_2 = N_1(\xi)y_1 + [N_2(\xi) - 1]y_2 + N_3(\xi)y_3 \tag{2.81}$$

For integration purposes, the element needs to be divided into two regions, from the second node to the third node ($\xi = 0$ to 1) and from the second node to the first node ($\xi = 0$ to –1). Between the second and third nodes the intrinsic co-ordinate η_1 is chosen as

$$\eta_1 = \xi \qquad \left|\frac{\mathrm{d}\xi}{\mathrm{d}\eta_1}\right| = 1 \tag{2.82}$$

$$\text{and } N_1(\xi) = \frac{1}{2}\xi(\xi - 1) = \frac{1}{2}\eta_1(\xi - 1) \tag{2.83}$$

$$[N_2(\xi) - 1] = -\xi^2 = -\eta_1\xi \tag{2.84}$$

$$N_3(\xi) = \frac{1}{2}\xi(\xi + 1) = \frac{1}{2}\eta_1(\xi + 1) \tag{2.85}$$

Therefore

$$x - x_2 = \eta_1 \left[\tfrac{1}{2}(\xi - 1)x_1 - \xi x_2 + \tfrac{1}{2}(\xi + 1)x_3\right] \tag{2.86}$$

$$y - y_2 = \eta_1 \left[\tfrac{1}{2}(\xi - 1)y_1 - \xi y_2 + \tfrac{1}{2}(\xi + 1)y_3\right] \tag{2.87}$$

$$R(\xi) = \left\{\left[\tfrac{1}{2}(\xi - 1)x_1 - \xi x_2 + \tfrac{1}{2}(\xi + 1)x_3\right]^2 + \left[\tfrac{1}{2}(\xi - 1)y_1 - \xi y_2 + \tfrac{1}{2}(\xi + 1)y_3\right]^2\right\}^{\frac{1}{2}} \tag{2.88}$$

Between the second and first nodes the intrinsic co-ordinate is chosen as

$$\eta_2 = -\xi \qquad \left|\frac{d\xi}{d\eta_2}\right| = 1 \tag{2.89}$$

and $R(\xi)$ is as defined in Equation 2.88.

The integral of the second kernel in Equation 2.63 can be expressed as

$$\int_{-1}^{+1} \ln\left[\frac{1}{r(P,Q)}\right] N_c(\xi)\, J(\xi) d\xi = \int_0^1 \ln\left(\frac{1}{\eta_1}\right) N_c(\xi) J(\xi) \left|\frac{d\xi}{d\eta_1}\right| d\eta_1 +$$

$$\int_0^1 \ln\left(\frac{1}{\eta_2}\right) N_c(\xi) J(\xi) \left|\frac{d\xi}{d\eta_2}\right| d\eta_2 + \int_{-1}^{+1} \ln\left(\frac{1}{R(\xi)}\right) N_c(\xi) J(\xi) d\xi \tag{2.90}$$

The third integral on the right hand side can be evaluated by normal Gaussian quadrature, while the first and second involve the singular logarithmic function and must be evaluated using the appropriate quadrature formula.

P at the third node

If P is at the third node of the element

$$r(P,Q) = \left[(x - x_3)^2 + (y - y_3)^2\right]^{\frac{1}{2}} \tag{2.91}$$

$$x - x_3 = N_1(\xi)x_1 + N_2(\xi)x_2 + [N_3(\xi) - 1]x_3 \tag{2.92}$$

$$y - y_3 = N_1(\xi)y_1 + N_2(\xi)y_2 + [N_3(\xi) - 1]y_3 \tag{2.93}$$

Intrinsic co-ordinate η is chosen to range from 0 at the third node $(\xi = +1)$ to 1 at the first node $(\xi = -1)$, so that

$$\eta = \tfrac{1}{2}(1 - \xi) \qquad \left|\frac{d\xi}{d\eta}\right| = 2 \tag{2.94}$$

$$N_1(\xi) = \tfrac{1}{2}\xi(\xi - 1) = -\eta\xi \tag{2.95}$$

$$N_2(\xi) = 1 - \xi^2 = 2\eta(1+\xi) \tag{2.96}$$

$$[N_3(\xi) - 1] = \frac{1}{2}(\xi^2 + \xi - 2) = \frac{1}{2}(\xi - 1)(\xi + 2) = -\eta(\xi + 2) \tag{2.97}$$

Therefore

$$x - x_3 = \eta[-\xi x_1 + 2(1+\xi)x_2 - (\xi+2)x_3] \tag{2.98}$$

$$y - y_3 = \eta[-\xi y_1 + 2(1+\xi)y_2 - (\xi+2)y_3] \tag{2.99}$$

$$R(\xi) = \{[-\xi x_1 + 2(1+\xi)x_2 - (\xi+2)x_3]^2 + [-\xi y_1 + 2(1+\xi)y_2 - (\xi+2)y_3]^2\}^{\frac{1}{2}} \tag{2.100}$$

The integral of the second kernel in Equation 2.63 can be expressed as

$$\int_{-1}^{+1} \ln\left[\frac{1}{r(P,Q)}\right] N_c(\xi)\, J(\xi)\, \mathrm{d}\xi = \int_0^1 \ln\left(\frac{1}{\eta}\right) N_c(\xi) J(\xi) \left|\frac{\mathrm{d}\xi}{\mathrm{d}\eta}\right| \mathrm{d}\eta + \int_{-1}^{+1} \ln\left(\frac{1}{R(\xi)}\right) N_c(\xi) J(\xi) \mathrm{d}\xi \tag{2.101}$$

The second integral on the right hand side can be evaluated by normal Gaussian quadrature, while the first involves the singular logarithmic function and must be evaluated using the appropriate quadrature formula.

2.6 Scaling

Whether using either constant or quadratic boundary elements, or indeed any other type, spurious results can sometimes be obtained if the average domain dimension is of the order of unity. This makes the typical radial distance between nodes to be of order unity, at which value the logarithmic kernel function $\ln\left(\frac{1}{r}\right)$ is zero. This difficulty can be simply avoided by ensuring that all distances between nodes are either less than or greater than unity. Of the two, the first is easier to achieve, by first determining the maximum dimension of the problem (the maximum distance between any pair of nodes) and using this to scale all the problem dimensions to be less than unity. Once the solution to the scaled problem has been obtained, the scaling can be removed.

An interesting side effect of this scaling is that problems are in effect being solved in the terms of dimensionless ratios between actual dimensions and the maximum dimensions, thereby avoiding the issue of what is meant by the logarithm of a dimensioned quantity.

2.7 Solving the Linear Equations

Using either constant or quadratic boundary elements the final outcome of the analysis is a set of linear equations of the form

$$[A][x] = [b] \tag{2.102}$$

where $[A]$ and $[b]$ are respectively a matrix and column vector of known coefficients. Few if any of these coefficients are zero in magnitude. This feature has an important bearing on the best choice of method of solution. For example, iterative methods such as Gauss-Seidel are not suitable (and probably would not converge). Direct elimination-type methods are appropriate, and one of the most common, Gaussian elimination, is chosen here. The method is described in detail, including an appropriate computer subprogram, in Appendix B.

2.8 Solving Poisson's Equation

So far in this chapter attention has been focussed on solving Laplace's equation, Equation 2.2, rather than the more general Poisson's equation, Equation 2.1

$$\frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2} = \nabla^2\psi = f_1(x,y) \tag{2.1}$$

With a non-zero function on the right hand side, integration of this function is required. For a general function of position, which cannot be integrated analytically, numerical integration over the solution domain is required. From the problems reviewed in Chapter 1, however, it is clear that the special case of constant f_1 (independent of co-ordinates x and y) is one that embraces many problems of practical interest in engineering.

The solution to Poisson's equation can be expressed as

$$\psi = \psi_L + \psi_{PI} \tag{2.103}$$

where ψ_L is the solution to Laplace's equation ($f_1 = 0$) and ψ_{PI} is any function (a particular integral) which satisfies

$$\nabla^2 \psi_{PI} = f_1 \tag{2.104}$$

One possible form is

$$\psi_{PI} = c_1 x^2 + c_2 y^2 \tag{2.105}$$

where c_1 and c_2 are constants which must be such that

$$2c_1 + 2c_2 = f_1 \tag{2.106}$$

A possible choice of constants is

$$c_1 = c_2 = \frac{f_1}{4} \tag{2.107}$$

Because the total solution expressed in Equation 2.103 must satisfy the applied boundary conditions, it means that ψ_L, the solution to Laplace's equation, must satisfy these conditions with the boundary conditions implicit in Equation 2.105 subtracted. In other words, at any point on the boundary at which potential is prescribed, from this prescribed value must be subtracted the value of

$$\psi_{PI} = \frac{f_1}{4}(x^2 + y^2) \tag{2.108}$$

At any point on the boundary at which potential gradient is prescribed, from this prescribed value must be subtracted the value of

$$\frac{\mathrm{d}\psi_{PI}}{\mathrm{d}n} = \frac{\partial \psi_{PI}}{\partial x}\hat{n}_x + \frac{\partial \psi_{PI}}{\partial y}\hat{n}_y = \frac{f_1}{2}\left(x\hat{n}_x + y\hat{n}_y\right) \tag{2.109}$$

where $\hat{n}_x$ and $\hat{n}_x$ are the components in the x and y directions of the unit outward normal vector to the boundary at the point concerned.

So Poisson's equation can be solved by the boundary element method as though it were Laplace's equation, by simply modifying boundary conditions. Once the solution is obtained, the contributions of the particular integral to the computed boundary values are added back in.

With an $f_1(x,y)$ function other than constant, the same kind of procedure can be applied, provided a particular integral equivalent to Equation 2.105 can be found by analytical integration of $f_1(x,y)$. If this is not possible then numerical integration over the domain must be used.

Problems

2.1 Show that the fundamental solution defined by Equation 2.4 satisfies Laplace's equation in plane polar co-ordinates

$$\nabla^2\phi = \frac{\partial^2\phi}{\partial r^2} + \frac{1}{r}\frac{\partial\phi}{\partial r} + \frac{\partial^2\phi}{\partial\theta^2} = 0$$

2.2 Define the constants a_1, a_2 and a_3 in the general potential boundary condition Equation 2.3 in terms of the heat transfer parameters in Equation 1.57.

2.3 For a linear boundary element with a node at each end, the distribution of potential can be expressed as

$$\psi(\xi) = N_1(\xi)\psi_1 + N_2(\xi)\psi_2$$

where ψ_1 and ψ_2 are the values at the first node where ξ=-1 and the second node where ξ=+1, respectively. Derive expressions for the shape functions N_1 (ξ) and N_2 (ξ).

2.4 Show that the quadratic element shape functions are such that

$$N_1(\xi) + N_2(\xi) + N_3(\xi) = 1$$

for all values of the intrinsic co-ordinate.

2.5 Show that if a quadratic boundary element is a straight line, the quadratic interpolations of geometry implied by Equations 2.54 and 2.55 reduce to the appropriate linear forms.

2.6 For a straight line boundary element, show that the Jacobian of co-ordinate transformation defined by Equation 2.60 is constant along the element, and find its magnitude in terms of the global co-ordinates of the end nodes of the element.

2.7 A quadratic boundary element is used to approximate the arc of a circle, with its three nodes located on the arc. The arc subtends an angle of either (a) $90°$ or (b) $45°$ at the centre of the circle. In each case find the error in the distance from the centre of the circle, expressed as a percentage of the true arc radius, at a point half way between two of the nodes.

2.8 Show that for a straight quadratic element the function $R(\xi)$ in Equations 2.67 and 2.77 reduces to the length of the element, and therefore cannot be zero.

2.9 Derive a suitable particular integral for Poisson's equation, Equation 2.1, when

$$f_1(x, y) = Ex + Fy + G$$

where E, F and G are constants. What changes to the boundary conditions for the boundary element solution to the corresponding Laplace equation should be made?

3 Constant Boundary Element Program for Potential Problems

In this chapter a computer program to implement the constant boundary element formulation for two-dimensional potential problems developed in Chapter 2 is presented and described in detail. It is then used to solve some typical problems to demonstrate the capabilities of the method, particularly problems where exact analytical solutions are available for direct comparison.

The programming language used is Fortran, whose name is derived from **For**mula **tran**slation, and which is particularly suitable for engineering and scientific applications. Despite being somewhat unfashionable, the fact remains that the majority of engineering computer programs currently in use are written in Fortran. The programs presented here were run using a Fortran 95 compiler. While more recent versions of Fortran have been published, the enhancements they offer are not of significance to the type of simple programming of numerical computation procedures employed here.

For readers who prefer to use Matlab, a translation is provided in Appendix C.

Digital computers, whether desktop or laptop, work in binary arithmetic. Both numbers and characters are represented and stored as a number of binary digits or 'bits', and these bits are grouped together in 'bytes' (1 byte = 8 bits). Individual numbers are usually represented by either 32 bits (4 bytes) or 64 bits (8 bytes) depending on both the computer and whether the double precision option is selected in the Fortran program. If the number of bits is small the precision of stored numbers is relatively low and significant roundoff errors may be accumulated in the course of a calculation, depending on the method of calculation used. All the case studies described in this book were run using 64-bit number storage.

The style of writing programs should be such as to make the coding straightforward to follow and check, and at the same time efficient in terms of execution time and memory. With these requirements in mind the programs in this book use variable names which are readily identifiable with the physical or mathematical quantities they represent. Whenever possible the same names are used throughout, their definitions being listed at the beginning of the book. The programs are divided into relatively short subprograms which can be written, developed and tested separately. Also, comment statements are used liberally, both to explain the coding and to separate successive sets of statements for improved readability. For the same reason, a uniform system of statement numbering is used within each subprogram, input and output FORMAT statements being numbered from 51 and 61 respectively. This last convention is a hangover from the Fortran tradition in which the (magnetic tape) unit number for input was 5, while that for output was 6, the same digits being used to start the corresponding FORMAT statement numbers.

Using either a desktop or laptop, input data for a program can be entered from the keyboard as required. Alternatively, a data file can be prepared and stored before the program is run. In this book, the latter approach is adopted. This is because in practice programs need to be run several times, with mostly the same input data, requiring only small changes to the data file. Similarly, computed results can be output to either the computer screen or a stored file. The latter is preferred here because the file provides a potentially permanent record. The convention is adopted of naming the input file DATA and the main output file RESULTS.

3.1 Program BEM2PC

The program name indicates that it is for **b**oundary **e**lement **m**ethod analysis of **two**-dimensional **p**otential problems using **c**onstant elements. Each of the subprogram units which make up the whole are described in turn. The Preface explains how the full program can be accessed as a single file.

If the program appears long and complex, it is mainly because of the needs of data input, testing and output. The boundary element core is quite compact. Much of the data handling is concerned with the definition of the arrangement of elements on the boundary (or boundaries) of the solution domain, and the application of boundary conditions to them. The simplest approach would be to enter the co-ordinates of the end points of each and every element, followed by the type and magnitude of the relevant boundary condition applied to each. Indeed, this is still an option. But with potentially hundreds of elements, this would be tedious. Instead, the strategy adopted is to divide each boundary up into a series of boundary segments, which are either straight lines or circular arcs. The number of elements within a segment can then be chosen, and varied easily, and from which the program generates all the element geometric data. The elements on a segment do not have to be uniform in size, but can be varied in length by a constant ratio between successive elements. In the present version of the program, each segment is subject to only one uniform boundary condition, be it prescribed potential, prescribed potential gradient, or prescribed mixed condition connecting potential and potential gradient. The program distributes this condition to all the elements involved. Consequently, the ends of segments are conveniently defined as points where there is a significant change in either shape (a corner, for example) or boundary condition. How this works in practice is demonstrated later in this chapter.

3.1.1 Main program

At the beginning of the program is a storage module named SHAREDDATA2PC which allows stored data to be accessed and shared by all those subprograms that require it (by means of a USE statement). The dimensioned array sizes in the module allow for up to 1000 constant boundary elements on up 10 different boundaries forming the solution domain. The maximum number of linear equations to solved is 1000. A dictionary of the variable names used is provided at the beginning of the book.

The main program named BEM2PC is designed mainly to call each of the other subprograms in turn. It does, however, also serve to name the files with which the program communicates via OPEN statements. File DATA, which is addressed as file number 5 in the program, serves to supply the input data which defines the current problem. The main output of results is to file RESULTS, addressed as file number 6. Element mesh data, on the other hand, are output to file MESHRES (mesh results) and numbered 7. The term mesh is borrowed from finite elements, although for two-dimensional boundary element analysis the mesh takes the form of a line following the boundary. The rationale behind splitting data output between two files is that, in a typical problem once the mesh data are established and checked, attention turns to the results of the computation, which will often require several runs, looking at file RESULTS each time. Mesh data, which generally involve a lot of information, are available in MESHRES but need not be accessed every time the program is run.

```
      MODULE SHAREDDATA2PC
!
!  MODULE STORING SHARED DATA.
!
      REAL :: XEEND(1010),YEEND(1010),XNODE(1000),YNODE(1000)
      REAL :: XSEND(1000),YSEND(1000),UNX(1000),UNY(1000)
      REAL :: F1,XC,YC,PSI(1000),DPSI(1000),ALPHA(1000),BETA(1000)
      REAL :: PSISEG(1000),DPSISEG(1000),ALPHASEG(1000),BETASEG(1000)
      REAL :: STORE(1000),PSIPI(1000),DPSIPI(1000),PSIT(1000)
      REAL :: DPSIT(1000),FLOWSEG(1000)
      REAL :: PI,A(1000,1001),MAXL,ELENGTH(1000)
      INTEGER :: NEL,NNP,MAXNNP,MAXNB,NEP1(1000),NEP2(1000),NEEND
      INTEGER :: NBOUND,NSEGTOT,NNPB(10),NSEGB(10)
      INTEGER :: NBCP,NBCD,NBCM,NBCT,IBC(1000)
      INTEGER :: ISEGBC(1000),IFIRST(1000),ILAST(1000)
      INTEGER :: ISEGEND(1000),ISEGELEM(1000)
      END MODULE SHAREDDATA2PC

      PROGRAM  BEM2PC
!
!  PROGRAM FOR SOLVING TWO DIMENSIONAL POTENTIAL PROBLEMS BY THE BOUNDARY
!  ELEMENT METHOD USING CONSTANT ELEMENTS.
!
      USE SHAREDDATA2PC
      OPEN(5,FILE="DATA")
      OPEN(6,FILE="RESULTS")
      OPEN(7,FILE="MESHRES")
      PI=4.0*ATAN(1.)
```

```
!
!  DEFINE THE MAXIMUM PROBLEM SIZE PERMITTED BY THE ARRAY DIMENSIONS.
      MAXNNP=1000
      MAXNB=10
!
!  INPUT THE PROBLEM TITLE AND TYPE.
      CALL  INTITLE
!
!  INPUT OR GENERATE THE MESH DATA.
      CALL  MESHC
!
!  OUTPUT THE MESH DATA.
      CALL  MSHOUT
!
!  INPUT, PROCESS AND OUTPUT THE BOUNDARY CONDITIONS.
      CALL  BCS
!
!  IF GOVERNING EQUATION IS POISSON TYPE, MODIFY THE BOUNDARY CONDITIONS.
      CALL POISSON
!
!  FORM THE COEFFICIENT MATRIX AND APPLY THE BOUNDARY CONDITIONS.
      CALL  FRMTRX
!
!  SOLVE THE LINEAR EQUATIONS.
      MAXNNPP1=MAXNNP+1
      CALL  ELIMIN(A,PSI,NNP,MAXNNP,MAXNNPP1,IFLAG)
      IF(IFLAG == 1) THEN
        WRITE(6,61)
 61     FORMAT(/ "MATRIX ILL-CONDITIONING DETECTED IN EQUATION SOLVER")
        STOP
      END IF
!
!  OUTPUT NODAL POINT VALUES OF POTENTIAL AND POTENTIAL GRADIENT,
!  ALSO POTENTIAL FLOWS ACROSS BOUNDARY SEGMENTS.
      CALL  OUTPUT
!
!  COMPUTE VALUES OF POTENTIAL AT INTERNAL POINTS.
      CALL INTERNAL
!
      STOP
      END PROGRAM BEM2PC
```

After defining the maximum numbers of nodal points and elements (MAXNNP) and boundaries (MAXNB) permitted by the array dimensions, the main program calls the following subprograms in turn:

INTITLE for the problem title,
MESHC to input and create the mesh data,
MSHOUT to write out the mesh data,
BCS for the boundary conditions,
POISSON to compute the particular integral if the problem is of the Poisson type,
FRMTRX to define the $[A]$ and $[B]$ coefficient matrices,
ELIMIN to solve the equations,
OUTPUT to write out the results, and finally
INTERNAL to find values of potential at any desired points within the domain.

Note that if ELIMIN detects a singular or very ill-conditioned matrix a warning is written out and execution terminates. In practice this is most likely to be due to the coefficient matrix being singular. This can happen when the problem is inadequately defined, for example with only potential gradient or mixed boundary conditions, and potential nowhere defined.

3.1.2 Subprogram INTITLE

An alphanumeric title for the problem is first read into TITLE. Next the constant value of function f_1 (Equation 2.1) is read into F1. If this value is zero the problem is of the Laplace type, and an appropriate message is written out. If it is not zero the problem is Poisson type and the global co-ordinates of the origin used to define to particular integral are read into XC and YC, normally chosen to be roughly at the centre of the domain. The value of f_1 together with the origin co-ordinates are then written out.

```
      SUBROUTINE  INTITLE
!
!  SUBPROGRAM TO INPUT PROBLEM TITLE AND TYPE (LAPLACE OR POISSON).
!
      USE SHAREDDATA2PC
      CHARACTER(80) :: TITLE
!
!  INPUT THE PROBLEM TITLE.
      READ(5,FMT="(A80)") TITLE
      WRITE(6,61) TITLE
 61   FORMAT("CONSTANT BOUNDARY ELEMENT SOLUTION FOR",
     &       " TWO DIMENSIONAL POTENTIAL PROBLEM"  // A)
!
!  INPUT THE VALUE OF THE (CONSTANT) F1 FUNCTION IN THE GOVERNING
!  EQUATION.
      READ(5,*) F1
      IF(F1 == 0.) WRITE(6,62)
 62   FORMAT(/ "LAPLACE EQUATION")
      IF(F1 /= 0.) THEN
        WRITE(6,63) F1
 63     FORMAT(/ "POISSON EQUATION,  F1 = ",E12.4,"  CONSTANT")
!
!  INPUT THE COORDINATES OF THE ORIGIN FOR THE PARTICULAR INTEGRAL.
        READ(5,*) XC,YC
        WRITE(6,64) XC,YC
 64     FORMAT(/ "ORIGIN FOR PARTICULAR INTEGRAL:",5X,"X =",E12.4,5X,
     &          "Y =",E12.4)
      END IF
!
      RETURN
      END SUBROUTINE INTITLE
```

3.1.3 Subprogram to input and generate the element mesh data

Subprogram MESHC reads in a minimal amount of information from which to create the mesh of constant boundary elements. The process is complicated by the fact that there are several levels of geometric features. Firstly, boundaries: there will be more than one for a multiply-connected domain, with internal holes within the outer boundary. Secondly, segments of boundaries whose ends are located at changes of either geometry or boundary condition, and can be either straight or follow circular arcs. Thirdly, the ends of elements which are distributed either uniformly or non-uniformly along the segments. Finally, the nodes of the elements, which are located at the mid points between successive element end points.

```
      SUBROUTINE  MESHC
!
!  SUBPROGRAM TO READ IN AND GENERATE THE GEOMETRIC DATA FOR A MESH OF
!  CONSTANT ELEMENTS.
!
      USE SHAREDDATA2PC
!
!  INPUT THE NUMBER OF SEPARATE BOUNDARIES.
      READ(5,*) NBOUND
!
!  TEST THE NUMBER OF BOUNDARIES.
      IF(NBOUND < 1 .OR. NBOUND > MAXNB) THEN
        WRITE(6,61) NBOUND,MAXNB
 61     FORMAT(/ "NBOUND =",I4,2X,"OUTSIDE PERMITTED RANGE 1 TO",I4)
        STOP
      END IF
!
!  FOR EACH BOUNDARY IN TURN INPUT THE NUMBER OF SEGMENTS.
      NNP=0
      IEEND=0
      NSEGTOT=0
      Each boundary in turn: DO IBOUND=1,NBOUND
      NNPB(IBOUND)=0
      READ(5,*) NSEGB(IBOUND)
      NSEGTOT=NSEGTOT+NSEGB(IBOUND)
!
!  TEST THE NUMBER OF SEGMENTS.
      IF(NSEGTOT < 1 .OR. NSEGTOT > MAXNNP) THEN
        WRITE(6,62) NSEGTOT,MAXNNP
 62     FORMAT(/ "NSEGTOT =",I6,2X,"OUTSIDE PERMITTED RANGE 1 TO",I6)
        STOP
      END IF
```

```
!
!  INPUT THE CARTESIAN GLOBAL COORDINATES OF THE END POINTS OF THE
!  SEGMENTS.  TAKE THE END POINTS CONSECUTIVELY, KEEPING THE DOMAIN
!  TO THE LEFT OF THE DIRECTION OF NUMBERING.
      READ(5,*) (XSEND(ISEND),YSEND(ISEND),ISEND=1,NSEGB(IBOUND))
!
!  DEFINE THE FIRST END POINT ON THE CURRENT BOUNDARY.
      IEEND=IEEND+1
      XEEND(IEEND)=XSEND(1)
      YEEND(IEEND)=YSEND(1)
!
!  FOR EACH OF THE SEGMENTS (BETWEEN ENDS 1 AND 2, 2 AND 3, ETC.)
!  INPUT THE RADIUS OF CURVATURE (+VE FOR CONVEX WITH CENTRE OF
!  CURVATURE INSIDE DOMAIN, -VE FOR CONCAVE), THE NUMBER OF
!  ELEMENTS IN THE SEGMENT, AND THE LENGTH RATIO BETWEEN SUCCESSIVE
!  ELEMENTS IN THE DIRECTION OF NUMBERING.
      ISEGMAX=NSEGTOT
      ISEGMIN=ISEGMAX-NSEGB(IBOUND)+1
      Each segment in turn: DO ISEG=ISEGMIN,ISEGMAX
      READ(5,*) RSEG,NELSEG,RATSEG
```

```
!
!  FIND AND TEST NUMBER OF NODES/ELEMENTS SO FAR.
      NNP=NNP+NELSEG
      NNPB(IBOUND)=NNPB(IBOUND)+NELSEG
      IF(NNP < 1 .OR. NNP > MAXNNP) THEN
        WRITE(6,63) NNP,MAXNNP
 63     FORMAT(/ "NNP =",I6,2X,"OUTSIDE PERMITTED RANGE 1 TO",I6)
        STOP
      END IF
!
!  FIRST AND LAST NODES ON CURRENT SEGMENT.
      ILAST(ISEG)=NNP
      IFIRST(ISEG)=NNP-NELSEG+1
!
!  COORDINATES OF THE FIRST END POINT OF THE SEGMENT.
      ISEND=ISEG-ISEGMIN+1
      XFIRST=XSEND(ISEND)
      YFIRST=YSEND(ISEND)
!
!  COORDINATES OF THE LAST END POINT OF THE SEGMENT.
      ISEND=ISEND+1
      IF(ISEG == ISEGMAX) ISEND=1
      XLAST=XSEND(ISEND)
      YLAST=YSEND(ISEND)
!
!  GENERATE ELEMENT DATA FOR A STRAIGHT SEGMENT.
      IF(RSEG == 0.) THEN
!
!  DEFINE THE ELEMENT END POINT COORDINATES ON THE SEGMENT.
        Each element in turn: DO M=1,NELSEG
        IEEND=IEEND+1
        ISEGEND(IEEND)=ISEG
        IF(RATSEG == 1.) THEN
          XEEND(IEEND)=XFIRST+(XLAST-XFIRST)*FLOAT(M)/FLOAT(NELSEG)
          YEEND(IEEND)=YFIRST+(YLAST-YFIRST)*FLOAT(M)/FLOAT(NELSEG)
        ENDIF
        IF(RATSEG /= 1.) THEN
          XEEND(IEEND)=XFIRST+(XLAST-XFIRST)*(1.-RATSEG**M)
     &                    /(1.-RATSEG**NELSEG)
```

```
            YEEND(IEEND)=YFIRST+(YLAST-YFIRST)*(1.-RATSEG**M)
     &                        /(1.-RATSEG**NELSEG)
          END IF
          END DO Each element in turn
        END IF
!
!  GENERATE ELEMENT DATA FOR A SEGMENT IN THE FORM OF A CIRCULAR ARC.
        IF(RSEG /= 0.) THEN
!
!  LOCATE THE CENTRE OF THE ARC.
          XMID=(XFIRST+XLAST)/2.
          YMID=(YFIRST+YLAST)/2.
          ALSEG=SQRT((XLAST-XFIRST)**2+(YLAST-YFIRST)**2)
          ALPERP2=RSEG**2-(ALSEG/2.)**2
          IF(ABS(ALPERP2) < 1.E-6*RSEG**2) ALPERP2=0.
          IF(ALPERP2 < -1.E-6*RSEG**2) THEN
            WRITE(6,64) ISEG
 64         FORMAT(/ "DATA ERROR FOR SEGMENT NUMBER",I6,
     &             / "NOT POSSIBLE TO CREATE A CIRCULAR ARC")
            STOP
          END IF
          ALPERP=SQRT(ALPERP2)
          UVFLX=(XLAST-XFIRST)/ALSEG
          UVFLY=(YLAST-YFIRST)/ALSEG
          FACT=1.
          IF(RSEG < 0.) FACT=-1.
          XCENT=XMID-ALPERP*UVFLY*FACT
          YCENT=YMID+ALPERP*UVFLX*FACT
!
!  FIND THE ANGLE SUBTENDED THERE BY THE SEGMENT.
          IF(ALPERP /= 0.) ANGSEG=2.*ATAN(ALSEG*0.5/ALPERP)
          IF(ALPERP == 0.) ANGSEG=PI
!
!  DEFINE THE ELEMENT END POINT COORDINATES ON THE SEGMENT.
          ANGFIR=ATAN2(YFIRST-YCENT,XFIRST-XCENT)
          Each element in turn: DO M=1,NELSEG
          IEEND=IEEND+1
          ISEGEND(IEEND)=ISEG
          IF(RATSEG == 1.) ANG=ANGSEG*FLOAT(M)/FLOAT(NELSEG)
          IF(RATSEG /= 1.) ANG=ANGSEG*(1.-RATSEG**M)/(1.-RATSEG**NELSEG)
          IF(RSEG < 0.) ANG=-ANG
          XEEND(IEEND)=XCENT+ABS(RSEG)*COS(ANGFIR+ANG)
          YEEND(IEEND)=YCENT+ABS(RSEG)*SIN(ANGFIR+ANG)
          END DO Each element in turn
        END IF
!
        END DO Each segment in turn
        END DO Each boundary in turn
        NEEND=IEEND
        NEL=NNP
!
!  GENERATE THE ELEMENT POINT DATA AND THE NODAL POINT COORDINATES.
        MMIN=1
        IEP1=0
        IEP2=1
        Each boundary in turn: DO IBOUND=1,NBOUND
        MMAX=MMIN+NNPB(IBOUND)-1
        Each element on current boundary: DO M=MMIN,MMAX
        IEP1=IEP1+1
        IEP2=IEP2+1
```

```
      NEP1(M)=IEP1
      NEP2(M)=IEP2
      XNODE(M)=0.5*(XEEND(IEP1)+XEEND(IEP2))
      YNODE(M)=0.5*(YEEND(IEP1)+YEEND(IEP2))
      ISEGELEM(M)=ISEGEND(IEP2)
      END DO Each element on current boundary
      MMIN=MMAX+1
      IEP1=IEP1+1
      IEP2=IEP2+1
      END DO Each boundary in turn
!
!  FIND AND STORE THE COMPONENTS OF THE UNIT OUTWARD NORMALS AT THE NODES,
!  ALSO THE ELEMENT LENGTHS.
      Each node in turn: DO I=1,NNP
      IEP1=NEP1(I)
      IEP2=NEP2(I)
      EVX=XEEND(IEP2)-XEEND(IEP1)
      EVY=YEEND(IEP2)-YEEND(IEP1)
      UNX(I)=EVY
      UNY(I)=-EVX
      DENOM=SQRT(UNX(I)**2+UNY(I)**2)
      UNX(I)=UNX(I)/DENOM
      UNY(I)=UNY(I)/DENOM
      ELENGTH(I)=SQRT(EVX**2+EVY**2)
      END DO Each node in turn
!
!  DETERMINE THE MAXIMUM DIMENSION OF THE SOLUTION DOMAIN.
      MAXL=0.
      Each node in turn: DO I=1,NNP
      Each other node in turn: DO J=1,NNP
      DIST=SQRT((XNODE(I)-XNODE(J))**2+(YNODE(I)-YNODE(J))**2)
      IF(DIST > MAXL) MAXL=DIST
      END DO Each other node in turn
      END DO Each node in turn
!
      RETURN
      END SUBROUTINE MESHC
```

The number of boundaries is first read into NBOUND. If there is more than one boundary, the order of numbering them is arbitrary. Then, for each boundary in turn, the number of segments forming the boundary is read into NSEGB, followed by the global co-ordinates of the segment end points (taken consecutively) into XSEND and YSEND. The starting point for defining segments on the boundary is arbitrary, but then the order of numbering must be such as to keep the domain on the left: anticlockwise for the outer boundary surrounding the domain, clockwise for a boundary defining a hole in the domain. Note that the first end point, which is an end of both the first and last segment, must only be defined once: the number of end points on a boundary is equal to the number of segments, and the program makes the connection to close the boundary.

Then, for each segment of the current boundary in turn, its radius of curvature, the number of elements on the segment, and the ratio between the lengths of successive elements on the segment are read into RSEG, NELSEG and RATSEG, respectively. A zero value of RSEG indicates a straight segment, while the sign of a non-zero value of RSEG indicates whether the part of the boundary defined by the segment is convex or concave. A positive radius indicates convex, with the centre of curvature inside the domain, while a negative value indicates concave, with the centre outside the domain. A unit value of RATSEG indicates a uniform distribution of elements along the segment, whether it be straight or curved. A value greater than one requires the lengths of successive elements to increase in the direction of segment end numbering, a value less than one requires a similar decrease.

The subprogram uses a number of housekeeping variables to keep track of various totals: IEEND stores the current element end point number, array NNPB the number of nodal points (or elements) on each boundary, and NNP the overall total number of nodal points. Similarly, ISEGMIN and ISEGMAX store the numbers of the first and last segments on the current boundary (the segment numbering runs sequentially from one boundary to the next), while IFIRST and ILAST do the same for nodes (elements) on the current segment. Note that wherever possible tests are applied (and warning messages given) to check that array dimensions are not going to be exceeded. It is also worth noting that arrays concerned with segments are given the same dimensions (1000, in module SHAREDDATA2PC) as those for elements. This is to give the user the freedom to define the geometry and boundary condition of each element individually, by making each segment contain only one element. Two arrays, XEEND and YEEND (element end point co-oordinates) are actually given dimensions of 1010. This is because due to the way element end point co-ordinates are defined it is convenient to have the last point on a boundary distinct from the first point, whose co-ordinates it shares. This means that the number of element end points on a boundary is one more than the number of elements, and there could be as many as 10 boundaries.

The subprogram stores the co-ordinates of the first and last element end points on the current segment and then defines the element end point co-ordinates, storing in XEEND and YEEND. Note that the segment number associated with a particular element end point is stored temporarily in array ISEGEND. Once the element data have been generated this information is transferred from ISEGEND to ISEGELEM, the segment number associated with each element. This information will be required later in the final output of results, when the segment number for each element is required for calculating potential flows across the various segments.

For a straight segment, simple linear interpolation is applied to define the element end points. If x_1 and x_2 (XFIRST and XLAST) are the x global co-ordinates of the first and last segment end points, m (M) is the element counter (from 1 to the number of elements on the segment) and n_s (NELSEG) is the number of elements on the segment, then the x co-ordinate of the second end of the mth element is given by

$$x = x_1 + \frac{m}{n_s}(x_2 - x_1) \tag{3.1}$$

and similarly for the co-ordinate. The position of the first end point on the segment has already been defined as that of the last point of the previous segment, except for the first end point on each boundary, which was defined initially for the boundary. If the distribution of element end points along the segment is not uniform, but has a constant ratio S between the lengths of successive elements (RATSEG in the subprogram), then a different formula is required. Let x^* be the component in the direction of the x length of the first element, then

$$x_2 - x_1 = x^*(1 + S + S^2 + S^3 + \ldots.. + S^{n_s - 1}) \tag{3.2}$$

Multiply through by S

$$S(x_2 - x_1) = x^*(S + S^2 + S^3 + S^4 \ldots.. + S^{n_s})$$

and subtracting

$$(1 - S)(x_2 - x_1) = x^*(1 - S^{n_s}) \tag{3.3}$$

from which

$$x^* = \frac{(1-S)(x_2-x_1)}{(1-S^{n_s})} \tag{3.4}$$

The x co-ordinate of the second end of the m th element is given by

$$x - x_1 = x^*(1 + S + S^2 + S^3 + \ldots\ldots + S^{m-1}) = (x_2 - x_1)\frac{(1 - S^m)}{(1 - S)}$$

$$x = x_1 + (x_2 - x_1)\frac{(1-S^m)}{(1-S^{n_s})} \tag{3.5}$$

and similarly for the y co-ordinate. In the limit as $S \rightarrow 1$, Equation 3.5 reduces to 3.1.

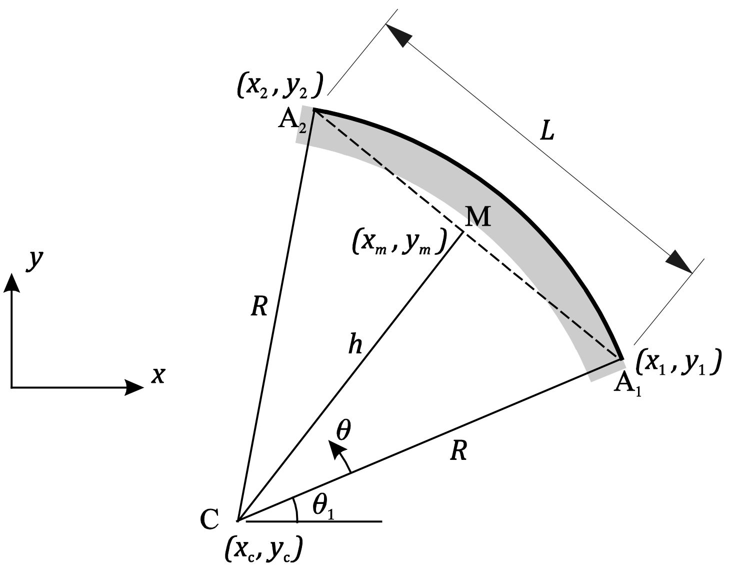


Figure 3.1 Geometry of a boundary segment in the form of a circular arc

The same formulae are applied in the case of a segment in the form of a circular arc, but to angles subtended at the centre of curvature rather than to lengths between element end points. Given that the input data provide only the co-ordinates of the end points of a segment and its radius of curvature, the location of the centre of curvature requires some care. Figure 3.1 shows a boundary segment with end points A_1 and A_2 with co-ordinates (x_1, y_1) and (x_2, y_2), respectively, and radius of curvature R (the centre is within the domain when R is positive). The midpoint of the straight line joining the end points is M, at (x_m, y_m) or (XMID, YMID) in terms of program variables, where

$$x_m = \frac{1}{2}(x_1 + x_2) \text{ and } y_m = \frac{1}{2}(y_1 + y_2) \tag{3.6}$$

The length of the line joining the end points is L (ALSEG)

$$L = \sqrt{}[(x_2 - x_1)^2 + (y_2 - y_1)^2] \tag{3.7}$$

If the centre of curvature is C, at (x_c, y_c) or (XCENT, YCENT), and the length of the perpendicular from C to the line A_1A_2 at M is h (ALPERP) then

$$h^2 = R^2 - \left(\frac{L}{2}\right)^2 \tag{3.8}$$

Before taking the square root to find h, it is necessary to test that the expression for h^2 (ALPERP2) is either positive or zero. A negative value would imply that the distance apart of the end points exceeds twice the radius of curvature: either the points are too far apart or the radius of curvature is too small. It is then not possible to create a circular arc. The particular case of $h^2 = 0$ deserves some care, because it corresponds to the case of centre C at point M, giving a semi-circular segment, which can often occur in practice. Because floating point arithmetic cannot be carried out with perfect precision, the test applied in the program is $h^2 < 10^{-6}R^2$. If satisfied, h^2 is assumed to be zero and the segment is treated as a semi circle. The program does not accommodate circular arc segments whose subtended angles exceed 180°, because more information would be required to detect them. An attempt to enter data for such a segment would result in a segment with an angle less than $180°$, which could be identified by inspecting the mesh data output.

The position of the centre of curvature is found by first defining the unit vector in the direction from point A_1 and to point A_2

$$\hat{\boldsymbol{v}} = \hat{v}_x \boldsymbol{i} + \hat{v}_y \boldsymbol{j} \text{ where } \hat{v}_x = \frac{(x_2 - x_1)}{L} \text{ and } \hat{v}_y = \frac{(y_2 - y_1)}{L}$$

The unit vector perpendicular to this towards C is

$$\hat{\boldsymbol{u}} = -\hat{v}_y \boldsymbol{i} + \hat{v}_x \boldsymbol{j} \tag{3.9}$$

and would be the negative of this if R is negative, implying that C is outside the domain. The co-ordinates of C (XCENT, YCENT) are therefore

$$x_c = x_m - h\hat{v}_y \qquad y_c = y_m + h\hat{v}_x \tag{3.10}$$

Let θ be the angular polar co-ordinate centred at point C in Figure 3.1, measured from the line CA_1. The angular position θ_1 (ANGFIR) of CA_1 (to the first segment end point) relative to the x axis is given by

$$\tan\theta_1 = \frac{(y_1 - y_c)}{(x_1 - x_c)} \tag{3.11}$$

and the angular extent of the segment, $\Delta\theta$ (ANGSEG), by

$$\tan\left(\frac{\Delta\theta}{2}\right) = \frac{L}{2h} \tag{3.12}$$

If the elements on the segment are of uniform size, the angular co-ordinate of the second end of the mth element is

$$\theta = \frac{m}{n_s}\Delta\theta \tag{3.13}$$

If the lengths of the elements increase by a constant ratio S (RATSEG), the angular co-ordinate of the second end of the mth element is

$$\theta = \frac{(1 - S^m)}{(1 - S^{n_s})}\Delta\theta \tag{3.14}$$

In either case, the corresponding global coordinates are

$$x = x_c + |R|\cos(\theta_1 + \theta) \tag{3.15}$$

$$y = y_c + |R|\sin(\theta_1 + \theta) \tag{3.16}$$

Given all the element end point data for all the boundaries, the subprogram then goes on to define the nodes as the midpoints of the straight lines between the ends of the elements and stores the nodal co-ordinates in XNODE and YNODE. The numbers of the corresponding first and second end points are stored in NEP1 and NEP2. Then the element lengths (ELENGTH) are computed, and the components of the unit outward normals (UNX and UNY) are found. Finally, the maximum dimension (MAXL) of the solution domain is found as the maximum distance between any pair of nodal points, for subsequent use in scaling as described in Section 2.6.

3.1.4 Mesh data output subprogram

The subprogram MSHOUT serves to write out the geometric data for the mesh to file MESHRES. Following the number of elements, number of nodes, and number of element ends, the numbers and co-ordinates of the end points are written out, two sets to a line. Then the element numbers, end point numbers, nodal co-ordinates and components of the unit outward normals to the elements are written out.

```
      SUBROUTINE  MSHOUT
!
!  SUBPROGRAM TO WRITE OUT THE MESH DATA.
!
      USE SHAREDDATA2PC
!
!  OUTPUT THE NUMBERS OF ELEMENTS AND NODES, ALSO THE ELEMENT END
!  POINT COORDINATE DATA.
      WRITE(7,71) NEL,NNP,NEEND,
     &            (IEEND,XEEND(IEEND),YEEND(IEEND),IEEND=1,NEEND)
 71   FORMAT(/ "GEOMETRIC DATA FOR THE MESH" //
     &   10X,"NUMBER OF ELEMENTS =",I6 //
     &   10X,"NUMBER OF NODAL POINTS =",I6 //
     &   10X,"NUMBER OF ELEMENT END POINTS =",I6 //
     &       "COORDINATES OF ELEMENT END POINTS" //
     &       2("     I     X           Y      ") /
     &       2(I6,2E12.4))
!
!  OUTPUT THE ELEMENT END POINT NUMBERS, NODAL COORDINATES AND
!  COMPONENTS OF THE UNIT OUTWARD NORMALS AT THE NODES.
      WRITE(7,72) (M,NEP1(M),NEP2(M),XNODE(M),YNODE(M),UNX(M),
     &             UNY(M),M=1,NEL)
 72   FORMAT(/ "ELEMENT END POINT NUMBERS, NODAL COORDINATES",
     &  " AND UNIT NORMAL COMPONENTS"      //
     & ("     M   NEP1  NEP2   X(NODE)      Y(NODE)",
     &  "     UNX          UNY") / (3I6,4E12.4))
!
!  SCALE THE ELEMENT END POINT AND NODAL POINT COORDINATES.
      Each element end point in turn: DO IEEND=1,NEEND
      XEEND(IEEND)=XEEND(IEEND)/MAXL
      YEEND(IEEND)=YEEND(IEEND)/MAXL
      END DO Each element end point in turn
      Each node in turn: DO I=1,NNP
      XNODE(I)=XNODE(I)/MAXL
      YNODE(I)=YNODE(I)/MAXL
      END DO Each node in turn
!
      RETURN
      END SUBROUTINE MSHOUT
```

Finally, the co-ordinates of the element end points and nodes are scaled using the maximum dimension of the domain.

ie business school
#1 EUROPEAN BUSINESS SCHOOL FINANCIAL TIMES 2013
#gobeyond
MASTER IN MANAGEMENT
Because achieving your dreams is your greatest challenge. IE Business School's Master in Management taught in English, Spanish or bilingually, trains young high performance professionals at the beginning of their career through an innovative and stimulating program that will help them reach their full potential.
Choose your area of specialization.
Customize your master through the different options offered.
Global Immersion Weeks in locations such as Rio de Janeiro, Shanghai or San Francisco.
Because you change, we change with you.
www.ie.edu/master-management
mim.admissions@ie.edu
Click on the ad to read more

3.1.5 Subprogram for applying the boundary conditions

The subprogram BCS serves to apply boundary conditions of either the prescribed potential, prescribed potential gradient, or mixed (linear relationship between potential and potential gradient) types. As already indicated, it is assumed that each segment of elements has a uniform boundary condition applied to it, which is an important consideration when defining the segments. The total numbers of segments subject to the three types of condition are first read into variables NBCP, NBCD and NBCM, respectively. In the case of prescribed normal derivatives (gradients) it is only necessary to include those segments subject to non-zero gradients. Storage arrays including ALPHA and BETA for the mixed boundary condition are set to zero for each nodal point. Array IBC stores the type of boundary condition (1, 2 or 3 for three types) for each node. In the absence of other information, the condition at a node is assumed to be zero potential gradient. This is perhaps the most common single condition encountered in practice, for example on a line of symmetry, a thermally insulated boundary in a heat conduction problem, or an impermeable boundary in a fluid mechanics problem. The default boundary condition type is therefore set as 2, with the corresponding zero value of gradient being stored in array STORE.

```
      SUBROUTINE  BCS
!
! SUBPROGRAM TO INPUT, PROCESS AND OUTPUT THE BOUNDARY CONDITIONS.
!
      USE SHAREDDATA2PC
!
! INPUT THE NUMBERS OF SEGMENTS SUBJECT TO EACH TYPE OF BOUNDARY
! CONDITION.
!     NBCP - PRESCRIBED POTENTIAL.
!     NBCD - NON-ZERO PRESCRIBED NORMAL DERIVATIVE OF POTENTIAL.
!     NBCM - MIXED BOUNDARY CONDITION.
! ANY SEGMENT NOT INCLUDED IS ASSUMED TO BE SUBJECT TO A ZERO
! NORMAL DERIVATIVE OF POTENTIAL.
!
      READ(5,*) NBCP,NBCD,NBCM
!
! TEST THESE BOUNDARY CONDITION NUMBERS.
      NBCT=NBCP+NBCD+NBCM
      IF(NBCP < 0 .OR. NBCP > MAXNNP .OR. NBCD < 0 .OR. NBCD > MAXNNP
     &   .OR. NBCM < 0 .OR. NBCM > MAXNNP .OR. NBCT < 0 .OR.
     &   NBCT > MAXNNP) THEN
        WRITE(6,61) NBCP,NBCD,NBCM,NBCT,MAXNNP
 61     FORMAT(/ "NBCP =",I6,3X,"NBCD =",I6,3X,"NBCM =",I6
     &         / "NBCT =",I6,3X,"OUTSIDE PERMITTED RANGE 0 TO",I6)
      STOP
      END IF
```

```
!
!  INITIALISE THE BOUNDARY CONDITION STORAGE ARRAYS.
      Each node in turn: DO I=1,NNP
      IBC(I)=2
      STORE(I)=0.
      ALPHA(I)=0.
      BETA(I)=0.
      END DO Each node in turn
!
!  INPUT, STORE AND OUTPUT THE PRESCRIBED POTENTIAL BOUNDARY CONDITIONS.
      IF(NBCP > 0) THEN
        READ(5,*) (ISEGBC(IBCP),PSISEG(IBCP),IBCP=1,NBCP)
        WRITE(6,62)
 62     FORMAT(/ "PRESCRIBED POTENTIAL BOUNDARY CONDITIONS")
        Each segment with prescribed potential: DO IBCP=1,NBCP
        ISEG=ISEGBC(IBCP)
        IF(ISEG < 1 .OR. ISEG > NSEGTOT) THEN
          WRITE(6,63) ISEG,NSEGTOT
 63       FORMAT(/ "ISEG = ",I6,2X,"OUTSIDE PERMITTED RANGE 1 TO",I6)
          STOP
        END IF
        Each node on current segment: DO I=IFIRST(ISEG),ILAST(ISEG)
        IBC(I)=1
        STORE(I)=PSISEG(IBCP)
        END DO Each node on current segment
        WRITE(6,64) PSISEG(IBCP),IFIRST(ISEG),ILAST(ISEG)
 64     FORMAT(/ "POTENTIAL =",E12.4,5X,"AT NODES ",I6,5X,"TO ",I6)
        END DO Each segment with prescribed potential
      END IF
!
!  INPUT, STORE AND OUTPUT THE PRESCRIBED POTENTIAL GRADIENT BOUNDARY
!  CONDITIONS.
      IF(NBCD > 0) THEN
        READ(5,*) (ISEGBC(IBCD),DPSISEG(IBCD),IBCD=1,NBCD)
        WRITE(6,65)
 65     FORMAT(/ "PRESCRIBED POTENTIAL GRADIENT BOUNDARY CONDITIONS")
        Each segment with prescribed potential gradient: DO IBCD=1,NBCD
        ISEG=ISEGBC(IBCD)
        IF(ISEG < 1 .OR. ISEG > NSEGTOT) THEN
```

```
          WRITE(6,63) ISEG,NSEGTOT
          STOP
        END IF
        Each node on current segment: DO I=IFIRST(ISEG),ILAST(ISEG)
        IBC(I)=2
        STORE(I)=DPSISEG(IBCD)
        END DO Each node on current segment
        WRITE(6,66) DPSISEG(IBCD),IFIRST(ISEG),ILAST(ISEG)
 66     FORMAT(/ "GRADIENT =",E12.4,5X,"AT NODES ",I6,5X,"TO ",I6)
        END DO Each segment with prescribed potential gradient
      END IF
!
!  INPUT, STORE AND OUTPUT THE MIXED BOUNDARY CONDITIONS.
!  ASSUMED FORM - DPSI=ALPHA*PSI+BETA.
      IF(NBCM > 0) THEN
        READ(5,*)(ISEGBC(IBCM),ALPHASEG(IBCM),BETASEG(IBCM),IBCM=1,NBCM)
        WRITE(6,67)
 67     FORMAT(/ "PRESCRIBED MIXED BOUNDARY CONDITIONS")
        Each segment with prescribed mixed condition: DO IBCM=1,NBCM
        ISEG=ISEGBC(IBCM)
```

```
          IF(ISEG < 1 .OR. ISEG > NSEGTOT) THEN
            WRITE(6,63) ISEG,NSEGTOT
            STOP
          END IF
          Each node on current segment: DO I=IFIRST(ISEG),ILAST(ISEG)
          IBC(I)=3
          ALPHA(I)=ALPHASEG(IBCM)
          BETA(I)=BETASEG(IBCM)
          END DO Each node on current segment
          WRITE(6,68)ALPHASEG(IBCM),BETASEG(IBCM),IFIRST(ISEG),ILAST(ISEG)
   68     FORMAT(/ "ALPHA =",E12.4,5X,"BETA =",E12.4,5X,
     &             "AT NODES ",I6,5X,"TO ",I6)
          END DO Each segment with prescribed mixed condition
        END IF
!
!  ASSEMBLE THE VECTOR OF KNOWN VARIABLES, SCALING ANY PRESCRIBED
GRADIENTS.
        Each node in turn: DO I=1,NNP
        IF(IBC(I) == 1) DPSI(I)=STORE(I)
        IF(IBC(I) == 2) DPSI(I)=STORE(I)*MAXL
        IF(IBC(I) == 3) DPSI(I)=BETA(I)*MAXL
        END DO Each node in turn
!
        RETURN
        END SUBROUTINE BCS
```

For each segment over which potential is prescribed, the segment number and value of potential are read into ISEGBC and PSISEG, respectively. Taking each of these segments in turn, the nodes have their boundary condition type set to 1 (in array IBC), and the value of potential is transferred temporarily to array STORE. The prescribed potential and numbers of the nodes (elements) to which it is applied are then written out.

For each segment over which potential gradient is prescribed, the segment number and value of potential gradient are read into ISEGBC and DPSISEG, respectively. Taking each of these segments in turn, the nodes have their boundary condition type set to 2 (in array IBC), and the value of potential gradient is transferred temporarily to array STORE. The prescribed potential gradient and numbers of the nodes (elements) to which it is applied are then written out.

For each segment over which mixed boundary conditions prescribed, the segment number and values of the constants α and β (in Equation 2.32) are read into ISEGBC, ALPHASEG and BETASEG, respectively. Taking each of these segments in turn, the nodes have their boundary condition type set to 3 (in array IBC), and the values of boundary condition constants are transferred to arrays ALPHA and BETA. The constants and numbers of the nodes (elements) to which the mixed boundary condition is applied are then written out.

Finally, for each node in turn, the array DPSI is made to contain the known variable values: according to Equation 2.30 this is typically potential gradient (hence the use of array DPSI). In the more general form of Equation 2.31, however, it is either potential or potential gradient according to the boundary conditions. If potential is prescribed its value is transferred to DPSI from STORE. If potential gradient is prescribed, its value is similarly transferred from STORE and multiplied by MAXL, the maximum domain dimension. While mesh dimensions have already been scaled, gradient values have not. Potential is not affected by scaling because it does not involve dimensions of length. Potential gradients have the dimension of potential divided by length, so in order to scale must be multiplied by the length scale. If a mixed boundary condition is prescribed, the (unscaled) value of is transferred from STORE and multiplied by MAXL as the known variable (potential gradient) contribution, as described in connection with Equation 2.32.

3.1.6 Subprogram for Poisson type problems

In subprogram POISSON the particular integral contributions to potentials and potential gradients at the nodes are first set to zero. If the problem is of the Laplace type (indicated by a zero value of f_1 in Equation 2.1 stored in F1 in the program) control returns to the calling program. If the problem is of the Poisson type, then for every node the values of the particular integral potential and potential gradient defined by Equations 2.108 and 2.109, in co-ordinates (XX and YY) which have their origin at the point (XC, YC), are computed and stored in arrays PSIPI and DPSIPI. Note that the particular integral is defined in terms of scaled variables, so that the co-ordinates of the origin are first divided by MAXL, and f_1 is multiplied by the square of this length. From Equation 2.1, f_1 must have the dimensions of potential divided by length squared, because it is equated to second derivatives of potential.

```
      SUBROUTINE POISSON
!
! MODIFY BOUNDARY CONDITIONS IF GOVERNING EQUATION IS POISSON TYPE.
!
      USE SHAREDDATA2PC
!
! INITIALISE THE PARTICULAR INTEGRAL ARRAYS.
      Each node in turn: DO I=1,NNP
      PSIPI(I)=0.
      DPSIPI(I)=0.
      END DO Each node in turn
      IF(F1 == 0.) RETURN
!
!  FIND AND STORE THE POTENTIAL AND POTENTIAL GRADIENT AT EVERY NODE
!  DUE TO THE PARTICULAR INTEGRAL.
      XC=XC/MAXL
      YC=YC/MAXL
      F1=F1*MAXL**2
      Each node in turn: DO I=1,NNP
      XX=XNODE(I)-XC
      YY=YNODE(I)-YC
      PSIPI(I)=0.25*F1*(XX**2+YY**2)
      DPSIPI(I)=0.5*F1*(XX*UNX(I)+YY*UNY(I))
      END DO Each node in turn
!
!  SUBTRACT THE PARTICULAR INTEGRAL CONTRIBUTIONS FROM THE PRESCRIBED
!  BOUNDARY CONDITIONS.
      Each node in turn: DO I=1,NNP
      IF(IBC(I) == 1) DPSI(I)=DPSI(I)-PSIPI(I)
```

```
          IF(IBC(I) == 2) DPSI(I)=DPSI(I)-DPSIPI(I)
          IF(IBC(I) == 3) DPSI(I)=DPSI(I)-DPSIPI(I)+ALPHA(I)*PSIPI(I)*MAXL
          END DO Each node in turn
!
          RETURN
          END SUBROUTINE POISSON
```

Then for each node, the scaled particular integral contribution is subtracted from the scaled known variable. In other words, for a type 1 boundary condition (prescribed potential) the potential due to the particular integral (Equation 2.108) is subtracted from the scaled potential defined by the actual boundary condition, which is already stored in the array DPSI. For a type 2 boundary condition (prescribed potential gradient) the potential gradient due to the particular integral (Equation 2.109) is subtracted from the (scaled) potential gradient defined by the actual boundary condition, which is already stored in the array DPSI.

The type 3 (mixed) boundary condition is a little more complicated. From Equation 2.103, the required solution to the Poisson equation, which can be represented by the "total" potential ψ_T to avoid confusion, is the solution to the Laplace form of the problem (with $f_1 = 0$) plus the particular integral

$$\psi_T = \psi_L + \psi_{PI} \tag{3.17}$$

a relationship which is independent of scaling. Similarly, for potential gradient

$$\left(\frac{\partial\psi}{\partial n}\right)_T = \left(\frac{\partial\psi}{\partial n}\right)_L + \left(\frac{\partial\psi}{\partial n}\right)_{PI} \tag{3.18}$$

which applies to either all scaled or all unscaled gradients. The mixed boundary condition expressed by Equation 2.32 in terms of unscaled variables is

$$\left(\frac{\partial\psi}{\partial n}\right)_T = \alpha\psi_T + \beta \tag{3.19}$$

Applying scaling, and using * to indicate scaled quantities

$$\left(\frac{\partial\psi}{\partial n}\right)_T^* = L\left(\frac{\partial\psi}{\partial n}\right)_T = L(\alpha\psi_T + \beta) \tag{3.20}$$

where L is the maximum domain dimension (MAXL). Using Equations 3.17 and 3.18

$$\left(\frac{\partial\psi}{\partial n}\right)_L^* = \left(\frac{\partial\psi}{\partial n}\right)_T^* - \left(\frac{\partial\psi}{\partial n}\right)_{PI}^* = L[\alpha(\psi_L + \psi_{PI}) + \beta] - \left(\frac{\partial\psi}{\partial n}\right)_{PI}^*$$

$$\left(\frac{\partial\psi}{\partial n}\right)_L^* = L\alpha\psi_L + L\alpha\psi_{PI} + L\beta - \left(\frac{\partial\psi}{\partial n}\right)_{PI}^* \tag{3.21}$$

So in the program, which is solving the corresponding scaled Laplace problem, to the array DPSI (known variables) which already contains $L\beta$ (from subprogram BCS) must be added $L\alpha\psi_{PI}$ and subtracted the particular integral potential gradient (which is already scaled). The $L\alpha\psi_L$ contains the as yet unknown potential and has to be taken to the left hand side of the equations.

3.1.7 Subprogram to form the coefficient matrix

Subprogram FRMTRX forms the coefficient matrix and right hand side vector of Equations 2.31, and stores them as an extended matrix in array A. The extended matrix is simply $[A^*]$ with an extra column to contain vector $[b]$. Each node in turn is treated as point *P*, the source point for the fundamental solution. Then each element in turn is taken as the element over which integration is being carried out, and therefore containing the field point *Q*. Node counters I and J are used for P and the element node, respectively, so that I is the equation counter and numbers rows in $[A^*]$, while J numbers columns. If I and J are not the same number, then P is not in the element containing Q. Subprogram KERNEL is called to compute the integrals of the kernel functions to give coefficients A_{ij} and B_{ij} (AIJ and BIJ in program variables). On the other hand, if I and J are the same, then *P* and *Q* are in the same element and A_{ii} takes the value π according to Equation 2.45. Subprogram KERN2 is called to compute the integral of the second kernel function to find coefficient B_{ii}.

```
      SUBROUTINE  FRMTRX
!
!  SUBPROGRAM TO FORM THE COEFFICIENT MATRIX AND RIGHT HAND SIDE VECTOR,
!  MODIFIED TO SUIT THE BOUNDARY CONDITIONS.
!
      USE SHAREDDATA2PC
!
!  DEFINE THE NUMBER OF COLUMNS IN THE EXTENDED COEFFICIENT MATRIX A.
      JMAX=NNP+1
!
!  FORM THE COEFFICIENT MATRIX A, AND THE RIGHT HAND SIDE VECTOR B*DPSI.
      Take each node in turn as P: DO I=1,NNP
      BDPSI=0.
      Take each element in turn to contain Q: DO J=1,NNP
!
!  EVALUATE THE KERNELS WHEN P IS NOT IN THE ELEMENT CONTAINING Q.
      IF(I /= J) CALL  KERNEL(I,J,AIJ,BIJ)
!
!  EVALUATE THE KERNELS WHEN P IS IN THE ELEMENT CONTAINING Q.
      IF(I == J) THEN
        AIJ=PI
        CALL  KERN2(J,BIJ)
      END IF
```

```
!
!  IF POTENTIAL IS PRESCRIBED OVER THE ELEMENT CONTAINING Q,
!  INTERCHANGE THE A AND B COEFFICIENTS.
      IF(IBC(J) == 1) THEN
        TEMP=AIJ
        AIJ=-BIJ
        BIJ=-TEMP
      END IF
!
!  IF THE BOUNDARY CONDITION OVER THE ELEMENT CONTAINING Q IS MIXED,
!  MODIFY THE A COEFFICIENT.
      IF(IBC(J) == 3) AIJ=AIJ-BIJ*ALPHA(J)*MAXL
!
!  STORE THE A COEFFICIENT AND ACCUMULATE THE B*DPSI VECTOR COEFFICIENT.
      A(I,J)=AIJ
      BDPSI=BDPSI+BIJ*DPSI(J)
      END DO Take each element in turn to contain Q
!
!  STORE THE B*DPSI COEFFICIENT AS AN EXTENSION OF MATRIX A.
      A(I,JMAX)=BDPSI
      END DO Take each node in turn as P
!
      RETURN
      END SUBROUTINE FRMTRX
```

If the boundary condition over a particular element is of type 1 the potential there is prescribed, and therefore known. The $A_{ij}\,\psi_j$ term in the equations must be moved from the left hand side of the equations to the right hand side, while the $B_{ij}\left(\frac{\partial\psi}{\partial n}\right)_j$ term must be moved in the opposite direction. These changes are effected by interchanging the A_{ij} and B_{ij} coefficients, and the vector of known quantities stored in array DPSI already contains the prescribed value of potential from subprogram BCS. If the boundary condition over a particular element is of type 2 the potential gradient there is prescribed, and therefore known, and no changes are required: DPSI already contains the scaled prescribed value of potential gradient.

If the boundary condition over a particular element is of type 3 then there is a linear relationship between potential and potential gradient as in Equation 2.32. BDPSI already contains the scaled prescribed value of the constant β. Because the potential is unknown the scaled $B_{ij}\,\alpha\psi_j$ term (from $B_{ij}\,(\alpha\psi_j+\beta)$) in the equations must be moved from the right hand side to the left hand side, which is achieved by subtracting the scaled $B_{ij}\,\alpha$ from the corresponding coefficient in the $[A]$ matrix, contributing to its modified form $[A^*]$.

Finally for each equation, the product of B_{ij} and the relevant known quantity is accumulated in the variable BDSI (part of the $[B^*][y]=[b]$ operation required by Equations 2.31) and when complete is stored as the final column of array A.

3.1.8 Subprogram for computing integrals of the kernel functions when *P* is not in the element containing *Q*

Subprogram KERNEL implements the formulae derived in Section 2.4.1, first computing the distances from point to the ends of the element containing node Q. Because the same subprogram is later used when P is a point within the solution domain there is a test for whether by mistake P coincides with one of the ends, which would lead to an unacceptable division by zero. The integral of the first kernel function, A_{ij}, (Equation 2.36) is computed according to Equation 2.39 and stored in AIJ, and the integral of the second kernel, B_{ij}, according to Equation 2.41 and stored in BIJ.

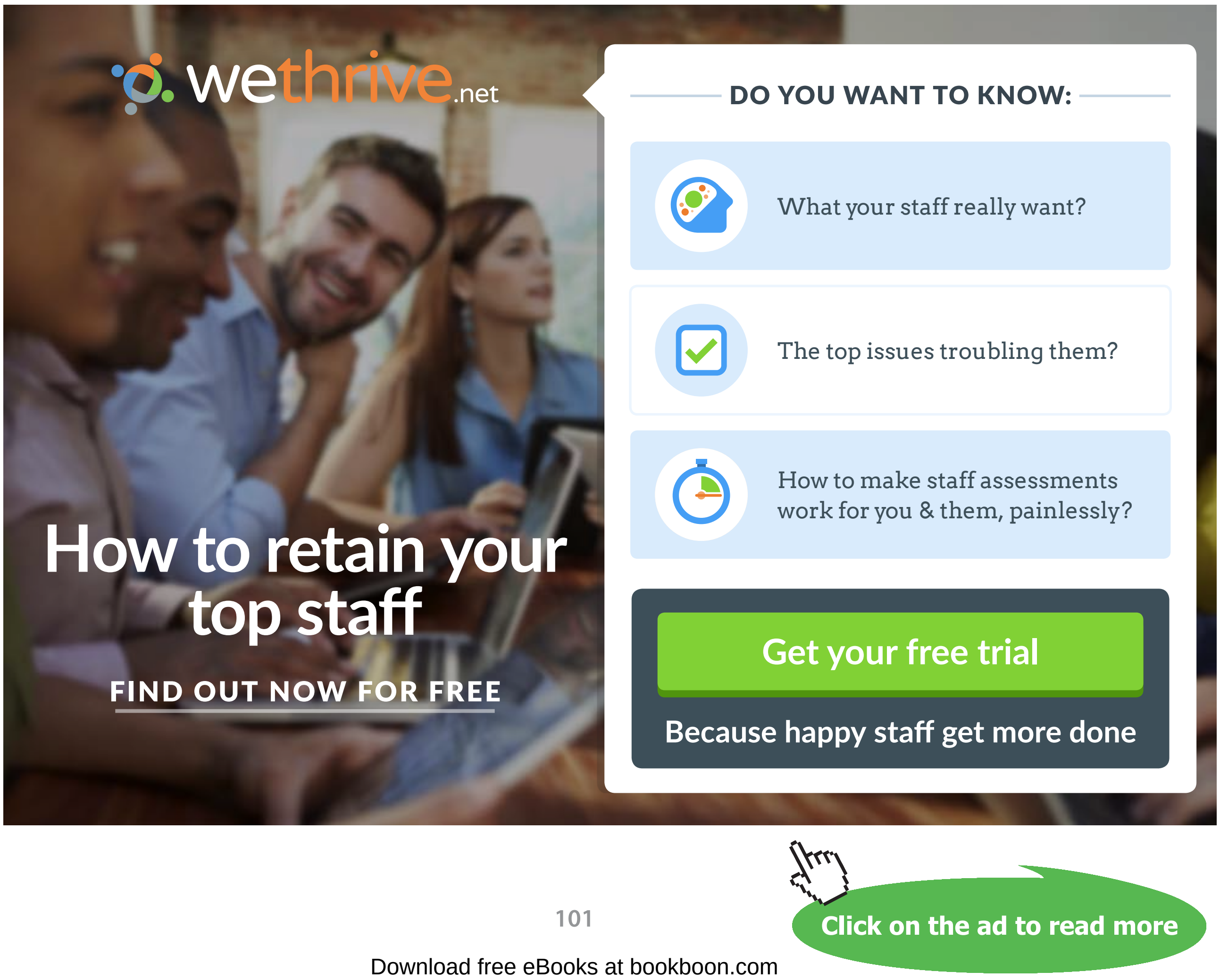
```
      SUBROUTINE  KERNEL(I,J,AIJ,BIJ)
!
! SUBPROGRAM TO EVALUATE THE INTEGRALS OF THE KERNEL FUNCTIONS
! WHEN P IS NOT IN THE ELEMENT CONTAINING Q.
!
      USE SHAREDDATA2PC
!
```

```
!  FIND THE COMPONENTS OF THE RADIAL DISTANCES OF P FROM THE ENDS
!  OF THE ELEMENT CONTAINING Q, AND THE DISTANCES THEMSELVES.
      IEP1=NEP1(J)
      IEP2=NEP2(J)
      R1X=XEEND(IEP1)-XNODE(I)
      R2X=XEEND(IEP2)-XNODE(I)
      R1Y=YEEND(IEP1)-YNODE(I)
      R2Y=YEEND(IEP2)-YNODE(I)
      R1=SQRT(R1X**2+R1Y**2)
      R2=SQRT(R2X**2+R2Y**2)
      IF(R1 < 1.E-6 .OR. R2 < 1.E-6) THEN
        WRITE(6,61) J
 61     FORMAT(/ "ERROR - INTERNAL POINT COINCIDENT WITH AN END",
     &            " POINT OF ELEMENT",I6)
      STOP
      END IF
!
!  FIND THE COMPONENTS OF THE UNIT VECTORS FROM P TO THE ENDS OF
!  THE ELEMENT CONTAINING Q.
      UR1X=R1X/R1
      UR2X=R2X/R2
      UR1Y=R1Y/R1
      UR2Y=R2Y/R2
!
!  FIND THE PERPENDICULAR DISTANCE OF P FROM THE ELEMENT CONTAINING Q.
      D=R1X*UNX(J)+R1Y*UNY(J)
!
!  EVALUATE THE INTEGRALS OF THE KERNELS.
      AIJ=ASIN(UR1Y*UR2X-UR1X*UR2Y)
      COSTH=UR1X*UR2X+UR1Y*UR2Y
      IF(COSTH < 0. .AND. AIJ > 0.) AIJ=PI-AIJ
      IF(COSTH < 0. .AND. AIJ <= 0.) AIJ=-PI-AIJ
      SINTH1=UNX(J)*UR1Y-UNY(J)*UR1X
      SINTH2=UNX(J)*UR2Y-UNY(J)*UR2X
      BLIM1=R1*SINTH1*(ALOG(1./R1)+1.)
      BLIM2=R2*SINTH2*(ALOG(1./R2)+1.)
      BIJ=BLIM2-BLIM1+D*AIJ
!
      RETURN
      END SUBROUTINE KERNEL
```

Using Equation 2.39 to find A_{ij} involves taking an inverse sine to obtain the required difference in angles, using function ASIN in the subprogram. But inverse sine is a multi-valued function, and ASIN only gives an angle in radians in the range $-\pi/2$ to $+\pi/2$ Bearing in mind that the magnitude of A_{ij} is the angle between the two lines from P to the ends of the element containing Q, this is normally satisfactory in that the angle is usually small, and less than $\pi/2$. There are, however, circumstances where this is no longer true and the angle can approach π. The first is in a narrow domain where a node of the mesh is sufficiently close to an element on another part of the boundary for the angle to be greater than $\pi/2$. The second is where point P is not a node but an internal point which is again very close to the boundary. For example, if the angle associated with a particular combination of point and element is $3\pi/4$, the inverse sine function would give an incorrect $\pi/4$. The way to test for this situation is via the cosine of the angle, which would be negative for an angle greater than $\pi/2$ in magnitude. The cosine can be found from the scalar product of vectors $\hat{\boldsymbol{r}}_1$ and $\hat{\boldsymbol{r}}_2$ (similar to the vector product in Equation 2.37)

$$\hat{\boldsymbol{r}}_1 . \hat{\boldsymbol{r}}_2 = (\hat{r}_{1x}\boldsymbol{i} + \hat{r}_{1y}\boldsymbol{j}) . (\hat{r}_{2x}\boldsymbol{i} + \hat{r}_{2y}\boldsymbol{j}) = 1 \times 1 \times \cos(\theta_2 - \theta_1) \qquad (3.22)$$

and $\cos(\theta_2 - \theta_1) = \cos(\theta_1 - \theta_2) = \hat{r}_{1x}\hat{r}_{2x} + \hat{r}_{1y}\hat{r}_{2y}$ (3.23)


wethrive.net
How to retain your top staff
FIND OUT NOW FOR FREE
DO YOU WANT TO KNOW:
What your staff really want?
The top issues troubling them?
How to make staff assessments work for you & them, painlessly?
Get your free trial
Because happy staff get more done
Click on the ad to read more

If this expression is negative then the value of A_{ij} obtained from the inverse sine function must be modified to obtain the correct angle. If the angle A_{ij} is positive then it must be subtracted from π: for example, $\pi/4$ becomes $3\pi/4$. On the other hand, if it is negative then it must be subtracted from $-\pi$: for example, $-\pi/4$ becomes $-\pi-(-\pi/4)=-3\pi/4$. These changes are implemented in subprogram KERNEL.

3.1.9 Subprogram for computing the integral of the second kernel function when *P* is in the element containing *Q*

Subprogram KERN2 implements the formula derived in Section 2.4.2. The integral of the second kernel function, B_{ij}, is computed according to Equation 2.48 and stored in BIJ. Note that the element lengths stored in array ELENGTH have to be scaled before use.

```
      SUBROUTINE  KERN2(J,BIJ)
!
!  SUBPROGRAM TO EVALUATE THE INTEGRAL OF THE SECOND KERNEL
!  FUNCTION WHEN P IS IN THE ELEMENT CONTAINING Q.
!
      USE SHAREDDATA2PC
!
!  EVALUATE THE SEMI LENGTH OF THE ELEMENT CONTAINING Q.
      R1=0.5*ELENGTH(J)/MAXL
!
!  HENCE THE INTEGRAL OF THE KERNEL.
      BIJ=2.*R1*(ALOG(1./R1)+1.)
!
      RETURN
      END SUBROUTINE KERN2
```

3.1.10 Results output subprogram

Subprogram OUTPUT organises and writes out the primary results of the computation, but not including secondary results such as values of potentials at internal points. Firstly, potentials and potential gradients are put back into their proper arrays, and the contributions of the particular integral added to get from the solution to the Laplace equation (with modified boundary conditions) which has been solved back to the "total" solution to the original problem. If the original problem was of the Laplace type, then the contributions from the particular integral potential and gradient are both zero.

```
      SUBROUTINE  OUTPUT
!
!  SUBPROGRAM TO WRITE OUT THE NODAL POINT VALUES OF POTENTIAL AND
!  POTENTIAL GRADIENT, AND COMPUTE POTENTIAL FLOWS ACROSS THE
!  BOUNDARY SEGMENTS.
!
      USE SHAREDDATA2PC
!
!  ARRANGE FOR PSI AND DPSI TO CONTAIN THE POTENTIALS AND POTENTIAL
!  GRADIENTS, RESPECTIVELY.
      Each node in turn: DO I=1,NNP
      IF(IBC(I) == 1) THEN
        TEMP=PSI(I)
        PSI(I)=DPSI(I)
        DPSI(I)=TEMP
      END IF
      IF(IBC(I) == 3) THEN
        PSITOT=PSI(I)+PSIPI(I)
        DPSI(I)=(ALPHA(I)*PSITOT+BETA(I))*MAXL-DPSIPI(I)
        END IF
      END DO Each node in turn
!
!  ADD CONTRIBUTIONS OF THE PARTICULAR INTEGRAL.
      WRITE(6,61)
 61   FORMAT(/ "NODAL POINT POTENTIALS AND POTENTIAL GRADIENTS" //
     &          "      I      PSI             DPSI      " )
      Each node in turn: DO I=1,NNP
      PSIT(I)=PSI(I)
      DPSIT(I)=DPSI(I)
      IF(F1 /= 0.) THEN
        PSIT(I)=PSIT(I)+PSIPI(I)
        DPSIT(I)=DPSIT(I)+DPSIPI(I)
      END IF
!
!  REMOVE THE SCALE FACTOR FROM THE GRADIENT VALUES.
      DPSIT(I)=DPSIT(I)/MAXL
!
!  OUTPUT THE NODAL VALUES OF POTENTIAL AND POTENTIAL GRADIENT.
      WRITE(6,62) I,PSIT(I),DPSIT(I)
 62   FORMAT(1X,I6,2E15.6)
      END DO Each node in turn
!
```

```
 !  COMPUTE THE POTENTIAL FLOWS ACROSS THE BOUNDARY SEGMENTS.
       FLOWIN=0.
       FLOWOUT=0.
       Each segment in turn: DO ISEG=1,NSEGTOT
       FLOWSEG(ISEG)=0.
       END DO Each segment in turn
       Each element in turn: DO M=1,NEL
       ISEG=ISEGELEM(M)
       FLOWELEM=DPSIT(M)*ELENGTH(M)
       FLOWSEG(ISEG)=FLOWSEG(ISEG)+FLOWELEM
       IF(FLOWELEM > 0.) FLOWIN=FLOWIN+FLOWELEM
       IF(FLOWELEM < 0.) FLOWOUT=FLOWOUT-FLOWELEM
       END DO Each element in turn
       WRITE(6,63) (ISEG,FLOWSEG(ISEG),ISEG=1,NSEGTOT)
  63   FORMAT(/ "POTENTIAL FLOWS INTO DOMAIN ACROSS BOUNDARY SEGMENTS"
      &      // "SEGMENT      FLOW" / (I5,E16.6))
       WRITE(6,64) FLOWIN,FLOWOUT
  64   FORMAT(/ "TOTAL POTENTIAL FLOW INTO DOMAIN =",E15.6 //
      &         "TOTAL POTENTIAL FLOW OUT OF DOMAIN =",E15.6)
 !
       RETURN
      END SUBROUTINE OUTPUT
```

If the boundary condition at a particular point is of the first type (prescribed potential) the solution array PSI contains the computed potential gradient, and the relevant components of arrays PSI and DPSI must be interchanged. If the boundary condition is of the third, mixed, type then potential was the unknown and is now correctly stored in array PSI. On the other hand, the potential gradient solution (for the Laplace problem) has to be found using Equation 3.21 and stored in DPSI.

Arrays PSI and DPSI now contain all the nodal point values of potential and potential gradient, either from the boundary conditions or from the boundary element solution. But they are for the Laplace equation with the particular integral subtracted. The latter contributions, stored in PSIPI and DPSIPI are now added to PSI and DPSI to give the "total" values solving the original problem and stored in PSIT and DPSIT. Both sets are scaled values, and the gradients have to be unscaled by dividing by the length scale, MAXL. For each node, the node number, potential and potential gradient are written out.

In many problems of practical engineering interest, the “flows” of potential across different parts of the boundary of a problem are of interest. For example, the flow rate of fluid across the boundary of a porous medium flow, or the flow rate of heat across the boundary of a heat conduction problem may be required. Such flow rates are locally proportional to the potential gradient normal to the boundary, and their totals over parts or all of the boundary can be found by integrating the potential gradient with respect to distance along the boundary. In the case of constant boundary element results this is very simply done by finding the flow contribution of each element (FLOWELEM) as the product of potential gradient and element length, and summing these contributions over the required part of the boundary. In the subprogram, the total flows over each boundary segment (FLOWSEG) are computed, together with the total flows into (FLOWIN) and out of (FLOWOUT) the domain which are found and finally written out.

3.1.11 Subprogram for computing potentials at internal points

Subprogram INTERNAL implements Equation 2.26 to find values of potential at points within the solution domain. Internal point *p* can be chosen freely, but the result may not be very accurate if the point is very close to the boundary. As already noted in subprogram KERNEL, the calculation procedure only breaks down completely if *p* happens to be chosen to coincide with an end point connecting two elements.

```
      SUBROUTINE INTERNAL
!
!  SUBPROGRAM TO COMPUTE AND OUTPUT THE POTENTIALS AT SELECTED
!  INTERNAL POINTS.
!
      USE SHAREDDATA2PC
!
!  INPUT NUMBER OF INTERNAL POINTS.
      READ(5,*) NINT
      IF(NINT == 0) RETURN
      WRITE(6,61)
 61   FORMAT(/ "POTENTIALS AT INTERNAL POINTS" //
     &        5X,"PSI                X           Y")
!
!  INPUT THE INTERNAL POINT COORDINATES.
      Each internal point in turn: DO IINT=1,NINT
      READ(5,*) XINT,YINT
      XNODE(1)=XINT/MAXL
      YNODE(1)=YINT/MAXL
!
!  INTEGRATE ROUND THE BOUNDARY.
      SUM=0.
      Each node in turn: DO J=1,NNP
      CALL KERNEL(1,J,AIJ,BIJ)
      SUM=SUM-AIJ*PSI(J)+BIJ*DPSI(J)
      END DO Each node in turn
      PSIIP=SUM*0.5/PI
      PSIIPT=PSIIP
!
!  ADD THE PARTICULAR INTEGRAL FOR A POISSON TYPE PROBLEM.
      IF(F1 /= 0.) THEN
        XX=XNODE(1)-XC
        YY=YNODE(1)-YC
        PSIIPT=PSIIPT+0.25*F1*(XX**2+YY**2)
      END IF
!
!  OUTPUT THE POTENTIAL AT THE INTERNAL POINT.
      WRITE(6,62) PSIIPT,XINT,YINT
 62   FORMAT(E14.6,5X,2E12.4)
      END DO Each internal point in turn
!
      RETURN
      END SUBROUTINE INTERNAL
```

In the subprogram the number of internal points is first read into variable NINT. Then for each point in turn its global co-ordinates are input to XINT and YINT. These are first scaled and stored as though they defined the first node of the mesh. This is to facilitate the use of subprogram KERNEL in the subsequent integration process, the subprogram being designed to treat the source point as a node. The integration over the boundary required by Equation 2.26 is performed by summing over the boundary elements, using KERNEL to return the integrals of the kernel functions in variables AIJ and BIJ. These are no longer identifiable as coefficients of matrices $[A]$ and $[B]$, because p is not now a node of the mesh. The resulting value of potential at the internal point is stored in variable PSIIP. In the case of a Poisson problem the particular integral at the internal point is found according to Equation 2.108, noting that no scaling is required (the previous scaling of F1 and the co-ordinates cancelling each other out), and added to give the required result. Finally, the potential and the (unscaled) co-ordinates of the internal point are written out.

3.1.12 Subprogram for solving the linear equations

Both the Gaussian elimination algorithm and subprogram ELIMIN for solving the linear equations are described in Appendix B. The matrix $[A^*]$ extended to include the right hand side column vector $[b]$ (in Equation 2.31) is entered in array A, and the solutions returned in array X (PSI in the calling main program).

3.2 Some Test Problems for BEM2PC

Before attempting to solve real problems for which the solutions are not known, it is essential to test any computer program as extensively as possible on problems for which exact analytical solutions are available for comparison. If necessary, test problems have to be simplified to the point where analytical solutions can be obtained. Such problems serve not only to verify that the program is working correctly, but also to examine the level of discretisation (the fineness of the boundary element mesh in this case) necessary to obtain accurate results.

3.2.1 A simple one-dimensional potential problem

The first test is effectively a one-dimensional one, although the domain is two-dimensional. Figure 3.2 shows a rectangular region 4 units wide and 2 deep, subject to zero derivative boundary conditions on the top and bottom edges, a potential value of 5 on the left hand edge and a value of 1 at the right hand edge. This could, for example, be a heat conduction problem in a rectangular slab of material, with thermal insulation (no conduction) at the top and bottom surfaces and temperature difference between the opposite end faces.

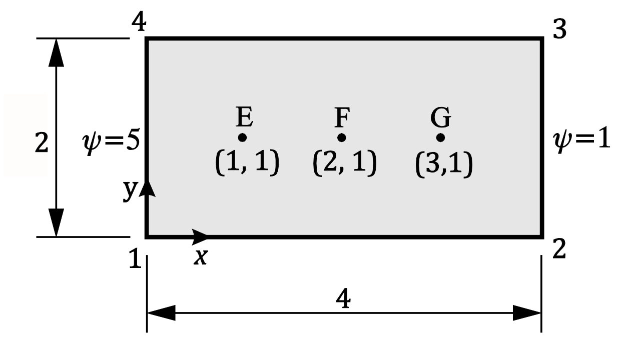


Figure 3.2 One-dimensional test problem

Due to the two zero derivative boundary conditions, plus the fact that the prescribed potentials on the end faces are uniform, the problem is one-dimensional, with no variations of potential in the vertical direction. It is convenient to choose global co-ordinates x and y as shown, with an origin at the bottom left hand corner of the domain. The Poisson equation governing the potential distribution simplifies to

$$\frac{d^2\psi}{dx^2} = f_1 \tag{3.24}$$

which can be integrated first to find the potential gradient

$$\frac{d\psi}{dx} = f_1 x + C_1 \tag{3.25}$$

and secondly to find the potential distribution

$$\psi = \frac{f_1 x^2}{2} + C_1 x + C_2 \tag{3.26}$$

where C_1 and C_2 are constants of integration.

The Laplace problem

Consider first the Laplace problem with $f_1 = 0$, and

$$\psi = C_1 x + C_2 \tag{3.27}$$

Given the prescribed potential boundary conditions $\psi = 5$ at $x = 0$ and $\psi = 1$ at $x = 4$, then $C_2 = 5$ and $C_1 = -1$. The exact analytical solution to the problem is

$$\psi = -x + 5\,, \qquad \frac{\mathrm{d}\psi}{\mathrm{d}x} = -1 \tag{3.28}$$

The potential "flows" into the left hand edge and out of the right hand edge of the domain are both of magnitude 2 (gradient times edge length).

To obtain a constant boundary element solution to the problem it is necessary to first divide the boundary into segments. Four segments along the four edges of the domain are appropriate, because they are straight lines between corners and the boundary condition over each one is uniform. The ends of the segments are shown numbered 1 to 4 in Figure 3.2. The choice of starting point is arbitrary, but then the direction of numbering must keep the domain to the left. The data file DATA to solve the problem using 3 elements (and nodes) per side of the domain is as follows

```
ONE-DIMENSIONAL TEST PROBLEM
0.                               (Value of f1)
1                                (Number of boundaries)
4                                (Number of segments on the boundary)
0. 0. 4. 0. 4. 2. 0. 2.          (Co-ordinates of the segment ends)
0. 3 1.                          (Radii of curvature,
0. 3 1.                          number of elements
0. 3 1.                          and element length ratio
0. 3 1.                          for each of the 4 segments)
2 0 0                            (Two prescribed boundary potentials)
2 1.                             (Potential = 1 on segment 2)
4 5.                             (Potential = 5 on segment 4)
3                                (Number of internal points)
```

```
1. 1.                                        (Co-ordinates of first internal point)
2. 1.                                        (Co-ordinates of second internal point)
3. 1.                                        (Co-ordinates of third internal point)
```

The annotations on the right have been added for explanation, and are not part of the data file. The potential values at three internal points are called for, points E, F and G in Figure 3.2. Note that the prescribed zero values of potential gradient on the top and bottom edges do not have to be entered explicitly.

The mesh data output file MESHRES is as follows

```
       GEOMETRIC DATA FOR THE MESH

            NUMBER OF ELEMENTS = 12

            NUMBER OF NODAL POINTS = 12

            NUMBER OF ELEMENT END POINTS = 13

       COORDINATES OF ELEMENT END POINTS

      I     X           Y           I     X           Y
 1  0.0000E+00  0.0000E+00     2  0.1333E+01  0.0000E+00
 3  0.2667E+01  0.0000E+00     4  0.4000E+01  0.0000E+00
 5  0.4000E+01  0.6667E+00     6  0.4000E+01  0.1333E+01
 7  0.4000E+01  0.2000E+01     8  0.2667E+01  0.2000E+01
 9  0.1333E+01  0.2000E+01    10  0.0000E+00  0.2000E+01
 11  0.0000E+00  0.1333E+01    12  0.0000E+00  0.6667E+00
 13  0.0000E+00  0.0000E+00

 ELEMENT END POINT NUMBERS, NODAL COORDINATES AND UNIT NORMAL COMPONENTS

      M   NEP1  NEP2   X(NODE)      Y(NODE)      UNX         UNY
      1     1     2  0.6667E+00  0.0000E+00  0.0000E+00 -0.1000E+01
      2     2     3  0.2000E+01  0.0000E+00  0.0000E+00 -0.1000E+01
      3     3     4  0.3333E+01  0.0000E+00  0.0000E+00 -0.1000E+01
      4     4     5  0.4000E+01  0.3333E+00  0.1000E+01  0.0000E+00
      5     5     6  0.4000E+01  0.1000E+01  0.1000E+01  0.0000E+00
      6     6     7  0.4000E+01  0.1667E+01  0.1000E+01  0.0000E+00
      7     7     8  0.3333E+01  0.2000E+01  0.0000E+00  0.1000E+01
      8     8     9  0.2000E+01  0.2000E+01  0.0000E+00  0.1000E+01
      9     9    10  0.6667E+00  0.2000E+01  0.0000E+00  0.1000E+01
     10    10    11  0.0000E+00  0.1667E+01 -0.1000E+01  0.0000E+00
     11    11    12  0.0000E+00  0.1000E+01 -0.1000E+01  0.0000E+00
     12    12    13  0.0000E+00  0.3333E+00 -0.1000E+01  0.0000E+00
```

With three elements per segment 12 elements and nodes are created. Element ends are numbered as for the segments, anticlockwise from the origin and there are 13 of them: end 13 is coincident with end 1. The elements and nodes are numbered from 1 to 12 in the same way. Element 5, for example, is at the middle of the right hand edge of the domain, has ends numbered 5 and 6, and its node is at (4, 1), the centre point of the edge.

The RESULTS data file output by the program is

```
CONSTANT BOUNDARY ELEMENT SOLUTION FOR TWO DIMENSIONAL POTENTIAL
PROBLEM

ONE-DIMENSIONAL TEST PROBLEM

LAPLACE EQUATION

PRESCRIBED POTENTIAL BOUNDARY CONDITIONS

POTENTIAL = 0.1000E+01 AT NODES 4 TO 6

POTENTIAL = 0.5000E+01 AT NODES 10 TO 12

NODAL POINT POTENTIALS AND POTENTIAL GRADIENTS

I          PSI          DPSI
1     0.438490E+01      0.000000E+00
2     0.300000E+01      0.000000E+00
3     0.161510E+01      0.000000E+00
4     0.100000E+01      -0.116979E+01
5     0.100000E+01      -0.912910E+00
6     0.100000E+01      -0.116979E+01
7     0.161510E+01      0.000000E+00
8     0.300000E+01      0.000000E+00
9     0.438490E+01      0.000000E+00
10    0.500000E+01      0.116979E+01
11    0.500000E+01      0.912910E+00
12    0.500000E+01      0.116979E+01
```

```
POTENTIAL FLOWS INTO DOMAIN ACROSS BOUNDARY SEGMENTS

SEGMENT              FLOW
   1              0.000000E+00
   2             -0.216833E+01
   3              0.000000E+00
   4              0.216833E+01

TOTAL POTENTIAL FLOW INTO DOMAIN = 0.216833E+01

TOTAL POTENTIAL FLOW OUT OF DOMAIN = 0.216833E+01

POTENTIALS AT INTERNAL POINTS

     PSI              X                  Y
0.402394E+01     0.1000E+01  0.1000E+01
0.300000E+01     0.2000E+01  0.1000E+01
0.197607E+01     0.3000E+01  0.1000E+01
```

The prescribed potential boundary conditions are a value of 1 over nodes 4, 5 and 6 on the right end edge (segment 2), and a value of 5 over nodes 10, 11 and 12 on the left hand edge (segment 4). Segments 1 and 3 defining the top and bottom edges had no boundary condition prescribed by the DATA file, so zero normal derivative conditions are assumed.

The computed results show that at nodes 2 and 8 (at $x = 2$ on the bottom and top edges, respectively) the exact potential value of 3 (Equations 3.28) is computed. At nodes 1 and 9 $(x = 0.6667)$ it is 4.3849, compared with the exact 4.3333, and at nodes 3 and 7 $(x = 3.3333)$ it is 1.6151, compared with the exact 1.6667. At node 5 at the centre of the right hand edge the computed potential gradient is -0.9129 compared with the exact -1 (Equations 3.28), and similarly at node 11 at the left hand edge it is +0.9129 compared with +1 (at this edge the normal direction is in the negative x direction). While at node 5 the computed gradient is 8.7% too low, at the adjacent nodes 4 and 6 it is 17% too high.

The computed potential "flows" into and out of the domain across the boundary segments are both 2.168, compared with the exact figure of 2 (8.4% difference). In any Laplace problem with no source or sink for potential within the domain it is to be expected that the net flow of potential is zero. The computed values of potential at points E, F and G (Figure 3.2) are 4.024, 3, and 1.976, compared with exact values of 4, 3 and 2.

It should be noted that the computed results are exactly symmetrical about the horizontal centre line of the domain, which is to be expected because the problem is similarly symmetrical. Also, the exact values of potential are obtained on the vertical centre line of the domain. The accuracies of other computed results are relatively poor, but at only three constant boundary elements per side the mesh is coarse. The next step is therefore progressively to refine the mesh and monitor the improvement in accuracy. For this latter purpose the potential gradient at the centre of the right hand edge, the potential at the internal point E, and the flows in and out of the domain (always equal) are examined. The last of these is useful in that it effectively averages the potential gradient over the domain edge. To refine the mesh, the only changes that have to be made in file DATA are the numbers of nodes (3 initially) on each of the four boundary segments. These numbers are progressively increased by about factors of 2 each time, but keeping an odd number so that a node coincides with the centre of the right hand edge. Table 3.1 shows the results obtained.

Nodes per edge	Gradient at (4,1)	Potential at (1, 1)	Flow in/out
Exact:	-1	4	2
3	-0.9129	4.0239	2.1683
7	-0.9783	4.0058	2.0424
15	-0.9932	4.0018	2.0121
31	-0.9978	4.0006	2.0037
61	-0.9993	4.0002	2.0012
121	-0.9997	4.0001	2.0004
241	-0.9999	4.0000	2.0001

Table 3.1 Effects of refining the mesh (Laplace problem)

Accuracies of better than 1% for the chosen values are achieved by the time the mesh refinement reaches 15 nodes per edge, which is the sort of accuracy which is often adequate in practical engineering problems. Further refinement leads to very high accuracy, with results correct to four significant figures at 241 nodes per edge, which is close to the maximum permitted by the program array dimensions. With very refined meshes, and certainly at 241 nodes per edge, the results computed for the boundary points, in this case notably the gradient values at the right hand boundary, are beginning to display some random fluctuations from point to point in the sixth, and occasionally fifth, decimal places. This is characteristic of roundoff errors resulting from representing numbers with only a finite number of significant figures and then performing a large amount of arithmetic on them. Roundoff errors can accumulate in the Gaussian elimination process used to solve very large sets of linear equations, as discussed in Appendix B, even though subprogram ELIMIN is designed to keep these errors to a minimum. Computed results at internal points and for the flows in and out of the domain are the results of integrations of boundary values over all or part of the boundary, which tend to smooth out the fluctuations due to roundoff, and these results converge more smoothly with mesh refinement. If roundoff were to become a serious problem, double precision arithmetic (doubling the number of bytes used to store each number) could be used, at least for the Gaussian elimination.

The Poisson problem

The same problem can be turned into one of the Poisson type by introducing a non-zero value of function f_1. A value can be arbitrarily chosen as 1. If this were a thermal conduction problem a positive value of f_1 would imply a uniform heat absorption throughout the domain (in Equation 1.10 heat generation is positive on the left hand side of the equation, resulting in a negative f_1). Applying the prescribed potential boundary conditions $\psi = 5$ at $x = 0$ and $\psi = 1$ at $x = 4$, to Equation 3.26 gives $C_2 = 5$ and $C_1 = -3$. The exact analytical solution to the problem is

$$\psi = \frac{x^2}{2} - 3x + 5, \qquad \frac{d\psi}{dx} = x - 3 \tag{3.29}$$

The potential at point E $(x = 1)$ is now 2.5, and the potential gradients at the left and right hand edges $(x = 0$ and 4) are and , respectively. The potential "flow" into the left hand and right hand edges of the domain are 6 and 2 (8 in total, the magnitude of sink f_1 multiplied by the area $4 \times 2 = 8$ over which it is uniformly distributed).

Although the analytical solution remains one-dimensional, the two-dimensional form of the particular integral means that the Laplace problem solved by the program, with boundary conditions which are now two-dimensional, is certainly two-dimensional. The only change required to file DATA is to replace the zero value (for f_1) in the second line by the value 1., followed immediately by a new line with the co-ordinates of the origin for the particular integral function. The centre of the domain at $x = 2., y = 1.$ is convenient.

Nodes per edge	Gradient at (4, 1)	Potential at (1, 1)	Flow in at $x = 0$	Flow in at $x = 4$
Exact:	1	2.5	6	2
3	0.9917	2.5827	6.2542	1.9175
7	0.9931	2.5221	6.0664	1.9816
15	0.9979	2.5065	6.0194	1.9951
31	0.9994	2.5020	6.0059	1.9986
61	0.9998	2.5006	6.0019	1.9995
121	0.9999	2.5002	6.0006	1.9999
241	1.0000	2.5001	6.0002	2.0000

Table 3.2 Effects of refining the mesh (Poisson problem)

Table 3.2 shows the results of mesh refinement in a form comparable to Table 3.1. Very similar trends can be observed, including the effects of roundoff on individual boundary values. The change from a Laplace to a Poisson type problem, with the introduction of a two-dimensional particular integral, does not adversely affect accuracy.

Internal points close to the boundary

In Figure 3.2 the chosen internal points E, F and G are not close to any part of the boundary. With most boundary element methods, taking an internal point close to the boundary leads to loss of accuracy, due to the singular nature of the fundamental solutions and the use of approximate numerical integration. Table 3.3 shows the effect of taking internal point E from its initial position at (1, 1) progressively closer to the boundary at (1, 0) (not varying its x co-ordinate, $x = 1$). The meshes used are 61 and 241 nodes per edge.

y	61 nodes per edge	241 nodes per edge
1	2.5006	2.5001
10^{-1}	2.5009	2.5001
10^{-2}	2.4951	2.5000
10^{-3}	2.4893	2.4977
10^{-4}	2.4887	2.4970
10^{-5}	2.4886	2.4970
10^{-6}	2.4886	2.4970
0	2.4886	2.4970

Table 3.3 Potential at internal points at (1, *y*)

YOU THINK.
YOU CAN WORK
AT RMB
RAND MERCHANT BANK
A division of FirstRand Bank Limited
Traditional values. Innovative ideas.

Although there is some loss of accuracy compared to the exact value of 2.5, the effect of approaching the boundary is small. For both meshes, the change from computed values slightly above the exact to values somewhat below it happens as the internal point comes within about one element length of the boundary. The last line in the Table is for a point actually on the boundary. The reason for being able to maintain good accuracy is that with constant elements all the integrations around the boundary are carried out analytically. Note that it would not have been possible to obtain the potential at an element end point in this way, and the program would have given an error message. Interestingly, it is possible to treat a node as an internal point. For example, using the 61 node per edge mesh, the potential computed as part of the main solution at the centre point of the lower edge, at (2, 0) is 1.00053 (exact 1), and precisely the same value is obtained when (2, 0) is treated as an internal point.

Domain aspect ratio

Rather like taking an internal point close to a boundary, making the domain very narrow and bringing opposite sides of the boundary close together can cause loss of accuracy. In the present problem, the width of the domain is twice its height, giving an aspect ratio of 2. Table 3.4 shows the effects of increasing this aspect ratio, by progressively reducing the depth of the domain, using a mesh of 61 nodes per edge.

Aspect ratio	Potential at node 1	Potential at node 31	Gradient at node 92
Exact	4.9022	1	1
2	4.9082	1.0005	0.9998
4	4.9092	1.0013	0.9995
8	4.9099	1.0028	0.9989
16	4.9109	1.0062	0.9977
32	4.9129	1.0130	0.9960
64	4.9176	1.0262	0.9935
128	4.9264	1.0462	0.9881
256	4.9376	1.0672	0.9829

Table 3.4 Effects of varying domain aspect ratio

The variables selected for scrutiny are the potential at node 31, at the centre of the lower edge, the potential gradient at node 92, at the centre of the right hand edge, and the potential at node 1 on the bottom edge, at $x = 2/61$. Node 1 is chosen deliberately close to a corner of the domain. While there is some deterioration of accuracy with increasing aspect ratio, the effect is not disastrous, even when the top and bottom edges approach to within the length of one element on these edges (which occurs at an aspect ratio of about 60).

Prescribed potential gradient boundary conditions

So far, only prescribed potential and zero potential gradient boundary conditions have been used. Staying with the present Poisson problem, the condition on the left hand edge can be changed from a prescribed potential of magnitude 5 to a prescribed gradient of 3: the problem is unchanged. If this is done using a mesh with 61 nodes per edge, the following are obtained

Potential at (2, 0) = 0.9987 (exact 1)

Potential gradient at (4, 1) = 1.0007 (exact 1, previously 0.9998)

Potential at (0, 1) = 4.9977 (exact 5, previously 5 prescribed)

The results are, as is to be expected, not identical to those obtained when potential rather than the gradient is prescribed, but the accuracy is not impaired.

Mixed boundary conditions

Again staying with the same problem, on the left hand edge the exact solution gives the potential as 5 and gradient as 3, so in place of either the prescribed potential or prescribed gradient, a possible mixed boundary condition on the edge is

$$\frac{\mathrm{d}\psi}{\mathrm{d}n} = \psi - 2$$

Prescribing values of $\alpha = 1$ and $\beta = -2$ and again using a mesh of 61 nodes per edge, the following are obtained

Potential at (2, 0) = 1.0011 (exact 1)

Potential gradient at (4, 1) = 0.9953 (exact 1)

Potential at (0, 1) = 5.0032 (exact 5)

Potential gradient at (0, 1) = 3.0032 (exact 3)

Again there is no loss of accuracy.

3.2.2 A thick-walled cylinder test problem

The second test is designed to demonstrate the ability of the program to solve problems with curved boundaries, problems with more than one boundary, and the use of elements of varying size along a boundary segment. Figure 3.3 shows the cross-section of a thick-walled circular cylinder with internal and external radii of r_1 and r_2. The boundary conditions are prescribed potentials of ψ_1 and ψ_2 at the inner and outer surfaces, respectively. A practical engineering application would be to heat conduction through the walls of a cylinder with different temperatures at the two curved surfaces.

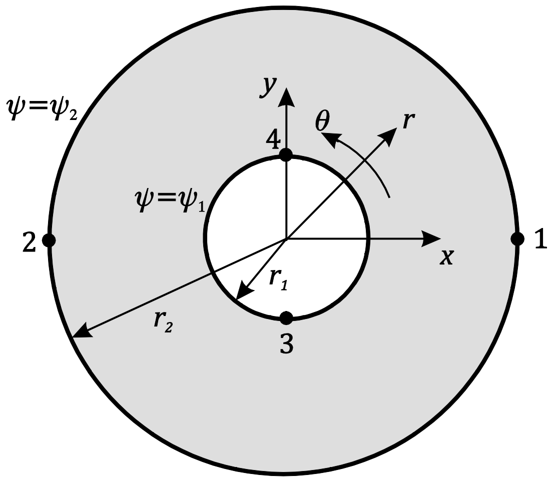


Figure 3.3 Thick-walled cylinder test problem

The problem is simple enough for an analytical solution to be obtainable. In plane polar co-ordinates (r, θ) with r in the radial direction and θ in the circumferential direction as in Figure 3.3, Poisson's equation, Equation 2.1, becomes

$$\frac{\partial^2 \psi}{\partial r^2} + \frac{1}{r}\frac{\partial \psi}{\partial r} + \frac{1}{r^2}\frac{\partial^2 \psi}{\partial \theta^2} = \nabla^2 \psi = f_1 \tag{3.30}$$

The problem is axisymmetric, with no variations in the circumferential direction. The second derivative with respect to θ is therefore zero, and Equation 3.30 can be written

$$\frac{1}{r}\frac{\mathrm{d}}{\mathrm{d}r}\left[r\frac{\mathrm{d}\psi}{\mathrm{d}r}\right] = f_1 \tag{3.31}$$

Hence $\frac{\mathrm{d}}{\mathrm{d}r}\left[r\frac{\mathrm{d}\psi}{\mathrm{d}r}\right] = f_1 r$

$$r\frac{\mathrm{d}\psi}{\mathrm{d}r} = \frac{f_1 r^2}{2} + C_1$$

$$\frac{\mathrm{d}}{\mathrm{d}r}\left[r\frac{\mathrm{d}\psi}{\mathrm{d}r}\right] = f_1 r \tag{3.32}$$

$$\psi = \frac{f_1 r^2}{4} + C_1 \ln r + C_2 \tag{3.33}$$

where C_1 and C_2 are constants of integration. Applying the boundary conditions $\psi = \psi_1$ at $r = r_1$ and $\psi = \psi_2$ at $r = r_2$ results in the following expressions for potential and potential gradient

$$\psi = \psi_1 + \frac{f_1}{4}(r^2 - r_1^2) + \left[(\psi_2 - \psi_1) - \frac{f_1}{4}(r_2^2 - r_1^2)\right]\frac{\ln(\frac{r}{r_1})}{\ln(\frac{r_2}{r_1})} \tag{3.34}$$

$$\frac{\mathrm{d}\psi}{\mathrm{d}r} = \frac{f_1 r}{2} + \left[(\psi_2 - \psi_1) - \frac{f_1}{4}(r_2^2 - r_1^2)\right]\frac{1}{r\ln(\frac{r_2}{r_1})} \tag{3.35}$$

Laplace problem

Consider first the Laplace problem with $f_1 = 0$, corresponding to thermal conduction with no heat generation or absorption within the wall of the cylinder. Take the case where $r_1 = 1$, $r_2 = 10$, $\psi_1 = 8$, and $\psi_2 = 1$. The ratio between the outer and inner radii is chosen to be quite large for reasons that should become clearer later. With these data, the potential and potential gradient distributions are

$$\psi = 8 - 3.04006 \ln r \tag{3.36}$$

$$\frac{\mathrm{d}\psi}{\mathrm{d}r} = -\frac{3.04006}{r} \tag{3.37}$$

The potential values at typical internal points are at $r = 4,\ \psi = 3.7856$ and at $r = 7, \psi = 2.0843$.

Bearing in mind that program BEM2PC cannot accept boundary segment specifications with angles greater than 180^o, the simplest mesh to represent the geometry shown in Figure 3.3 requires four segments, two on the outer boundary with end points arbitrarily chosen at (10, 0) and (-10, 0), and two on the inner boundary with end points at (0, -1) and (0, 1). These end points are numbered 1 to 4 in Figure 3.3. The following is an annotated DATA input file for this problem, with just 4 elements and nodes per semi-circular segment, 2 per quadrant. This means that each element has to represent an arc of 45^o by means of the straight chord between the ends of the arc, with the node at its centre. Note that the radii of curvature of the segments 3 and 4 on the inner boundary are both negative, because the centre of curvature is outside the boundary. The file calls for potentials to be computed at two internal points, on the axis at radii of 4 and 7.

`THICK-WALLED CYLINDER`	
`0.`	(Value of f_1)
`2`	(Number of boundaries)
`2`	(Number of segments on first boundary)
`10. 0. -10. 0.`	(Co-ordinates of the segment ends)
`10. 4 1.`	(Radii of curvature, numbers of elements
`10. 4 1.`	and length ratios)
`2`	(Number of segments on second boundary)
`0. -1. 0. 1.`	(Co-ordinates of the segment ends)
`-1. 4 1.`	(Radii of curvature, numbers of elements
`-1. 4 1.`	and length ratios)
`4 0 0`	(Four prescribed boundary potentials)
`1 1.`	(Potential = 1 on segment 1)
`2 1.`	(Potential = l on segment 2)
`3 8.`	(Potential = 8 on segment 3)
`4 8.`	(Potential = 8 on segment 4)
`2`	(Number of internal points)
`4. 0.`	(Co-ordinates of first internal point)
`7. 0.`	(Co-ordinates of second internal point)

Table 3.5 shows the results obtained for meshes ranging from 2 nodes per quadrant (4 per segment) up to 24 nodes per quadrant (48 per segment).

Nodes per Quadrant	Gradient at $r = 1$	Gradient at $r = 10$	Potential at (4, 0)	Potential at (7, 0)	Flow in at $r = 1$	Flow out at $r = 10$
Exact	3.0401	-0.30401	3.7856	2.0843	19.101	19.101
2	3.133	-0.3220	3.583	1.877	19.18	19.72
3	3.079	-0.3112	3.697	1.994	19.12	19.33
4	3.062	-0.3078	3.736	2.034	19.11	19.22
5	3.053	-0.3064	3.754	2.052	19.11	19.17
6	3.049	-0.3056	3.764	2.062	19.10	19.15
8	3.045	-0.3049	3.773	2.072	19.10	19.13
12	3.042	-0.3044	3.780	2.079	19.10	19.11
24	3.041	-0.3041	3.784	2.083	19.10	19.10

Table 3.5 Results for the thick-walled cylinder Laplace problem

Since the problem is axisymmetric it is to be expected that the computed gradients at all the nodes on the inner boundary would be identical, and similarly for all the nodes on the outer boundary. This proves to be the case, but for fluctuations in the sixth, and occasionally fifth, significant figure. This is again due to roundoff error, and becomes more noticeable as the mesh is refined. By the time 24 nodes per quadrant (192 nodes in total) is reached the fifth significant figures are definitely affected, and this represents about the maximum accuracy that can be achieved: further refinement only leads to roundoff errors exceeding any improvement in accuracy. At this level of refinement, however, the results are almost correct to four significant figures.

Note that using coarse meshes, say 2 or 3 elements per quadrant, the results are significantly inaccurate. This is because straight line elements, with nodes at their centres and not on the actual boundary, provide rather poor representations of the circular arc geometry. Accuracy clearly improves quite rapidly as the mesh is refined, and by about 12 nodes per quadrant (96 nodes in total) is adequate for most practical purposes.

Poisson problem

Consider the same problem, but of the Poisson type with $f_1 = -1$, corresponding to thermal conduction with heat generation within the wall of the cylinder. Equations 3.34 and 3.35 reduce to

$$\psi = 8.25 - \frac{r^2}{4} - 7.70872 \ln r \tag{3.38}$$

$$\frac{\mathrm{d}\psi}{\mathrm{d}r} = -\frac{r}{2} - \frac{7.70872}{r} \tag{3.39}$$

Hence, the gradients at the inner and outer surfaces of the cylinder are 7.2087 and -4.2291, respectively. The potential values at typical internal points are at $r = 4$ $\psi = 14.937$ and at $r = 7, \psi = 11.000$. The origin for the particular integral function for this Poisson problem is taken as the centre of the cylinder, which in this case coincides with the origin chosen for the global co-ordinates.

Table 3.6 again shows the results obtained for meshes ranging from 2 nodes per quadrant (4 per segment) up to 24 nodes per quadrant (48 per segment).

Nodes per Quadrant	Gradient at $r = 1$	Gradient at $r = 10$	Potential at (4, 0)	Potential at (7, 0)	Flow out at $r = 1$	Flow out at $r = 10$
Exact	-7.2087	-4.2291	14.937	11.000	45.294	265.72
2	-5.861	-3.970	13.13	8.319	35.88	243.1
3	-6.595	-4.114	14.12	9.790	40.97	255.6
4	-6.8603	-4.164	14.48	10.32	42.83	260.0
5	-6.9846	-4.188	14.64	10.56	43.70	262.1
6	-7.053	-4.201	14.73	10.70	44.19	263.2
8	-7.121	-4.213	14.83	10.83	44.67	264.3
12	-7.169	-4.222	14.89	10.93	45.01	265.1
24	-7.199	-4.227	14.92	10.98	44.22	265.6

Table 3.6 Results for the thick-walled cylinder Poisson problem

The trends are very similar to those observed for the Laplace version, with mesh refinement up to about 12 or 24 nodes per quadrant to give good accuracy.

Use of Symmetry

Returning to the Laplace problem, there is no need to analyse the whole of this axisymmetric problem, but only a small part of it. Figure 3.4 shows one quadrant of the thick-walled cylinder with the relevant boundary conditions on the curved surfaces. The normal gradients on the planes of symmetry that cut through the cylinder walls are zero. Of course, a slice substantially smaller than a quadrant could have been analysed. Not only is the size of the problem reduced, but it has been simplified from multiply-connected (two boundaries) to singly-connected (one boundary). It remains to be seen what benefits, if any, this offers. The boundary is divided into four segments, with the numbered end points as shown.

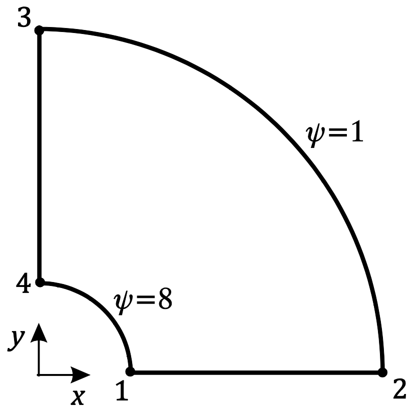


Figure 3.4 One quadrant of the thick-walled cylinder test problem

Table 3.7 shows the results obtained for meshes ranging from two nodes per segment (two per quadrant on the curved parts of the boundary) up to 24 nodes per segment. These should be compared with Table 3.5 for the whole cylinder. Note that the internal points for potential evaluation are at $(\frac{4}{\sqrt{2}}, \frac{4}{\sqrt{2}})$ and $(\frac{7}{\sqrt{2}}, \frac{7}{\sqrt{2}})$. These are at $r = 4$ and $r = 7$, but no longer on the axis, which now forms part of the boundary. The magnitudes of the potential flows in and out of the domain are reduced by factors of 4 because only a quarter of the problem is now being considered.

Nodes per Quadrant	Gradient at $r = 1$	Gradient at $r = 10$	Potential at $r = 4$	Potential at $r = 7$	Flow in at $r = 1$	Flow out at $r = 10$
Exact	3.0401	-0.30401	3.7856	2.0843	4.7753	4.7753
2	3.736	-0.3460	3.657	1.924	5.719	5.297
3	2.816	-0.3232	3.833	2.053	5.431	5.166
4	2.786	-0.3198	3.860	2.087	5.262	5.076
5	2.777	-0.3172	3.860	2.097	5.153	5.015
6	2.779	-0.3148	3.854	2.100	5.080	4.971
8	2.803	-0.3117	3.840	2.100	4.987	4.914
12	2.871	-0.3086	3.821	2.096	4.897	4.857
24	2.967	-0.3057	3.800	2.089	4.819	4.805
48	3.015	-0.3046	3.791	2.086	4.790	4.785
96	3.030	-0.3042	3.787	2.085	4.780	4.779

Table 3.7 Results for the thick-walled cylinder quadrant Laplace problem

The computed gradients on the curved parts of the boundary are no longer even nearly constant, until the mesh is considerably refined, and the values tabulated are for either the mid point or the two nodes immediately adjacent to the mid point (depending whether the number of nodes per quadrant is odd or even). The results show erratic behaviour, and are certainly less accurate than those in Table 3.5. Corresponding results for the same numbers of nodes per quadrant should agree, if the inaccuracies are due mainly to inadequate modelling using straight line elements. This is not the case, particularly at the inner curved boundary at $r = 1$, even though the potential flow (an average over the gradients) there is more accurate than the single point gradient.

The explanation for this behaviour is the nature of the mesh applied to this particular geometry. A thick-walled cylinder geometry with external radius ten times the internal radius was chosen deliberately so that the potential varies much more rapidly with radius at the inner surface than at the outer surface. While the whole cylinder was analysed this was not an issue. But the use of a uniform distribution of element size over the lines of symmetry of one quadrant of the cylinder does not adequately model the variation of potential, at least with elements over each of which the potential is assumed to be constant.

In this situation the use of non-uniform distributions of elements over the relevant segments of the boundary should be helpful. The method of varying the elements sizes which is implemented in the program is described in Section 3.1.3. The key parameter is *S*, the ratio between the lengths of successive elements in the direction of numbering. A value of *S* greater than unity is required on the first segment joining the end points numbered 1 and 2 in Figure 3.4, to ensure that the element size increases from point 1 towards point 2. On the other hand, on the third segment joining points 3 and 4, the corresponding ratio should be $1/S$, to ensure that element size decreases from point 3 towards point 4, in the direction of numbering.

Table 3.8 shows the effect of increasing the length ratio applied in this way from unity up to 1.4.

S	Gradient at $r = 1$	Gradient at $r = 10$	Potential at $r = 4$	Potential at $r = 7$	Flow in at $r = 1$	Flow out at $r = 10$
Exact	3.0401	-0.30401	3.7856	2.0843	4.7753	4.7753
1.0	2.871	-0.3086	3.821	2.096	4.897	4.857
1.1	2.960	-0.3053	3.793	2.084	4.827	4.818
1.2	3.022	-0.3033	3.778	2.076	4.799	4.808
1.3	3.045	-0.3016	3.769	2.071	4.792	4.814
1.4	3.055	-0.2999	3.761	2.066	4.793	4.825

Table 3.8 Effects of varying element length ratio for a mesh of 12 nodes per quadrant

While there is a steady improvement in gradient values at the inner cylinder surfaces for ratios up to 1.3, the values at the outer surface are deteriorating. Further increasing the ratio only makes matters worse. This is because, while there is a desirable reduction in element size at the inner surface, the elements close to the outer surface are becoming too large, with resulting loss of accuracy. As it happens, at a ratio of 1.3 in this mesh the sizes of the elements on the curved and straight segments at points 1 and 4 are roughly equal in length. Comparing the results in Table 3.8 with those for 12 nodes per quadrant in Table 3.5, it can be seen that even with a non-uniform mesh the results are not as good. At least for this problem, the use of symmetry to reduce the size of the problem results in a significant loss of accuracy. Constant boundary elements are, as is to be expected, not ideal in situations where the physical variables change rapidly. This is quite apart from their inability to accurately follow curved boundary shapes. Higher-order elements such as quadratic can be expected to offer substantial benefits.

3.3 An Example: Downstream Viscous Flow in a Rectangular Channel

A more practical problem for solution by program BEM2PC is the viscous channel flow problem described in Section 1.2.1. It is a problem that arises in, for example, the screw extrusion of materials such as thermoplastics.

Analytical solution

If the rectangular channel is infinitely wide compared with its depth, that is $H \ll W$ in Figure 1.2, the governing differential Equation 1.29 for the downstream flow is reduced to

$$\frac{\mathrm{d}^2 w}{\mathrm{d}y^2} = \frac{P_z}{\mu} \tag{3.40}$$

Integration of this ordinary differential equation with the boundary conditions defined in Equations 1.30 yields the following velocity profile

$$w = \frac{yV_z}{H} + \frac{P_z}{2\mu}(y^2 - yH) \tag{3.41}$$

The volumetric downstream flow rate can be found with the aid of Equation 1.31 as

$$Q = \frac{WHV_z}{2} - \frac{WP_zH^3}{12\mu} \tag{3.42}$$

For a channel of finite width, when the governing equation reverts to Equation 1.29, the flow rate can be expressed in a similar form as

$$Q = \frac{WHV_z}{2}F_D - \frac{WP_zH^3}{12\mu}F_P \tag{3.43}$$

The parameters F_D and F_P are known as the drag and pressure flow shape factors, respectively, because of their associations with the drag flow induced by the boundary velocity V_z and the pressure flow due to the gradient P_z. These shape factors take account of the two-dimensional nature of the problem, which can actually be solved by a series approach. The values of these factors for a particular value of the ratio of channel depth to width H/W may be obtained from

$$F_D = 16\frac{W}{H}\sum_{i=1,3,5,\dots}^{\infty}\frac{\tanh\left(\frac{\pi i H}{2W}\right)}{\pi^3 i^3} \tag{3.44}$$

$$F_P = 1 - 192\frac{H}{W}\sum_{i=1,3,5,\dots}^{\infty}\frac{\tanh\left(\frac{\pi i W}{2H}\right)}{\pi^5 i^5} \tag{3.45}$$

Equation 3.43 can be expressed in the more convenient dimensionless form

$$\pi_Q = \frac{F_D}{2} - \frac{\pi_P F_P}{12} \tag{3.46}$$

where the dimensionless flow rate and pressure gradient are defined as

$$\pi_Q = \frac{Q}{WHV_z}, \qquad \pi_P = \frac{P_zH^2}{\mu V_z} \tag{3.47}$$

The problem

The problem is to use constant boundary element program BEM2PC to compute the values of dimensionless flow rate, π_Q, for a series of values of dimensionless pressure gradient, π_P, of $-2, 0, 2$ and 4, for a channel that is five times as wide as it is deep. For $H/W = 0.2$ the particular form of Equation 3.46 is

$$\pi_Q = 0.4457 - 0.07283\,\pi_P \tag{3.48}$$

where the constants, based on the shape factors given by Equations 3.44 and 3.45, are given to four significant figures. Note that a positive pressure gradient, indicating pressure increasing with normal to the channel cross-section in the direction of the movement of the top boundary tends to cause flow in the opposite direction, and hence a reduction in the overall flow rate. A negative pressure gradient, on the other hand, enhances the flow rate.

Integration of potential over the domain

Before attempting to solve the problem, it is necessary to extend the program to compute the integral of potential over the solution domain in order to be able to find the volumetric flow rate.

To write a piece of program to meet this requirement for any domain, of arbitrary shape with an arbitrary number of boundaries, is not straightforward, and is beyond the scope of this book. The approach adopted is therefore to modify subprogram INTERNAL as follows to carry out the required integration for a domain which is rectangular in shape (but of arbitrary depth to width ratio), and has values of potential prescribed on each of its four edges (the four values can be different).

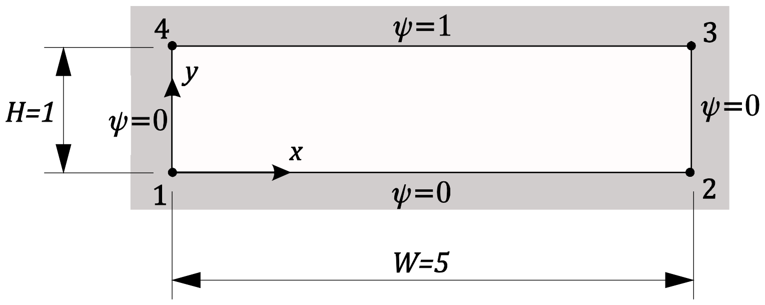


Figure 3.5 Rectangular channel cross-section solution domain

Figure 3.5 shows the channel cross-section with its boundary divided into four segments, with their ends numbered in order from the bottom left hand corner.

```
      SUBROUTINE INTERNAL
!
!  SUBPROGRAM TO FIND AND OUTPUT THE INTEGRAL OF POTENTIAL OVER THE
!  AREA OF THE SOLUTION DOMAIN.
!  THIS VERSION IS DESIGNED FOR A RECTANGULAR DOMAIN.
!
      USE SHAREDDATA2PC
      REAL :: PSIVAL(300),YINTGL(300)
!
!  INPUT THE NUMBERS OF SAMPLING POINTS INSIDE THE DOMAIN IN THE X
!  AND Y DIRECTIONS, WHICH SHOULD BOTH BE ODD.
      READ(5,*) NX,NY
      IF(MOD(NX,2) == 0) THEN
        WRITE(6,61) NX
 61     FORMAT(/ "WARNING - NX =",I4," NOT ODD - INCREASE BY 1")
        NX=NX+1
      END IF
      IF(MOD(NY,2) == 0) THEN
        WRITE(6,62) NY
 62     FORMAT(/ "WARNING - NY =",I4," NOT ODD - INCREASE BY 1")
        NY=NY+1
      END IF
      IF(NX+2 > 300 .OR. NY+2 > 300) THEN
        WRITE(6,63) NX,NY
 63     FORMAT(/ "NX =",I6,5X,"NY =",I6,5X,"ARRAY DIMENSIONS EXCEEDED",
     &            " IN INTEGRATION OVER DOMAIN")
        STOP
      END IF
```

```
!
!  FIND PRESCRIBED BOUNDARY VALUES OF POTENTIAL.
      For each segment in turn: DO IBCP=1,NBCP
      IF(ISEGBC(IBCP) ==  1) PSIBOT=PSISEG(IBCP)
      IF(ISEGBC(IBCP) ==  2) PSIRIGHT=PSISEG(IBCP)
      IF(ISEGBC(IBCP) ==  3) PSITOP=PSISEG(IBCP)
      IF(ISEGBC(IBCP) ==  4) PSILEFT=PSISEG(IBCP)
      END DO For each segment in turn
!
!  FIND COORDINATES OF DOMAIN SIDES.
      YBOT=YSEND(1)
      XRIGHT=XSEND(2)
      YTOP=YSEND(3)
      XLEFT=XSEND(4)
!
!  OUTER LOOP FOR INTEGRATION IN X DIRECTION.
      HX=(XRIGHT-XLEFT)/FLOAT(NX+1)
      HY=(YTOP-YBOT)/FLOAT(NY+1)
      IXMAX=NX+2
      IYMAX=NY+2
      YINTGL(1)=PSILEFT*(YTOP-YBOT)
      YINTGL(IXMAX)=PSIRIGHT*(YTOP-YBOT)
      For each X position in turn: DO IX=2,IXMAX-1
      PSIVAL(1)=PSIBOT
      PSIVAL(IYMAX)=PSITOP
      XPOINT=XLEFT+HX*FLOAT(IX-1)
!
!  INNER LOOP FOR INTEGRATION IN Y DIRECTION.
!
!  FIRST FIND POTENTIALS AT THE INTERNAL POINTS.
      For each Y position in turn: DO IY=2,IYMAX-1
      YPOINT=YBOT+HY*FLOAT(IY-1)
      XNODE(1)=XPOINT/MAXL
      YNODE(1)=YPOINT/MAXL
!
!  INTEGRATE ROUND THE BOUNDARY.
      SUM=0.
      Each node in turn: DO J=1,NNP
      CALL KERNEL(1,J,AIJ,BIJ)
      SUM=SUM-AIJ*PSI(J)+BIJ*DPSI(J)
```

```
          END DO Each node in turn
          PSIIP=SUM*0.5/PI
          PSIIPT=PSIIP
!
!   ADD PARTICULAR INTEGRAL.
          IF(F1 /= 0.) THEN
            XX=XNODE(1)-XC
            YY=YNODE(1)-YC
            PSIIPT=PSIIPT+0.25*F1*(XX**2+YY**2)
          END IF
          PSIVAL(IY)=PSIIPT
          END DO For each Y position in turn
!
!   INTEGRATE POTENTIAL IN Y DIRECTION.
          SUM=0.
          For alternate points in the Y direction: DO IY=2,IYMAX-1,2
          SUM=SUM+PSIVAL(IY-1)+4.*PSIVAL(IY)+PSIVAL(IY+1)
          END DO For alternate points in the Y direction
          YINTGL(IX)=SUM*HY/3.
          END DO For each X position in turn
```

```
!
!  NOW INTEGRATE IN X DIRECTION.
      SUM=0.
      For alternate points in the X direction: DO IX=2,IXMAX-1,2
      SUM=SUM+YINTGL(IX-1)+4.*YINTGL(IX)+YINTGL(IX+1)
      END DO For alternate points in the X direction
      PSIINTG=SUM*HX/3.
      AREA=(XRIGHT-XLEFT)*(YTOP-YBOT)
      APSIINTG=PSIINTG/AREA
!
!  OUTPUT RESULT.
      WRITE(6,64) PSIINTG,APSIINTG,NX,NY
 64   FORMAT(/ "INTEGRAL OF POTENTIAL OVER SOLUTION DOMAIN =",E15.6,
     &       / "AREA AVERAGE OVER SOLUTION DOMAIN =",E15.6,
     &       / "NUMBERS OF INTERNAL POINTS IN X AND Y DIRECTIONS =",
     &         I5,5X,"AND", I5)
!
      RETURN
      END SUBROUTINE INTERNAL
```

The new version of subprogram INTERNAL assumes that the segment ends are numbered as in Figure 3.5, but takes both the channel dimensions H and W, and the values of potential prescribed on the boundary edges from the data already available in the program. The only new information which has to be read in are the numbers of internal points in the two global co-ordinate directions to be used for numerical integration (NX and NY). It is normally appropriate to use numbers similar in magnitude to those of the nodes on the corresponding edges.

The method of integration used is Simpson's Rule. Given a function $g(x)$ which is defined over a small range of x by means of discrete values g_1, g_2 and g_3, and as shown in Figure 3.6, the spacing, h, of these values in the x direction being uniform, the integral of the function is given approximately by

$$\int_a^b g(x)\mathrm{d}x \approx \frac{h}{3}(g_1 + 4g_2 + g_3) \tag{3.49}$$

As indicated in Appendix A, this is a special case of Gaussian quadrature. This formula can be repeated for more than three uniformly spaced function values, provided that the total number of values is odd.

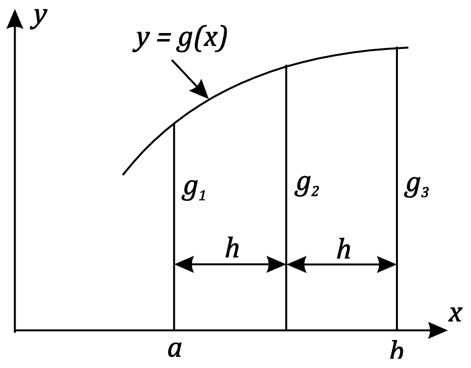


Figure 3.6 Numerical integration using Simpson's rule

The subprogram tests NX and NY for oddity, and if necessary increases an even number by one. The total numbers of function values in each direction to be used in the integration are each two more these, because the boundary values are included. The maximum size of these numbers is 300, imposed by declared array sizes.

The prescribed boundary values of potential are obtained from the segment values for each of the bottom, right, top and left hand edges of the channel, and stored in PSIBOT, PSIRIGHT, PSITOP, and PSILEFT. Similarly, the global co-ordinates of the same sides are found from appropriate segment end co-ordinates and stored in YBOT, XRIGHT, YTOP, and XLEFT. This allows the spacings in the x and y directions between columns and rows of internal points to be found and stored in HX and HY. Numerical integration is first carried out in the y direction. For each vertical column of points in turn the bottom and top values are found from the boundary values of potential. Then, for each internal point in turn, the potential is computed exactly as described for the first version of subprogram INTERNAL, and stored in array PSIVAL. Then Simpson's Rule is applied repeatedly from bottom to top of the column of points, the resulting integral being stored in array YINTG. The process is then applied for integration in the x direction, the function to be integrated now being represented by the array of y direction integrals YINTGL. Note that the values of this at the left and right hand boundaries are the integrals of the constant boundary potentials PSILEFT and PSIRIGHT, in other words the simple products of the potential values and the height of the domain. The result of the double integral with respect to both x and y is stored in PSIINTG. The area average is also found, and the two values written out, together with the numbers of internal points used in the integration.

Application to channel flow

The boundary conditions used are shown in Figure 3.5: a unit value of potential on the top edge, zero on the other three. In terms of the physical problem this is $V_z = 1$. Consequently, the dimensionless flow rate defined in Equations 3.47 can be obtained from the program as the area average of the integral of potential (downstream velocity) over the channel cross-section. From Equations 1.29 and 2.1

$$f_1 = \frac{P_z}{\mu} \tag{3.50}$$

and since $H = 1$ and $V_z = 1$ it follows from Equations 3.47 that

$$\pi_P = \frac{P_z H^2}{\mu V_z} = f_1 \tag{3.51}$$

Table 3.9 shows the results obtained for the relationship between π_Q and π_P for a range of number of nodes per edge of the domain (also number of internal points in each direction for the domain integration) from 3 to 241. The origin chosen for the particular integral function is the centre of the channel, at (2.5, 0.5).

Nodes per edge	$\pi_P = -2$	$\pi_P = 0$	$\pi_P = 2$	$\pi_P = 4$
Exact π_Q:	0.5914	0.4457	0.3000	0.1544
3	0.6677	0.4271	0.1864	-0.0542
7	0.6097	0.4430	0.2763	0.1095
15	0.5958	0.4453	0.2948	0.1444
31	0.5925	0.4457	0.2989	0.1521
61	0.5917	0.4457	0.2998	0.1538
121	0.5915	0.4457	0.3000	0.1543
241	0.5914	0.4457	0.3000	0.1544

Table 3.9 Dimensionless flow rates for $H/W = 0.2$

For the Laplace case $(\pi_P = 0)$ the convergence with mesh refinement is good, with three significant figure accuracy being obtained at 15 nodes per edge, four figures at 31 nodes per edge. This is because the velocity distribution is simple with, at least at the centre of the channel width, an essentially linear variation from bottom to top. With a pressure gradient, which changes the velocity profile from nearly linear to much more parabolic, there are more rapid changes in velocity near the top and bottom edges, which are somewhat more difficult to analyse. Nevertheless four significant figure accuracy is obtained by the time mesh refinement reaches 241 nodes per edge. The only parameter being assessed is the flow rate which is obtained by integration, a smoothing process, and therefore less subject to roundoff errors than individual values of velocity.

It is worth noting what happens at the top corners of the channel, the points labelled 3 and 4 in Figure 3.5. There the potential boundary condition is discontinuous, from the fixed sides of the channel to the moving top surface. Treatment of the corner points in general requires some care. Using constant boundary elements this is not an issue, because no node is located at any corner point, the nearest ones being at the centres of the elements which meet at the corner. This is not the case with higher-order elements such as quadratic, which are the subject of the next chapter.

Problems

3.1 A viscous liquid flows in a pipe of circular cross-section under the action of a pressure gradient. The pipe has an internal diameter of 150 mm, the viscosity of the fluid is 9.5 Ns/m^2 and the pressure gradient is 120 kN/m^3. Use program BEM2PC to compute the maximum velocity, to an accuracy of three significant figures. Compare this result with the exact analytical solution to the problem. Take advantage of the axial symmetry of the problem to analyse a 30^o slice of the cross-section, and compare the results.

3.2 The pipe in Problem 3.1 is changed from circular to elliptical cross-section with minor and major axes of 150 and 300 mm. Using program BEM2PC to analyse the entire cross-section, compute the maximum velocity, to an accuracy of three significant figures.

3.3 The pipe in Problem 3.1 is changed from circular to isosceles triangular cross-section with a base length of 150 mm and a height of 150 mm. Use program BEM2PC to compute the maximum velocity, to an accuracy of three significant figures.

3.4 Use program BEM2PC to find the torsional stiffness of a steel bar whose uniform cross-section is rectangular, with dimensions 60 mm by 30 mm. Take the shear modulus for steel to be 79.6 GN/m^2.

3.5 An electrical conductor having a square cross-section 25 mm by 25 mm carries a heavy current which results in a uniform heat generation within the metal of 16 MW/m^3. The conductor is made of copper, with a thermal conductivity of 295 W/m^{o}C. Use program BEM2PC to find the maximum temperature at the surface of the copper, and the rates of heat transfer at the uninsulated faces: (a) when three faces of the conductor are thermally insulated and the fourth is cooled to a temperature of 25^{o}C, (b) when two adjacent faces of the conductor are thermally insulated and both the other two faces are cooled to 25^{o}C.

3.6 Repeat Problem 3.5 for a conductor of circular cross-section of 25 mm diameter, with its outer surface cooled to 25^{o}C, finding the maximum temperature within the copper and the total rate of heat transfer at its surface.

3.7 A long thick-walled cylinder has external and internal diameters of 150 mm and 75 mm, respectively. The material of the cylinder has a thermal conductivity of 0.35 W/m^{o}C. Heat is generated within the body of the material at a rate of 12 kW/m^3. The outer surface of the cylinder is thermally insulated, and the inner surface is cooled. Use program BEM2PC to find the maximum temperature in the cylinder and the rate of heat transfer to the coolant when (a) the inner surface is cooled to a uniform temperature of 40^{o}C, (b) when the coolant temperature is 25^{o}C and the heat transfer coefficient at the inner surface is 15 W/m^{2o}C.

3.8 A steel conductor has a rectangular cross-section with three circular cooling passages running through it. In terms of Cartesian co-ordinates in the plane of the cross-section, and in mm units, the corners are at (0, 0), (100, 0), (100, 80), (0, 80). The first hole is 15 mm radius with its centre at (30, 40). The second hole is 10 mm radius with its centre at (80, 20). The third hole is 10 mm radius with its centre at (70, 60). The surface temperature of all three holes is 10^{o}C. If the rectangular outer surface is exposed to a fluid with a temperature of 150^{o}C and the surface heat transfer coefficient is 250 W/m^{2o}C, use program BEM2PC to find the maximum temperature on the outer surface of the conductor and the rates of heat transfer into each of the three holes. Take the thermal conductivity of steel to be 50 W/m^{o}C.

3.9 Problem 1.2 refers to the problem of a thin membrane under lateral pressure and gives the governing partial differential equation. A thin membrane is stretched over a frame which is trapezoidal in shape. In metre units the four corners of the trapezium can be defined in terms of Cartesian co-ordinates in its plane of (0, 0), (1.2, 0), (1.2, 0.6), (0, 1.2). The lateral pressure is $p = 50$ N/m^2 and the uniform tensile force in the membrane is $S = 72$ N/m. Use program BEM2PC to find the magnitude and location of the maximum deflection of the membrane.

3.10 Problem 1.3 introduces the problem of water seepage through porous rock underneath a concrete dam, and is concerned with the governing equation and its boundary conditions. A dam which is 10 m thick at its base can be modelled as being built on a flat plane of rock. It retains water to a depth of 15 m. If the permeability of the rock is $\kappa = 2 \times 10^{-10}$ m^4/Ns (see Equations 1.58), use program BEM2PC to estimate the rate of seepage flow under the dam.

4 Quadratic Boundary Element Program for Potential Problems

In this chapter a Fortran computer program to implement the quadratic boundary element formulation for two-dimensional potential problems developed in Chapter 2 is presented and described in detail. It is then used to solve a range of problems to demonstrate the capabilities of the method, including those solved using constant boundary elements described in Chapter 3.

For readers who prefer to use Matlab, a translation is provided in Appendix D.

As far as possible the program structure, file names, variable names, and actual coding follow those used in the constant element program. In terms of programming, there are several important differences between constant and boundary elements. Firstly, analytical integration is replaced by Gaussian quadrature, which requires more coding. Secondly, with quadratic elements there is no longer a one-to-one correspondence between nodes and elements: under some circumstances the focus has to be on elements, under others on nodes. Thirdly, using quadratic elements, nodes are placed at the corners of solution domains, which is a benefit in terms of accuracy but is a complication in terms of programming. For example, the direction of the normal to the boundary of a solution domain is not uniquely defined at a corner, something that was not an issue with constant elements.

The principle adopted in programming the quadratic element analysis is to try to make the coding straightforward to understand and follow, rather than necessarily the most efficient in terms of computation. A good example of this occurs in the integration of the kernel function products over an element to form the coefficient matrices (in subprogram INTKER). As programmed, the outer DO loop is for the nodes of the element, and the inner one for the Gauss points. Interchanging these substantially reduces the number of times the Jacobians and normals at the Gauss points have to be re-computed, but the program is less clear.

4.1 Program BEM2PQ

The program name indicates that it is for **b**oundary **e**lement **m**ethod analysis of **two**-dimensional **p**otential problems using **q**uadratic elements. Each of the subprogram units which make up the whole are described in turn. The Preface explains how the full program can be accessed as a single file.

As with program BEM2PC, much of the coding is concerned with the definition of the arrangement of elements on the boundary (or boundaries) of the solution domain, and the application of boundary conditions to them. Again, entering the co-ordinates of the nodes of each and every element, followed by the type and magnitude of the relevant boundary condition applied to each is an option, but tedious. Instead, each boundary can be divided up into a series of boundary segments, which are either straight lines or circular arcs. The number of elements within a segment can then be chosen, and varied easily, and from which the program generates all the element geometric data. The elements on a segment do not have to be uniform in size, but can be varied in length by a constant ratio between successive elements. In the present version of the program, each segment is subject to only one uniform boundary condition, be it prescribed potential, prescribed potential gradient, or prescribed mixed condition connecting potential and potential gradient. The program distributes this condition to all the elements involved. Consequently, the ends of segments are conveniently defined as points where there is a significant change in either shape (a corner, for example) or boundary condition. Use of this facility is demonstrated later in this chapter.

4.1.1 Main program

At the beginning of the program is a storage module named SHAREDDATA2PQ which allows stored data to be accessed and shared by all those subprograms that require it (by means of a USE statement). The dimensioned array sizes in the module allow for up to 500 quadratic boundary elements with 1000 nodes, with up to 10 different boundaries forming the solution domain. The maximum number of linear equations to be solved is 1000. A dictionary of the variable names used is provided at the beginning of the book.

The main program named BEM2PQ is designed mainly to call each of the other subprograms in turn. It does, however, also serve to name the files with which the program communicates via OPEN statements. File DATA, which is addressed as file number 5 in the program, serves to supply the input data which defines the current problem. The main output of results is to file RESULTS, addressed as file number 6. Element mesh data, on the other hand, are output to file MESHRES (mesh results) and numbered 7.

```
MODULE SHAREDDATA2PQ
!
! MODULE STORING SHARED DATA.
!
    REAL :: XEEND(510),YEEND(510),XNODE(1000),YNODE(1000)
    REAL :: XSEND(500),YSEND(500),UNX(1000),UNY(1000)
    REAL :: UNMX(500,3),UNMY(500,3),DPSIPIM(500,3)
    REAL :: F1,XC,YC,PSI(1000),DPSI(1000),ALPHA(500),BETA(500)
    REAL :: ALPHAN(1000),BETAN(1000)
    REAL :: PSISEG(500),DPSISEG(500),ALPHASEG(500),BETASEG(500)
    REAL :: STORE(500),PSIPI(1000),DPSIPI(1000),PSIT(1000)
    REAL :: DPSIT(1000),FLOWSEG(500),ANGSTORE(510)
    REAL :: PI,A(1000,1001),AROW(500,3),BROW(500,3),MAXL
    REAL :: ZG(8),WG(8),EGL(8),WGL(8),JACOB,UNGX,UNGY
    REAL :: SF(3,8),SD(3,13),SFL(4,3,8),SDL(4,3,8)
    INTEGER :: NEL,NNP,MAXNEL,MAXNNP,MAXNB,NEEND,NGAUSS
    INTEGER :: NODE(500,3),M1(500),M3(500)
    INTEGER :: NBOUND,NSEGTOT,NELB(10),NSEGB(10)
    INTEGER :: NBCP,NBCD,NBCM,NBCT,IBCE(500),IBCN(1000),ISEGBC(500)
    INTEGER :: ISEGEND(500),ISEGELEM(500),MFIRST(500),MLAST(500)
    END MODULE SHAREDDATA2PQ
    PROGRAM BEM2PQ
!
! PROGRAM FOR SOLVING TWO DIMENSIONAL POTENTIAL PROBLEMS BY THE BOUNDARY
! ELEMENT METHOD USING QUADRATIC ELEMENTS.
!
    USE SHAREDDATA2PQ
    OPEN(5,FILE="DATA")
    OPEN(6,FILE="RESULTS")
    OPEN(7,FILE="MESHRES")
    PI=4.0*ATAN(1.)
!
! DEFINE THE MAXIMUM PROBLEM SIZE PERMITTED BY THE ARRAY DIMENSIONS.
    MAXNEL=500
    MAXNNP=1000
    MAXNB=10
!
! INPUT THE PROBLEM TITLE AND TYPE.
    CALL INTITLE
!
! INPUT AND GENERATE THE MESH DATA.
    CALL MESHQ
!
! OUTPUT THE MESH DATA.
    CALL MSHOUT
!
! INPUT, PROCESS AND OUTPUT THE BOUNDARY CONDITIONS.
    CALL BCS
!
! EVALUATE AND STORE VALUES OF THE SHAPE FUNCTIONS AND THEIR DERIVATIVES
! AT THE GAUSS POINTS AND NODES.
    CALL SHAPE
!
! IF GOVERNING EQUATION IS POISSON TYPE, MODIFY THE BOUNDARY CONDITIONS.
    CALL POISSON
!
! FORM THE COEFFICIENT MATRIX AND APPLY THE BOUNDARY CONDITIONS.
    CALL FRMTRX
!
! SOLVE THE LINEAR EQUATIONS.
    MAXNNPP1=MAXNNP+1
    CALL ELIMIN(A,PSI,NNP,MAXNNP,MAXNNPP1,IFLAG)
    IF(IFLAG == 1) THEN
    WRITE(6,61)
  61 FORMAT(/ "MATRIX ILL-CONDITIONING DETECTED IN EQUATION SOLVER")
    STOP
    END IF
!
! OUTPUT NODAL POINT VALUES OF POTENTIAL AND POTENTIAL GRADIENT,
! ALSO POTENTIAL FLOWS ACROSS BOUNDARY SEGMENTS.
    CALL OUTPUT
!
! COMPUTE VALUES OF POTENTIAL AT INTERNAL POINTS.
    CALL INTERNAL
!
    STOP
    END PROGRAM BEM2PQ
```

After defining the maximum numbers of boundary elements (MAXNEL), nodes (MAXNNP) and boundaries (MAXNB) permitted by the array dimensions, the main program calls the following subprograms in turn:

INTITLE for the problem title,
MESHQ to input and create the mesh data,
MSHOUT to write out the mesh data, BCS for the boundary conditions,
BCS for the boundary conditions,
SHAPE for defining the element shape functions,
POISSON to compute the particular integral if the problem is of the Poisson type,
FRMTRX to define the $[A]$ and $[B]$ coefficient matrices,
ELIMIN to solve the equations,
OUTPUT to write out the results, and finally
INTERNAL to find values of potential at any desired point within the domain.

As in program BEM2PC, the matrix ill-conditioning warning is most likely to be triggered by trying to solve a poorly defined problem, such as one with only potential gradient or mixed boundary conditions, and potential nowhere defined.

4.1.2 Subprogram INTITLE

An alphanumeric title for the problem is first read into TITLE. Next the constant value of function (Equation 2.1) is read into F1. If this value is zero the problem is of the Laplace type, and an appropriate message is written out. If it is not zero the problem is Poisson type and the global co-ordinates of the origin used to define to particular integral are read into XC and YC, normally chosen to be roughly at the centre of the domain. The value of together with the origin co-ordinates are then written out.

```
      SUBROUTINE  INTITLE
!
!  SUBPROGRAM TO INPUT PROBLEM TITLE AND TYPE (LAPLACE OR POISSON).
!
      USE SHAREDDATA2PQ
      CHARACTER(80) :: TITLE
!
!  INPUT THE PROBLEM TITLE.
      READ(5,FMT="(A80)") TITLE
      WRITE(6,61) TITLE
 61   FORMAT("QUADRATIC BOUNDARY ELEMENT SOLUTION FOR",
     &       " TWO DIMENSIONAL POTENTIAL PROBLEM"  // A)
!
!  INPUT THE VALUE OF THE (CONSTANT) F1 FUNCTION IN THE GOVERNING
!  EQUATION.
      READ(5,*) F1
      IF(F1 == 0.) WRITE(6,62)
 62   FORMAT(/ "LAPLACE EQUATION")
      IF(F1 /= 0.) THEN
        WRITE(6,63) F1
 63     FORMAT(/ "POISSON EQUATION,  F1 = ",E12.4,"  CONSTANT")
!
!  INPUT THE COORDINATES OF THE ORIGIN FOR THE PARTICULAR INTEGRAL.
        READ(5,*) XC,YC
        WRITE(6,64) XC,YC
 64     FORMAT(/ "ORIGIN FOR PARTICULAR INTEGRAL:",5X,"X =",E12.4,5X,
     &           "Y =",E12.4)
      END IF
!
      RETURN
      END SUBROUTINE INTITLE
```

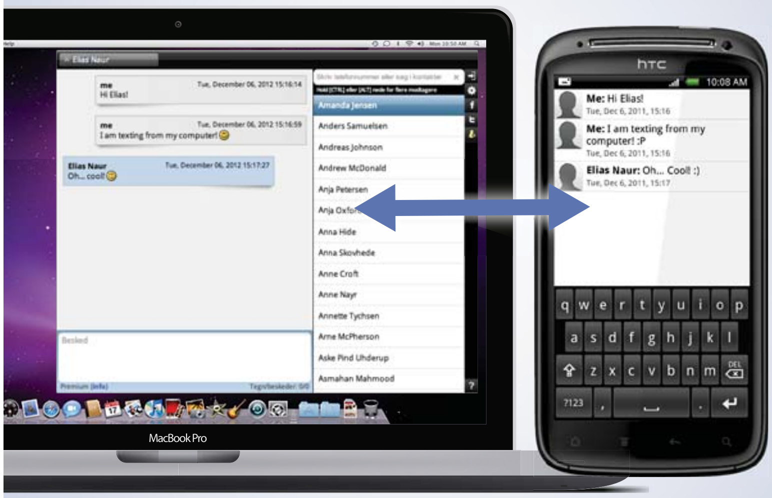
4.1.3 Subprogram to input and generate the element mesh data

Subprogram MESHQ reads in a minimal amount of information from which to create the mesh of quadratic boundary elements. The process is complicated by the fact that there are several levels of geometric features. Firstly, boundaries: there will be more than one for a multiply-connected domain, with internal holes within the outer boundary. Secondly, segments of boundaries whose ends are located at changes of either geometry or boundary condition, and can be either straight or follow circular arcs. Thirdly, the ends of elements which are distributed either uniformly or non-uniformly along the segments. Finally, the nodes of the elements, which are located at the element end points and mid points.

```
      SUBROUTINE  MESHQ
!
!  SUBPROGRAM TO READ IN AND GENERATE THE GEOMETRIC DATA FOR A MESH OF
!  QUADRATIC ELEMENTS.
!
      USE SHAREDDATA2PQ
!
!  INPUT THE NUMBER OF SEPARATE BOUNDARIES.
      READ(5,*) NBOUND
!
!  TEST THE NUMBER OF BOUNDARIES.
      IF(NBOUND < 1 .OR. NBOUND > MAXNB) THEN
        WRITE(6,61) NBOUND,MAXNB
 61     FORMAT(/ "NBOUND =",I4,2X,"OUTSIDE PERMITTED RANGE 1 TO",I4)
        STOP
      END IF
!
!  FOR EACH BOUNDARY IN TURN INPUT THE NUMBER OF SEGMENTS.
      NEL=0
      IEEND=0
      NSEGTOT=0
      MMAXB=0
      Each boundary in turn: DO IBOUND=1,NBOUND
      NELB(IBOUND)=0
      MMINB=MMAXB+1
      READ(5,*) NSEGB(IBOUND)
      NSEGTOT=NSEGTOT+NSEGB(IBOUND)
!
!  TEST THE NUMBER OF SEGMENTS.
      IF(NSEGTOT < 1 .OR. NSEGTOT > MAXNEL) THEN
        WRITE(6,62) NSEGTOT,MAXNEL
 62     FORMAT(/ "NSEGTOT =",I6,2X,"OUTSIDE PERMITTED RANGE 1 TO",I6)
        STOP
      END IF
!
!  INPUT THE CARTESIAN GLOBAL COORDINATES OF THE END POINTS OF THE
!  SEGMENTS.  TAKE THE END POINTS CONSECUTIVELY, KEEPING THE DOMAIN
!  TO THE LEFT OF THE DIRECTION OF NUMBERING.
      READ(5,*) (XSEND(ISEND),YSEND(ISEND),ISEND=1,NSEGB(IBOUND))
!
!  DEFINE THE FIRST END POINT ON THE CURRENT BOUNDARY.
      IEEND=IEEND+1
      XEEND(IEEND)=XSEND(1)
      YEEND(IEEND)=YSEND(1)
!
!  FOR EACH OF THE SEGMENTS (BETWEEN ENDS 1 AND 2, 2 AND 3, ETC.)
!  INPUT THE RADIUS OF CURVATURE (+VE FOR CONVEX WITH CENTRE OF
!  CURVATURE INSIDE DOMAIN, -VE FOR CONCAVE), THE NUMBER OF
!  ELEMENTS IN THE SEGMENT, AND THE LENGTH RATIO BETWEEN SUCCESSIVE
!  ELEMENTS IN THE DIRECTION OF NUMBERING.
      ISEGMAX=NSEGTOT
```

```
      ISEGMIN=ISEGMAX-NSEGB(IBOUND)+1
      Each segment in turn: DO ISEG=ISEGMIN,ISEGMAX
      READ(5,*) RSEG,NELSEG,RATSEG
!
!  FIND AND TEST THE NUMBER OF ELEMENTS SO FAR.
      NEL=NEL+NELSEG
      NELB(IBOUND)=NELB(IBOUND)+NELSEG
      IF(NEL < 1 .OR. NEL > MAXNEL) THEN
        WRITE(6,63) NEL,MAXNEL
 63     FORMAT(/ "NEL =",I6,2X,"OUTSIDE PERMITTED RANGE 1 TO",I6)
        STOP
      END IF
!
!  FIRST AND LAST ELEMENTS ON CURRENT SEGMENT.
      MLAST(ISEG)=NEL
      MFIRST(ISEG)=NEL-NELSEG+1
      MMAXB=MMAXB+NELSEG
!
!  COORDINATES OF THE FIRST END POINT OF THE SEGMENT.
      ISEND=ISEG-ISEGMIN+1
      XFIRST=XSEND(ISEND)
      YFIRST=YSEND(ISEND)
!
!  COORDINATES OF THE LAST END POINT OF THE SEGMENT.
      ISEND=ISEND+1
      IF(ISEG == ISEGMAX) ISEND=1
      XLAST=XSEND(ISEND)
      YLAST=YSEND(ISEND)
!
!  GENERATE ELEMENT DATA FOR A STRAIGHT SEGMENT.
      IF(RSEG == 0.) THEN
!
!  DEFINE THE ELEMENT END POINT COORDINATES ON THE SEGMENT.
        Each element in turn: DO M=1,NELSEG
        IEEND=IEEND+1
        ISEGEND(IEEND)=ISEG
        IF(RATSEG == 1.) THEN
          XEEND(IEEND)=XFIRST+(XLAST-XFIRST)*FLOAT(M)/FLOAT(NELSEG)
          YEEND(IEEND)=YFIRST+(YLAST-YFIRST)*FLOAT(M)/FLOAT(NELSEG)
        ENDIF
        IF(RATSEG /= 1.) THEN
          XEEND(IEEND)=XFIRST+(XLAST-XFIRST)*(1.-RATSEG**M)
     &                     /(1.-RATSEG**NELSEG)
          YEEND(IEEND)=YFIRST+(YLAST-YFIRST)*(1.-RATSEG**M)
     &                     /(1.-RATSEG**NELSEG)
        END IF
        END DO Each element in turn
!
!  DEFINE THE ELEMENT NODES AND COORDINATES.
        IEEND=IEEND-NELSEG
        Each element in turn: DO IELSEG=1,NELSEG
        IEEND=IEEND+1
        M=MFIRST(ISEG)+IELSEG-1
        I1=2*M-1
        I2=I1+1
        I3=I1+2
        IF(ISEG == ISEGMAX .AND. IELSEG == NELSEG) I3=NODE(MMINB,1)
        NODE(M,1)=I1
        NODE(M,2)=I2
        NODE(M,3)=I3
        ISEGELEM(M)=ISEG
        IF(ISEG == ISEGMIN .AND. IELSEG == 1) THEN
          XNODE(I1)=XEEND(IEEND-1)
          YNODE(I1)=YEEND(IEEND-1)
        END IF
        XNODE(I3)=XEEND(IEEND)
        YNODE(I3)=YEEND(IEEND)
        XNODE(I2)=0.5*(XNODE(I1)+XNODE(I3))
        YNODE(I2)=0.5*(YNODE(I1)+YNODE(I3))
!
!  STORE THE NUMBERS OF THE ADJACENT ELEMENTS.
        M1(M)=M-1
        M3(M)=M+1
        END DO Each element in turn
      END IF
!
!  GENERATE ELEMENT DATA FOR A SEGMENT IN THE FORM OF A CIRCULAR ARC.
      IF(RSEG /= 0.) THEN
```

```
!
!  LOCATE THE CENTRE OF THE ARC.
        XMID=(XFIRST+XLAST)/2.
        YMID=(YFIRST+YLAST)/2.
        ALSEG=SQRT((XLAST-XFIRST)**2+(YLAST-YFIRST)**2)
        ALPERP2=RSEG**2-(ALSEG/2.)**2
        IF(ABS(ALPERP2) < 1.E-6*RSEG**2) ALPERP2=0.
        IF(ALPERP2 < -1.E-6*RSEG**2) THEN
          WRITE(6,64) ISEG
 64       FORMAT(/ "DATA ERROR FOR SEGMENT NUMBER",I6,
     &           / "NOT POSSIBLE TO CREATE A CIRCULAR ARC")
          STOP
        END IF
        ALPERP=SQRT(ALPERP2)
        UVFLX=(XLAST-XFIRST)/ALSEG
        UVFLY=(YLAST-YFIRST)/ALSEG
        FACT=1.
        IF(RSEG < 0.) FACT=-1.
        XCENT=XMID-ALPERP*UVFLY*FACT
        YCENT=YMID+ALPERP*UVFLX*FACT
```

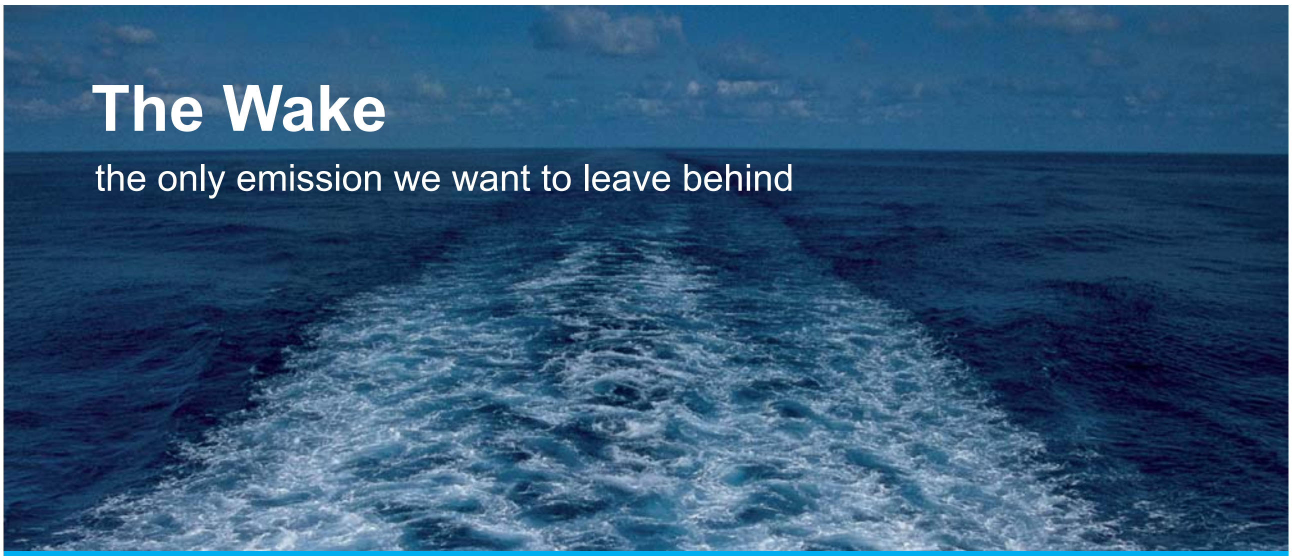


```
!
!  FIND THE ANGLE SUBTENDED THERE BY THE SEGMENT.
       IF(ALPERP /= 0.) ANGSEG=2.*ATAN(ALSEG*0.5/ALPERP)
       IF(ALPERP == 0.) ANGSEG=PI
!
!  DEFINE THE ELEMENT END POINT COORDINATES ON THE SEGMENT.
       ANGFIR=ATAN2(YFIRST-YCENT,XFIRST-XCENT)
       ANGSTORE(IEEND)=0.
       Each element in turn: DO M=1,NELSEG
       IEEND=IEEND+1
       ISEGEND(IEEND)=ISEG
       IF(RATSEG == 1.) ANG=ANGSEG*FLOAT(M)/FLOAT(NELSEG)
       IF(RATSEG /= 1.) ANG=ANGSEG*(1.-RATSEG**M)/(1.-RATSEG**NELSEG)
       IF(RSEG < 0.) ANG=-ANG
       XEEND(IEEND)=XCENT+ABS(RSEG)*COS(ANGFIR+ANG)
       YEEND(IEEND)=YCENT+ABS(RSEG)*SIN(ANGFIR+ANG)
       ANGSTORE(IEEND)=ANG
       END DO Each element in turn
!
!  DEFINE THE ELEMENT NODES AND COORDINATES.
       IEEND=IEEND-NELSEG
       Each element in turn: DO IELSEG=1,NELSEG
       IEEND=IEEND+1
       M=MFIRST(ISEG)+IELSEG-1
       I1=2*M-1
       I2=I1+1
       I3=I1+2
       IF(ISEG == ISEGMAX .AND. IELSEG == NELSEG) I3=NODE(MMINB,1)
       NODE(M,1)=I1
       NODE(M,2)=I2
       NODE(M,3)=I3
       ISEGELEM(M)=ISEG
       IF(ISEG == ISEGMIN .AND. IELSEG == 1) THEN
         XNODE(I1)=XEEND(IEEND-1)
         YNODE(I1)=YEEND(IEEND-1)
       END IF
       XNODE(I3)=XEEND(IEEND)
       YNODE(I3)=YEEND(IEEND)
       ANG=0.5*(ANGSTORE(IEEND-1)+ANGSTORE(IEEND))
```

```
         XNODE(I2)=XCENT+ABS(RSEG)*COS(ANGFIR+ANG)
         YNODE(I2)=YCENT+ABS(RSEG)*SIN(ANGFIR+ANG)
!
!  STORE THE NUMBERS OF THE ADJACENT ELEMENTS.
         M1(M)=M-1
         M3(M)=M+1
         END DO Each element in turn
       END IF
!
       END DO Each segment in turn
!
!  ADJACENT ELEMENTS FOR END ELEMENTS OF THE BOUNDARY.
       M1(MMINB)=MMAXB
       M3(MMAXB)=MMINB
       END DO Each boundary in turn
       NEEND=IEEND
       NNP=NEL*2
!
!  DETERMINE THE MAXIMUM DIMENSION OF THE SOLUTION DOMAIN.
       MAXL=0.
       Each node in turn: DO I=1,NNP
       Each other node in turn: DO J=1,NNP
       DIST=SQRT((XNODE(I)-XNODE(J))**2+(YNODE(I)-YNODE(J))**2)
       IF(DIST > MAXL) MAXL=DIST
       END DO Each other node in turn
       END DO Each node in turn
!
       RETURN
       END SUBROUTINE MESHQ
```

The number of boundaries is first read into NBOUND. If there is more than one boundary the order of numbering is arbitrary. Then, for each boundary in turn, the number of segments forming the boundary is read into NSEGB, followed by the global co-ordinates of the segment end points taken consecutively into XSEND and YSEND. The starting point for defining segments on the boundary is arbitrary, but then the order of numbering must be such as to keep the domain on the left: anticlockwise for the outer boundary surrounding the domain, clockwise for a boundary defining a hole in the domain. Note that the first end point, which is an end of both the first and last segment, must only be defined once: the number of end points on a boundary is equal to the number of segments, and the program makes the connection to close the boundary.

Then, for each segment of the current boundary in turn, its radius of curvature, the number of elements on the segment, and the ratio between the lengths of successive elements on the segment are read into RSEG, NELSEG and RATSEG, respectively. A zero value of RSEG indicates a straight segment, while the sign of a non-zero value of RSEG indicates whether the part of the boundary defined by the segment is convex or concave. A positive radius indicates convex, with the centre of curvature inside the domain, while a negative value indicates concave, with the centre outside the domain. A unit value of RATSEG indicates a uniform distribution of elements along the segment, whether it be straight or curved. A value greater than one requires the lengths of successive elements to increase in the direction of segment end numbering, a value less than one requires a similar decrease.



The subprogram uses a number of housekeeping variables to keep track of various totals: IEEND stores the current element end point number, array NELB the number of elements on each boundary, and NEL the overall total number of elements. NSEGTOT stores the total number of segments, while MMINB and MMAXB contain the minimum and maximum element numbers (that is, first and last) on the current boundary. Similarly, ISEGMIN and ISEGMAX store the numbers of the first and last segments on the current boundary (the segment numbering runs sequentially from one boundary to the next), while MFIRST and MLAST do the same for elements on the current segment. Note that wherever possible tests are applied (and warning messages given) to check that array dimensions are not going to be exceeded. It is also worth noting that arrays concerned with segments are given the same dimensions (500, in module SHAREDDATA2PQ) as those for elements. This is to give the user the freedom to define the geometry and boundary condition of each element individually, by making each segment contain only one element. Arrays associated with element ends are actually given dimensions of 510. This is because due to the way element end point co-ordinates are defined it is convenient to have the last point on a boundary distinct from the first point, whose co-ordinates it shares. This means that the number of element end points on a boundary is one more than the number of elements, and there could be as many as 10 boundaries.

The subprogram stores the co-ordinates of the first and last element end points on the current segment and then defines the element end point co-ordinates, storing in XEEND and YEEND. Note that the segment number associated with a particular element end point is stored temporarily in array ISEGEND. Once the element data have been generated this information is transferred from ISEGEND to ISEGELEM, the segment number associated with each element. This information will be required later in the final output of results, when the segment number for each element is required for calculating potential flows across the various segments.

For a straight segment the element end point co-ordinates, whether for elements of uniform length or varying in length by a constant ratio, are generated exactly as described in Section 3.1.3 for the constant element program. The first and third nodes of an element are the element ends, and the second node is at the middle of the straight line joining these ends. Element node numbers are stored in the two-dimensional array NODE, the first subscript indicating the element number, the second subscript the number of the node within the element (first, second or third). The global co-ordinates of the nodes are stored in XNODE and YNODE.

For a segment in the form of a circular arc, the element data are again generated exactly as described in Section 3.1.3, with the first and third nodes of an element taken as the element ends. The second node is at the centre of the part of the segment arc connecting the end nodes. The numbers of the elements immediately adjacent to a particular element, whether straight or curved, need to be known, and are stored in arrays M1 for the element sharing the first node, and M3 for the element sharing the third node. Care has to be taken with the elements which meet at the first node on any boundary, the adjacent elements there being each other. The reason for storing this information is that it will be needed when dealing with adjacent elements having different boundary conditions applied.

Finally, the maximum dimension (MAXL) of the solution domain is found as the maximum distance between any pair of nodal points, for subsequent use in scaling as described in Section 2.6.

4.1.4 Mesh data output subprogram

The subprogram MSHOUT serves to write out the geometric data for the mesh to file MESHRES. Following the number of elements and number of nodes, the co-ordinates of the nodes are written out. Then the node numbers of the elements, two sets to a line.

```
      SUBROUTINE  MSHOUT
!
!  SUBPROGRAM TO WRITE OUT THE MESH DATA.
!
      USE SHAREDDATA2PQ
!
!  OUTPUT THE NUMBERS OF ELEMENTS AND NODES, ALSO THE NODAL
!  COORDINATES.
      WRITE(7,71) NEL,NNP,(I,XNODE(I),YNODE(I),I=1,NNP)
 71   FORMAT(/ "GEOMETRIC DATA FOR THE MESH" //
     &   10X,"NUMBER OF ELEMENTS =",I6 //
     &   10X,"NUMBER OF NODAL POINTS =",I6 //
     &       "COORDINATES OF THE NODAL POINTS"//
     &       2("     I      X           Y     ") /
     &       2(I6,2E12.4))
!
!  OUTPUT THE ELEMENT NODE NUMBERS.
      WRITE(7,72) (M,(NODE(M,IN),IN=1,3),M=1,NEL)
 72   FORMAT(/ 'ELEMENT NODE NUMBERS' //
     &       1X,2(10X,'M   ND1  ND2  ND3')/ (2(7X,4I5)))
!
!  SCALE THE NODAL POINT COORDINATES.
      Each node in turn: DO I=1,NNP
      XNODE(I)=XNODE(I)/MAXL
      YNODE(I)=YNODE(I)/MAXL
      END DO Each node in turn
!
      RETURN
      END SUBROUTINE MSHOUT
```

Finally, the co-ordinates of the nodes are scaled using the maximum dimension of the domain.

4.1.5 Subprogram for applying the boundary conditions

The subprogram BCS serves to apply boundary conditions of either the prescribed potential, prescribed potential gradient, or mixed (linear relationship between potential and potential gradient) types. As already indicated, it is assumed that each segment of elements has a uniform boundary condition applied to it, which is an important consideration when defining the segments. The total numbers of segments subject to the three types of condition are first read into variables NBCP, NBCD and NBCM, respectively. In the case of prescribed normal derivative (gradient) it is only necessary to include those segments subject to non-zero gradient. Storage arrays including ALPHA and BETA for the mixed boundary condition are set to zero for each nodal point. Array IBCE stores the type of boundary condition (1, 2 or 3 for three types) for each element. In the absence of other information, the condition for an element is assumed to be zero potential gradient. This is perhaps the most common single condition encountered in practice, for example on a line of symmetry, a thermally insulated boundary in a heat conduction problem, or an impermeable boundary in a fluid mechanics problem. The default boundary condition type is therefore set as 2, with the corresponding zero value of gradient for the elements being stored in array STORE.

```
      SUBROUTINE  BCS
!
!  SUBPROGRAM TO INPUT, PROCESS AND OUTPUT THE BOUNDARY CONDITIONS.
!
      USE SHAREDDATA2PQ
!
!  INPUT THE NUMBERS OF SEGMENTS SUBJECT TO EACH TYPE OF BOUNDARY
!  CONDITION.
!     NBCP - PRESCRIBED POTENTIAL.
!     NBCD - NON-ZERO PRESCRIBED NORMAL DERIVATIVE OF POTENTIAL.
!     NBCM - MIXED BOUNDARY CONDITION.
!  ANY SEGMENT NOT INCLUDED IS ASSUMED TO BE SUBJECT TO A ZERO
!  NORMAL DERIVATIVE OF POTENTIAL.
!
      READ(5,*) NBCP,NBCD,NBCM
!
!  TEST THESE BOUNDARY CONDITION NUMBERS.
      NBCT=NBCP+NBCD+NBCM
      IF(NBCP < 0 .OR. NBCP > MAXNEL .OR. NBCD < 0 .OR. NBCD > MAXNEL
     &            .OR. NBCM < 0 .OR. NBCM > MAXNEL .OR. NBCT < 0 .OR.
     &                 NBCT > MAXNEL) THEN
        WRITE(6,61) NBCP,NBCD,NBCM,NBCT,MAXNEL
 61     FORMAT(/ "NBCP =",I6,3X,"NBCD =",I6,3X,"NBCM =",I6
     &         / "NBCT =",I6,3X,"OUTSIDE PERMITTED RANGE 0 TO",I6)
      STOP
      END IF
```

```
!
!  INITIALISE THE BOUNDARY CONDITION STORAGE ARRAYS.
      Each element in turn: DO M=1,NEL
      IBCE(M)=2
      STORE(M)=0.
      ALPHA(M)=0.
      BETA(M)=0.
      END DO Each element in turn
!
!  INPUT, STORE AND OUTPUT THE PRESCRIBED POTENTIAL BOUNDARY CONDITIONS.
      IF(NBCP > 0) THEN
        READ(5,*) (ISEGBC(IBCP),PSISEG(IBCP),IBCP=1,NBCP)
        WRITE(6,62)
 62     FORMAT(/ "PRESCRIBED POTENTIAL BOUNDARY CONDITIONS")
        Each segment with prescribed potential: DO IBCP=1,NBCP
        ISEG=ISEGBC(IBCP)
        IF(ISEG < 1 .OR. ISEG > NSEGTOT) THEN
          WRITE(6,63) ISEG,NSEGTOT
 63       FORMAT(/ "ISEG = ",I6,2X,"OUTSIDE PERMITTED RANGE 1 TO",I6)
          STOP
          END IF
          Each element on current segment: DO M=MFIRST(ISEG),MLAST(ISEG)
          IBCE(M)=1
          STORE(M)=PSISEG(IBCP)
          END DO Each element on current segment
!
          WRITE(6,64) PSISEG(IBCP),MFIRST(ISEG),MLAST(ISEG)
 64       FORMAT(/ "POTENTIAL =",E12.4,5X,"ON ELEMENTS ",I6,5X,"TO ",I6)
          END DO Each segment with prescribed potential
        END IF
!
!  INPUT, STORE AND OUTPUT THE PRESCRIBED POTENTIAL GRADIENT BOUNDARY
!  CONDITIONS.
        IF(NBCD > 0) THEN
          READ(5,*) (ISEGBC(IBCD),DPSISEG(IBCD),IBCD=1,NBCD)
          WRITE(6,65)
 65       FORMAT(/ "PRESCRIBED POTENTIAL GRADIENT BOUNDARY CONDITIONS")
          Each segment with prescribed potential gradient: DO IBCD=1,NBCD
          ISEG=ISEGBC(IBCD)
          IF(ISEG < 1 .OR. ISEG > NSEGTOT) THEN
            WRITE(6,63) ISEG,NSEGTOT
            STOP
          END IF
          Each element on current segment: DO M=MFIRST(ISEG),MLAST(ISEG)
          IBCE(M)=2
          STORE(M)=DPSISEG(IBCD)
          END DO Each element on current segment
!
          WRITE(6,66) DPSISEG(IBCD),MFIRST(ISEG),MLAST(ISEG)
 66       FORMAT(/ "GRADIENT =",E12.4,5X,"ON ELEMENTS ",I6,5X,"TO ",I6)
          END DO Each segment with prescribed potential gradient
        END IF
!
!  INPUT, STORE AND OUTPUT THE MIXED BOUNDARY CONDITIONS.
!  ASSUMED FORM - DPSI=ALPHA*PSI+BETA.
        IF(NBCM > 0) THEN
          READ(5,*)(ISEGBC(IBCM),ALPHASEG(IBCM),BETASEG(IBCM),IBCM=1,NBCM)
          WRITE(6,67)
 67       FORMAT(/ "PRESCRIBED MIXED BOUNDARY CONDITIONS")
          Each segment with prescribed mixed condition: DO IBCM=1,NBCM
          ISEG=ISEGBC(IBCM)
          IF(ISEG < 1 .OR. ISEG > NSEGTOT) THEN
            WRITE(6,63) ISEG,NSEGTOT
            STOP
          END IF
          Each element on current segment: DO M=MFIRST(ISEG),MLAST(ISEG)
          IBCE(M)=3
          ALPHA(M)=ALPHASEG(IBCM)
          BETA(M)=BETASEG(IBCM)
          Each element node in turn: DO IN=1,3
          I=NODE(M,IN)
          ALPHAN(I)=ALPHA(M)
          BETAN(I)=BETA(M)
          END DO Each element node in turn
          END DO Each element on current segment
!
          WRITE(6,68)ALPHASEG(IBCM),BETASEG(IBCM),MFIRST(ISEG),MLAST(ISEG)
 68       FORMAT(/ "ALPHA =",E12.4,2X,"BETA =",E12.4,2X,
     &              "ON ELEMENTS ",I3,2X,"TO ",I3)
          END DO Each segment with prescribed mixed condition
        END IF
!
!  ASSEMBLE THE VECTOR OF KNOWN VARIABLES, SCALING ANY PRESCRIBED GRADIENTS.
        Each node in turn: DO I=1,NNP
        DPSI(I)=0.
        END DO Each node in turn
        Each element in turn: DO M=1,NEL
        Each element node in turn: DO IN=1,3
        I=NODE(M,IN)
        IBCN(I)=IBCE(M)
!
!  CHECK THE CONDITION APPLIED TO THE ADJACENT ELEMENT FOR AN ELEMENT END
!  NODE.  IF THE CONDITION IS PRESCRIBED POTENTIAL BUT THE EXISTING
!  CONDITION AT THE NODE IS NOT, THEN IMPOSE PRESCRIBED POTENTIAL.
        MADJ=M
        IF(IN == 1) MADJ=M1(M)
        IF(IN == 3) MADJ=M3(M)
        IF(IBCE(MADJ) == 1 .AND. IBCN(I) /= 1) IBCN(I)=1
!
```

```
!   STORE KNOWN VARIABLE.
        IF(IBCN(I) == 1) THEN
          IF(IBCE(M) == 1) THEN
            IF(DPSI(I) ==  0.) DPSI(I)=STORE(M)
            IF(DPSI(I) /=  0.) DPSI(I)=0.5*(DPSI(I)+STORE(M))
          END IF
        END IF
        IF(IBCE(M) == 2 .AND. IBCN(I) == 2) DPSI(I)=STORE(M)*MAXL
        IF(IBCE(M) == 3 .AND. IBCN(I) == 3) DPSI(I)=BETA(M)*MAXL
        END DO Each element node in turn
        END DO Each element in turn
!
        RETURN
        END SUBROUTINE BCS
```

For each segment over which potential is prescribed, the segment number and value of potential are read into ISEGBC and PSISEG, respectively. Taking each of these segments in turn, the elements have their boundary condition type set to 1 (in array IBCE), and the value of potential is transferred temporarily to array STORE. The prescribed potential and numbers of the elements to which it is applied are then written out.

For each segment over which potential gradient is prescribed, the segment number and value of potential gradient are read into ISEGBC and DPSISEG, respectively. Taking each of these segments in turn, the elements have their boundary condition type set to 2 (in array IBCE), and the value of potential gradient is transferred temporarily to array STORE. The prescribed potential gradient and numbers of the elements to which it is applied are then written out.

For each segment over which mixed boundary conditions prescribed, the segment number and values of the constants α and β (in Equation 2.32) are read into ISEGBC, ALPHASEG and BETASEG, respectively. Taking each of these segments in turn, the elements have their boundary condition type set to 3 (in array IBCE), and the values of boundary condition constants are transferred to element arrays ALPHA and BETA. Nodal point values of α and β are stored in arrays ALPHAN and BETAN. The constants and numbers of the elements to which the mixed boundary condition is applied are then written out.

For each node in turn, the array DPSI is made to contain the known variable values: according to Equation 2.30 this is typically potential gradient (hence the use of array DPSI). In the more general form of Equation 2.31, however, it is either potential or potential gradient according to the boundary conditions. If potential is prescribed its value is transferred to DPSI from STORE. If potential gradient is prescribed, its value is similarly transferred from STORE and multiplied by MAXL, the maximum domain dimension. While mesh dimensions have already been scaled, gradient values have not. Potential is not affected by scaling because it does not involve dimensions of length. Potential gradients have the dimension of potential divided by length, so in order to be scaled must be multiplied by the length scale. If a mixed boundary condition is prescribed, the (unscaled) value of β is transferred from STORE and multiplied by MAXL as the known variable (potential gradient) contribution, as described in connection with Equation 2.32.

The last paragraph describes how boundary conditions determine what known variable information is stored for the majority of nodes. The only exceptions occur at the ends of segments, which may correspond to physical corners in the problem or may correspond only to changes in boundary conditions. Although there are three possible types of boundary condition: prescribed potential, prescribed potential gradient and mixed boundary conditions, the important distinction is between potential (prescribed potential) and gradient (prescribed potential gradient or mixed) types. It is only with the potential type that the known potential has to be moved from the left hand side to the right hand side of Equations 2.30 to give Equations 2.31. A node at which there is a change from potential to gradient boundary conditions, whether at a physical corner or not, presents a difficulty: which of the two boundary conditions takes precedence. In all cases the prescribed potential is chosen, which is why in the program there is a test for potential boundary condition on the current element, but gradient boundary condition at the current node (which can arise from the way the boundary conditions have been applied to the elements). The potential boundary condition type is imposed by setting the relevant value in array IBCN to 1.

If at a node linking two segments there is a change, not in boundary condition type (potential) but from one value of potential to another, it is appropriate to assign an average of the two values to the node and store this as the known variable for that node. This is achieved in the program by testing every node to see whether both the node and element boundary condition is of the potential type. If it is, the known variable is taken as the stored value of prescribed potential for the element if the known variable value is zero. This will apply to all element second nodes, and to all element end nodes which have not been tested in previous elements. If the known variable value is not zero, which can only occur if prescribed potential is applied to the immediately adjacent element and that conditions at the nodes of the element have already been tested, then the average of the present element value and the stored value is re-stored. This works both for adjacent elements with different values of potential and for the much more common case of adjacent elements with the same value prescribed: the common value is re-stored.

This averaging process is really a detail to give a sensible value of potential in the output of the final results. What is really important is that the prescribed potential is applied element by element in forming the $[b]$ column vector in Equations 2.31, so that each of the adjacent elements contributes according to its own prescribed potential. The same is true of adjacent elements with gradient boundary conditions: each should contribute to $[b]$ according to its own value of prescribed potential or its own values of mixed condition parameters. This will be explained further in connection with subprogram FRMTRX.

4.1.6 Subprogram for defining shape functions

Subprogram SHAPE serves to define the shape functions and derivatives of shape functions at relevant values of the intrinsic co-ordinate of any quadratic element. The relevant values of ξ are primarily those at the points where functions have to be computed in order to apply Gaussian quadrature as described in Appendix A. These are referred to as the Gauss points. If the problem is of the Poisson type, shape function derivatives have also to be determined for the nodal point values of ξ. This is because the components of the unit normal vectors at the nodes are required to compute the effects of the particular integral function on the boundary conditions.

```
    SUBROUTINE SHAPE
!
! SUBPROGRAM TO EVALUATE AND STORE VALUES OF THE SHAPE FUNCTIONS AND
! THEIR DERIVATIVES, AT THE GAUSS POINTS AND NODES.
!
    USE SHAREDDATA2PQ
!
! STORE APPROPRIATE COORDINATES AND WEIGHT FACTORS FOR NORMAL GAUSSIAN
! QUADRATURE IN ARRAYS XG AND CG, ALSO THOSE FOR LOGARITHMIC FUNCTION
! INTEGRATION IN XGL AND CGL.
```

```
      NGAUSS=8
      ZG(1)=-0.9602898564
      ZG(2)=-0.7966664774
      ZG(3)=-0.5255324099
      ZG(4)=-0.1834346424
      ZG(5)=-ZG(4)
      ZG(6)=-ZG(3)
      ZG(7)=-ZG(2)
      ZG(8)=-ZG(1)
      WG(1)=0.1012285362
      WG(2)=0.2223810344
      WG(3)=0.3137066458
      WG(4)=0.3626837833
      WG(5)=WG(4)
      WG(6)=WG(3)
      WG(7)=WG(2)
      WG(8)=WG(1)
      EGL(1)=0.013320244
      EGL(2)=0.079750429
      EGL(3)=0.197871029
```

```
    EGL(4)=0.354153994
    EGL(5)=0.529458575
    EGL(6)=0.701814530
    EGL(7)=0.849379320
    EGL(8)=0.953326450
    WGL(1)=0.164416605
    WGL(2)=0.237525610
    WGL(3)=0.226841984
    WGL(4)=0.175754079
    WGL(5)=0.112924030
    WGL(6)=0.057872211
    WGL(7)=0.020979074
    WGL(8)=0.003686407
  !
  ! NORMAL GAUSSIAN QUADRATURE.
    For each Gauss point in turn: DO IGAUSS=1,NGAUSS
    ZETA=ZG(IGAUSS)
    SF(1,IGAUSS)=0.5*ZETA*(ZETA-1.)
    SF(2,IGAUSS)=1.-ZETA**2
    SF(3,IGAUSS)=0.5*ZETA*(ZETA+1.)
    SD(1,IGAUSS)=ZETA-0.5
    SD(2,IGAUSS)=-2.*ZETA
    SD(3,IGAUSS)=ZETA+0.5
    END DO For each Gauss point in turn
  !
  ! FOUR CASES OF LOGARITHMIC GAUSSIAN QUADRATURE TO CONSIDER.
  !   IC=1 - INTEGRATION OVER WHOLE ELEMENT FROM FIRST TO THIRD NODE.
  !   IC=2 - INTEGRATION OVER HALF ELEMENT FROM SECOND TO THIRD NODE.
  !   IC=3 - INTEGRATION OVER HALF ELEMENT FROM SECOND TO FIRST NODE.
  !   IC=4 - INTEGRATION OVER WHOLE ELEMENT FROM THIRD TO FIRST NODE.
  !
    For each case in turn: DO IC=1,4
    For each Gauss point in turn: DO IGAUSS=1,NGAUSS
    ETA=EGL(IGAUSS)
    IF(IC == 1) ZETA=2.*ETA-1.
    IF(IC == 2) ZETA=ETA
    IF(IC == 3) ZETA=-ETA
    IF(IC == 4) ZETA=1.-2.*ETA
    SFL(IC,1,IGAUSS)=0.5*ZETA*(ZETA-1.)
    SFL(IC,2,IGAUSS)=1.-ZETA**2
    SFL(IC,3,IGAUSS)=0.5*ZETA*(ZETA+1.)
    SDL(IC,1,IGAUSS)=ZETA-0.5
    SDL(IC,2,IGAUSS)=-2.*ZETA
    SDL(IC,3,IGAUSS)=ZETA+0.5
    END DO For each Gauss point in turn
    END DO For each case in turn
!
! SHAPE FUNCTION DERIVATIVES AT THE NODES, STORED AS THOUGH THEY
! ARE FOR GAUSS POINTS NUMBERED 11, 12 AND 13.
    Each element node in turn: DO IGAUSS=11,13
    IF(IGAUSS == 11) ZETA=-1.
    IF(IGAUSS == 12) ZETA=0.
    IF(IGAUSS == 13) ZETA=1.
    SD(1,IGAUSS)=ZETA-0.5
    SD(2,IGAUSS)=-2.*ZETA
    SD(3,IGAUSS)=ZETA+0.5
    END DO Each element node in turn
!
    RETURN
    END SUBROUTINE SHAPE
```

The values of ξ and the corresponding weighting factors ω for normal Gaussian quadrature, as defined in Equations A.3, are first stored in arrays ZG and WG, respectively. For Gaussian quadrature applied to a function involving the singular logarithmic function, as defined in Equation A.4 and referred to here as logarithmic Gaussian quadrature, the values of intrinsic co-ordinate η and corresponding weights ω, as defined in Equations A.5, are stored in arrays EGL and WGL, respectively.

The shape functions $N_1(\xi)$, $N_2(\xi)$, and $N_3(\xi)$ are defined in Equations 2.50, 2.51 and 2.52. In subprogram SHAPE they are stored in array SF for each of the Gauss point values of ξ. SF is a two-dimensional array, the first subscript defining the shape function number, associated with one of the three nodes of an element, and the second subscript defining the Gauss point number. Similarly, the derivatives of the shape functions, defined in Equations 2.58, are stored in array SD. For normal Gaussian quadrature, the relevant values of ξ are the corresponding values of ZG.

For logarithmic Gaussian quadrature there are four different situations to be accommodated. The first is where logarithmic quadrature is to be applied to the whole element, treating the first node as the origin for η. The second is for the half element between the second and third nodes, treating the second node as the origin. The third is for the half element between the second and first nodes, again treating the second node as the origin. The fourth is for the whole element, treating the third node as the origin. For these four cases, the relationships between intrinsic co-ordinates η and ξ are given by Equations 2.71, 2.82, 2.89 and 2.94, respectively. Relevant values of η (ETA) are taken from EGL and converted into ξ , and the computed shape functions and derivatives stored in arrays SFL and SDL. Both of these arrays are three-dimensional, the three subscripts indicating case number (1 to 4), shape function number and Gauss point number.

Finally, the shape function derivatives are computed for the nodal points of an element, where the values of ξ are -1, 0 and +1 for the first, second and third nodes. For convenience of storage in array SD, the nodes are considered to be the 11th, 12th and 13th Gauss points.

4.1.7 Subprogram for Poisson type problems

In subprogram POISSON the particular integral contributions to potentials and potential gradients at the nodes are first set to zero. If the problem is of the Laplace type (indicated by a zero value of f_1 in Equation 2.1, stored in F1 in the program) control returns to the calling program. If the problem is of the Poisson type, then it is first necessary to compute the components of the unit normal vectors at the nodes (for use in Equation 2.109). In fact, it is only for Poisson type problems that the normals at the nodes (in addition to the Gauss points on the elements) are required.

The method for finding normals is described in Section 2.5, and is part of the process of defining the Jacobian of transformation from global to local intrinsic co-ordinates which is carried out in subprogram JACOBI. The components of the unit normal at a point, in this case one of the nodal points (treated for storage purposes as effectively a Gauss point), are returned from JACOBI in variables UNGX and UNGY. They are then stored in the two-dimensional arrays UNMX and UNMY. These are element-oriented arrays with subscripts indicating element number and element node number. This means that at a node where the boundary is not smooth, data for two different normals are stored, associated with the two elements which meet at that node.

```
    SUBROUTINE POISSON
!
! MODIFY BOUNDARY CONDITIONS IF GOVERNING EQUATION IS POISSON TYPE.
!
    USE SHAREDDATA2PQ
!
! INITIALISE THE PARTICULAR INTEGRAL ARRAYS.
    Each node in turn: DO I=1,NNP
    PSIPI(I)=0.
```

```
      DPSIPI(I)=0.
      END DO Each node in turn
      Each element in turn: DO M=1,NEL
      Each element node in turn: DO IN=1,3
      DPSIPIM(M,IN)=0.
      END DO Each element node in turn
      END DO Each element in turn
    !
    ! UNIT NORMALS AT THE NODES ARE REQUIRED FOR PARTICULAR INTEGRAL
    ! DERIVATIVES THERE.
    !
    ! FIRST FIND THE NORMALS AT THE ELEMENT NODES.
      Each element in turn: DO M=1,NEL
      Each element node in turn: DO IN=1,3
      IGAUSS=IN+10
      IT=1
      IC=1
      CALL JACOBI(M,IGAUSS,IT,IC)
      UNMX(M,IN)=UNGX
      UNMY(M,IN)=UNGY
```

```
    END DO Each element node in turn
    END DO Each element in turn
!
! MERGE ELEMENT NODE NORMALS TO FIND NORMALS AT THE NODES.
    Each node in turn: DO I=1,NNP
    UNX(I)=0.
    UNY(I)=0.
    END DO Each node in turn
    Each element in turn: DO M=1,NEL
!
! FIRST NODE.
    I=NODE(M,1)
    MADJ=M1(M)
!
! ICASE=1 BOTH CURRENT AND ADJACENT ELEMENT HAVE EITHER PRESCRIBED
!         POTENTIAL OR PRESCRIBED GRADIENT/MIXED BOUNDARY CONDITIONS.
! ICASE=2 CURRENT ELEMENT HAS PRESCRIBED POTENTIAL, ADJACENT HAS
!         PRESCIBED GRADIENT/MIXED.
! ICASE=3 CURRENT ELEMENT HAS PRESCRIBED GRADIENT/MIXED CONDITION,
!         ADJACENT HAS PRESCRIBED POTENTIAL.
    IF(IBCE(M) == 1 .AND. IBCE(MADJ) == 1) ICASE=1
    IF(IBCE(M) /= 1 .AND. IBCE(MADJ) /= 1) ICASE=1
    IF(IBCE(M) == 1 .AND. IBCE(MADJ) /= 1) ICASE=2
    IF(IBCE(M) /= 1 .AND. IBCE(MADJ) == 1) ICASE=3
    IF(ICASE == 1) THEN
    UNX(I)=UNX(I)+UNMX(M,1)
    UNY(I)=UNY(I)+UNMY(M,1)
    END IF
    IF(ICASE == 2) THEN
    UNX(I)=UNMX(M,1)
    UNY(I)=UNMY(M,1)
    END IF
!
! SECOND NODE.
    I=NODE(M,2)
    UNX(I)=UNMX(M,2)
    UNY(I)=UNMY(M,2)
!
! THIRD NODE.
    I=NODE(M,3)
    MADJ=M3(M)
    IF(IBCE(M) /= 1 .AND. IBCE(MADJ) /= 1) ICASE=1
    IF(IBCE(M) == 1 .AND. IBCE(MADJ) == 1) ICASE=1
    IF(IBCE(M) == 1 .AND. IBCE(MADJ) /= 1) ICASE=2
    IF(IBCE(M) /= 1 .AND. IBCE(MADJ) == 1) ICASE=3
    IF(ICASE == 1) THEN
    UNX(I)=UNX(I)+UNMX(M,3)
    UNY(I)=UNY(I)+UNMY(M,3)
    END IF
    IF(ICASE == 2) THEN
    UNX(I)=UNMX(M,3)
    UNY(I)=UNMY(M,3)
    END IF
    END DO Each element in turn
!
! FOR ELEMENT END NODES WITH ICASE=1, CONTRIBUTIONS TO THE NORMALS
! WILL HAVE COME FROM TWO ELEMENTS, EFFECTIVELY GIVING AVERAGE
! (VECTOR SUM) NORMALS, BUT WHICH MUST NOW BE REDUCED TO UNIT
! NORMALS.
    Each node in turn: DO I=1,NNP,2
    DENOM=SQRT(UNX(I)**2+UNY(I)**2)
    UNX(I)=UNX(I)/DENOM
    UNY(I)=UNY(I)/DENOM
    END DO Each node in turn
!
! FIND AND STORE THE POTENTIAL AND POTENTIAL GRADIENT AT EVERY NODE
! DUE TO THE PARTICULAR INTEGRAL.
    IF(F1 == 0.) RETURN
    XC=XC/MAXL
    YC=YC/MAXL
    F1=F1*MAXL**2
    Each node in turn: DO I=1,NNP
    XX=XNODE(I)-XC
    YY=YNODE(I)-YC
    PSIPI(I)=0.25*F1*(XX**2+YY**2)
    DPSIPI(I)=0.5*F1*(XX*UNX(I)+YY*UNY(I))
    END DO Each node in turn
!
! FIND AND STORE THE POTENTIAL GRADIENT AT EVERY NODE OF EVERY
! ELEMENT DUE TO THE PARTICULAR INTEGRAL.
    Each element in turn: DO M=1,NEL
    Each element node in turn: DO IN=1,3
    I=NODE(M,IN)
    XX=XNODE(I)-XC
    YY=YNODE(I)-YC
    DPSIPIM(M,IN)=0.5*F1*(XX*UNMX(M,IN)+YY*UNMY(M,IN))
    END DO Each element node in turn
    END DO Each element in turn
!
! SUBTRACT THE PARTICULAR INTEGRAL CONTRIBUTIONS FROM THE PRESCRIBED
! BOUNDARY CONDITIONS.
    Each node in turn: DO I=1,NNP
    IF(IBCN(I) == 1) DPSI(I)=DPSI(I)-PSIPI(I)
    IF(IBCN(I) == 2) DPSI(I)=DPSI(I)-DPSIPI(I)
    IF(IBCN(I) == 3) DPSI(I)=DPSI(I)-DPSIPI(I)+ALPHAN(I)*PSIPI(I)*MAXL
  END DO Each node in turn
!
    RETURN
    END SUBROUTINE POISSON
```

Next, the components of the unit normals at the nodes are computed and stored in arrays UNX and UNY. For element midside nodes this is a straightforward transfer from UNMX to UNX and UNMY to UNY. For element end nodes, possible differences between the normals in the adjacent elements have to be accommodated. Three cases need to be considered. Firstly, where both elements have either potential or gradient type boundary conditions prescribed. Secondly, where the current element has potential type, but the adjacent element has gradient type. Thirdly, where the current element has gradient type and the adjacent element has potential type. In the first case the normal at the node is taken as the average of those for the two elements. This is achieved by simply adding the components of the two unit normals to create the vector sum at the average angle to the global co-ordinates. A consequence of this is that at a node where the boundary should be smooth but the quadratic representation does not model the geometry exactly, for example on an arc of a circle, the true average normal direction will be used. In the second and third cases of different boundary condition types on the two adjacent elements, prescribed potential at the node is imposed, and the normal should be that associated with the element over which potential is prescribed.

First case type element end nodes now have averaged normals, but they are not unit normals. So all end nodes (odd-numbered nodes) have their normals reduced to unit normals, which has no effect on second or third case end nodes whose normals are already unit normals.

For every node, the values of the particular integral potential and potential gradient defined by Equations 2.108 and 2.109, in co-ordinates which have their origin at the point (XC, YC), are computed and stored in arrays PSIPI and DPSIPI. Note that the particular integral is defined in terms of scaled variables, so that the co-ordinates of the origin are first divided by MAXL, and f_1 is multiplied by the square of this length. From Equation 2.1, f_1 must have the dimensions of potential divided by length squared, because it is equated to second derivatives of potential. Particular integral potential gradients are also computed for every node of every element (with its own individual normal direction) and stored in the two-dimensional array DPSIPIM.

The final action in the subprogram is to subtract the scaled particular integral contribution from the scaled known variable for each node. In other words, for a type 1 boundary condition (prescribed potential) the potential due to the particular integral (Equation 2.108) is subtracted from the scaled potential defined by the actual boundary condition, which is already stored in the array DPSI. For a type 2 boundary condition (prescribed potential gradient) the potential gradient due to the particular integral (Equation 2.109) is subtracted from the (scaled) potential gradient defined by the actual boundary condition, which is already stored in the array DPSI. The type 3 (mixed) boundary condition is a little more complicated, but is dealt with exactly as described in Section 3.1.6 for constant elements, in particular using Equation 3.21.

4.1.8 Subprogram to form the coefficient matrix

Subprogram FRMTRX forms the coefficient matrix and right hand side vector of Equations 2.31, and stores them as an extended matrix in array A. The extended matrix is simply $[A^*]$ with an extra column to contain column vector $[b]$. Each node in turn is treated as point P, the source point for the fundamental solution, then integration is carried out over every element in turn. This integration involves the knowns and unknowns at each of the three nodes of every element. Node counters I and J are used for P and the current element node, respectively, so that I is the equation counter and numbers rows in $[A^*]$, while J numbers columns. The actual integrations of the kernel function products are carried out in subprogram INTKER which is called by FRMTRX. What INTKER does is to compute the integrals

$$\int_{-1}^{+1} \frac{\mathrm{d}}{\mathrm{d}n(Q)}\left\{\ln\left[\frac{1}{r(P,Q)}\right]\right\} N_c(\xi)\, J(\xi)\, \mathrm{d}\xi \text{ and } \int_{-1}^{+1} \ln\left[\frac{1}{r(P,Q)}\right] N_c(\xi)\, J(\xi)\, \mathrm{d}\xi \tag{4.1}$$

associated with the *c*th node of the current element m, and which are required in Equation 2.63. If the knowns and unknowns at this node are potential gradient and potential, then these two quantities will contribute to the $[A]$ and $[B]$ matrices, respectively. Initially they are stored (in subprogram INTKER) in the two-dimensional arrays AROW and BROW as the coefficients that will form the current row of $[A]$ and the current row of $[B]$, but with the contributions from every node of every element kept distinct.

```
          SUBROUTINE  FRMTRX
!
!  SUBPROGRAM TO FORM THE COEFFICIENT MATRIX AND RIGHT HAND SIDE VECTOR,
!  MODIFIED TO SUIT THE BOUNDARY CONDITIONS.
!
      USE SHAREDDATA2PQ
!
!  DEFINE THE NUMBER OF COLUMNS IN THE EXTENDED COEFFICIENT MATRIX A.
      JMAX=NNP+1
!
!  FORM THE COEFFICIENT MATRIX A, AND THE RIGHT HAND SIDE VECTOR B*DPSI.
      Take each node in turn as P: DO I=1,NNP
!
!  INITIALISE CURRENT ROW OF MATRIX A.
      Each coefficient in turn: DO J=1,JMAX
      A(I,J)=0.
      END DO Each coefficient in turn
!
!  INITIALISE ELEMENT CONTRIBUTIONS TO CURRENT ROWS OF A AND B MATRICES.
      Each element in turn: DO M=1,NEL
      Each element node in turn: DO IN=1,3
      AROW(M,IN)=0.
      BROW(M,IN)=0.
      END DO Each element node in turn
      END DO Each element in turn
!
!  SET UP THE LOOP TO INTEGRATE OVER EACH ELEMENT IN TURN.
      Each element in turn: DO M=1,NEL
!
!  INTEGRATE THE KERNEL PRODUCTS OVER THE CURRENT ELEMENT.
      CALL  INTKER(I,M)
      END DO Each element in turn
!
!  EVALUATE THE DIAGONAL COEFFICIENT OF MATRIX A.
      AII=0.
      Each element in turn: DO M=1,NEL
      Each element node in turn: DO IN=1,3
      AII=AII-AROW(M,IN)
      END DO Each element node in turn
      END DO Each element in turn
      IF(IBCN(I) /= 1) A(I,I)=AII
!
```

YOU THINK.
YOU CAN WORK
AT RMB
RAND MERCHANT BANK
A division of FirstRand Bank Limited
Traditional values. Innovative ideas.

```
!  INITIALISE THE B*DPSI VECTOR COEFFICIENT.
      BDPSI=0.
      IF(IBCN(I) == 1) BDPSI=-AII*DPSI(I)
!
!  APPLY THE BOUNDARY CONDITIONS TO THE CURRENT ROWS OF A AND B, BY
!  CONSIDERING EACH ELEMENT IN TURN.
      Each element in turn: DO M=1,NEL
      Each element node in turn: DO IN=1,3
      J=NODE(M,IN)
!
!  IF POTENTIAL IS PRESCRIBED OVER THE ELEMENT, INTERCHANGE THE A AND
!  B COEFFICIENTS.
      IF(IBCE(M) == 1) THEN
        A(I,J)=A(I,J)-BROW(M,IN)
        BDPSI=BDPSI-AROW(M,IN)*(STORE(M)-PSIPI(J))
      END IF
!
!  IF POTENTIAL GRADIENT IS PRESCRIBED OVER THE ELEMENT (WHICH MAY
!  INCLUDE A CORNER NODE WITH MIXED BOUNDARY CONDITION), STORE THE A
!  MATRIX COEFFICIENTS, EXCEPT AT A CORNER NODE WHERE PRESCRIBED
!  POTENTIAL HAS BEEN IMPOSED.
      IF(IBCE(M) == 2) THEN
        BDPSI=BDPSI+BROW(M,IN)*(STORE(M)*MAXL-DPSIPIM(M,IN))
        IF(IBCN(J) /= 1) A(I,J)=A(I,J)+AROW(M,IN)
        IF(IBCN(J) == 1) BDPSI=BDPSI-AROW(M,IN)*DPSI(J)
      END IF
!
!  IF BOUNDARY CONDITION OVER THE ELEMENT IS MIXED (WHICH MAY
!  INCLUDE A CORNER NODE WITH PRESCRIBED POTENTIAL GRADIENT),
!  MODIFY THE A MATRIX COEFFICIENTS, EXCEPT AT A CORNER NODE
!  WHERE PRESCRIBED POTENTIAL HAS BEEN IMPOSED.
      IF(IBCE(M) == 3) THEN
        DPSISTORE=ALPHA(M)*PSIPI(J)+BETA(M)
        DPSISTORE=DPSISTORE*MAXL-DPSIPIM(M,IN)
        BDPSI=BDPSI+BROW(M,IN)*DPSISTORE
        IF(IBCN(J) /= 1) THEN
          A(I,J)=A(I,J)+AROW(M,IN)-BROW(M,IN)*ALPHA(M)*MAXL
        END IF
        IF(IBCN(J) == 1) THEN
```

```
            BDPSI=BDPSI-AROW(M,IN)*DPSI(J)
            BDPSI=BDPSI+BROW(M,IN)*ALPHA(M)*DPSI(J)*MAXL
          END IF
        END IF
!
        END DO Each element node in turn
        END DO Each element in turn
!
!  STORE THE B*DPSI COEFFICIENT AS AN EXTENSION OF MATRIX A.
        A(I,JMAX)=BDPSI
        END DO Take each node in turn as P
!
        RETURN
        END SUBROUTINE FRMTRX
```

At the beginning of the subprogram the current row of array A and all the coefficients of arrays AROW and BROW are set to zero in anticipation of the assembly process. Then for each boundary element in turn subprogram INTKER is called to carry out the integrations. Before any boundary conditions are applied, the diagonal coefficient of the $[A]$ matrix is computed with the aid of Equation 2.66, noting that the A_{ij} coefficient for any element end node has two contributions, from the two elements which share it, stored in the relevant locations in AROW. The coefficient on the current row of the product of matrix $[B]$ with the vector of known variables is to be accumulated in BDPSI. This is set to zero initially in anticipation of the element contributions to be added to it. If the boundary condition at point P is of the potential type, the product of the diagonal coefficient of $[A]$ with the prescribed potential there is known and must be moved from the left hand side to the right hand side of the equation, and stored in BDPSI with its sign changed.

The boundary conditions applied to each of the elements in turn are now used to determine how the terms stored in AROW and BROW must be added to $[A]$ and $[B]$. It is perhaps helpful to consider the relevant fragments of the equation represented by the current rows of the $[A]$ and $[A]$ matrices, the fragments being for the INth node of the element numbered M as a result of the integration over the element

$$\ldots.+\text{AROW(M,IN)}\times\,\psi_j+\ldots. \quad = \quad \ldots.+\text{BROW(M,IN)}\times\left(\frac{\mathrm{d}\psi}{\mathrm{d}n}\right)_j+\ldots\ldots \tag{4.2}$$

where j is the number of the INth node of the element (J=NODE(M,IN) in programming terms). These parts of an equation are presented in a mixture of program and physical variables in an attempt to clarify what is going on. The values of potential and potential gradient in this equation are for the Laplace problem after subtraction of the particular integral contributions.

Potential gradient known over the element

If the potential gradient over the element is known and the potential unknown then the product BROW(M,IN)× $\left(\frac{\mathrm{d}\psi}{\mathrm{d}n}\right)_j$, which is a known quantity, is added to the right hand side vector whose coefficient for the current equation is BDPSI. Note that, irrespective of what is the known variable at the particular node, the gradient in this product is that prescribed for the element as a whole, if necessary subtracting the particular integral contribution at the node of the element, namely STORE(M)*MAXL-DPSIPIM(M,IN). Multiplication by the length MAXL is necessary to scale the stored value of gradient. Provided the potential at the particular node is the unknown, then on the left hand side of the equation the computed AROW(M,IN) is simply added to the $[A]$ matrix coefficient stored in A(I, J).

If the potential at the particular node is known, which implies that the node is an end node at a point where the boundary condition changes from potential type to gradient type, then the AROW(M,IN)× ψ_j term in Equation 4.2, where ψ_j is the value stored in DPSI(J), must be taken from the left hand side of the equation to the right hand side and subtracted from BDPSI.

Potential known over the element

If the potential over the element (and hence at all three nodes) is known and the potential gradient is unknown then the terms in Equation 4.2 have to be moved to the opposite sides

$$\ldots - \mathrm{BROW(M,IN)} \times \left(\frac{\mathrm{d}\psi}{\mathrm{d}n}\right)_j + \ldots\ldots \quad = \quad \ldots - \mathrm{AROW(M,IN)} \times \psi_j + \ldots \tag{4.3}$$

The product AROW(M,IN)×ψ_j, which is a known quantity, is subtracted from the right hand side vector whose coefficient for the current equation is BDPSI. Note that, irrespective of what is the known variable at the particular node, the potential in this product is that prescribed for the element as a whole, if necessary subtracting the particular integral contribution at the node of the element, namely STORE(M)–PSIPI(J). On the left hand side of the equation the computed BROW(M,IN) is simply subtracted from the $[A]$ matrix coefficient stored in A(I, J).

Mixed boundary condition over the element

If a mixed boundary condition is applied to the element then, from Equation 3.21, the gradient term in Equation 4.2 is

$$\left(\frac{\mathrm{d}\psi}{\mathrm{d}n}\right)_j = \left[\alpha_m(\psi_j + \psi_{PI}) + \beta_m\right]L - \left(\frac{\mathrm{d}\psi}{\mathrm{d}n}\right)_{PI} \tag{4.4}$$

where α_m and β_m are the boundary condition parameters for element numbered M, namely ALPHA(M) and BETA(M) in programming terms, L is the length scaling parameter MAXL, and ψ_{PI} and $\left(\frac{\mathrm{d}\psi}{\mathrm{d}n}\right)_{PI}$ are the (scaled) particular integral contributions at node j in the element, namely PSIPI(J) and DPSIPIM(M,IN). Irrespective of what is the known variable at the particular node the following part of this expression is known

$$[\alpha_m \psi_{PI} + \beta_m]L - \left(\frac{\mathrm{d}\psi}{\mathrm{d}n}\right)_{PI} \tag{4.5}$$

and is stored in DPSISTORE in the program. It is then multiplied by BROW(M,IN) according to Equation 4.2 and added to BDPSI.

If the potential at the particular node is the unknown, then the $\alpha_m \psi_j L$ term contains the unknown and it must be taken to the left hand side of the equation. Therefore, to the $[A]$ matrix coefficient stored in A(I, J) is added the expression

$$\text{AROW(M,IN)-BROW(M,IN)*ALPHA(M)*MAXL} \tag{4.6}$$

If the potential at the particular node is known, which implies that the node is an end node at a point where the boundary condition changes from potential type to gradient type, then the AROW(M,IN) × ψ_j term in Equation 4.2, where ψ_j is the value stored in DPSI(J), must be taken from the left hand side of the equation to the right hand side and subtracted from BDPSI. The $\alpha_m \psi_j L$ term in Equation 4.5 is now also known and is multiplied by BROW(M,IN) and added to BDPSI.

Once assembly of the linear equation for the current point *P* is complete, the right hand side vector coefficient accumulated in BDPSI is stored in the last column of extended matrix $[A]$. The assembly process is repeated for all nodes as source point *P*.

4.1.9 Subprogram for integrating kernel function products over an element

As already indicated (Equations 4.1), the purpose of subprogram INTKER is to compute the integrals involving the kernel functions

$$\int_{-1}^{+1} \frac{\mathrm{d}}{\mathrm{d}n(Q)} \left\{ \ln \left[\frac{1}{r(P,Q)} \right] \right\} N_c(\xi)\, J(\xi)\, \mathrm{d}\xi \text{ and } \int_{-1}^{+1} \ln \left[\frac{1}{r(P,Q)} \right] N_c(\xi)\, J(\xi)\, \mathrm{d}\xi$$

for a particular source point *P* (node numbered I) and element m, and which are required in Equation 2.63. The global co-ordinates of *P* are first stored in XP and YP. Then, for each element node in turn (numbered c in the above expressions, IN in the subprogram) the node number is stored as J.

```
      SUBROUTINE  INTKER(I,M)
!
!  SUBPROGRAM TO INTEGRATE THE KERNEL PRODUCTS FOR A PARTICULAR SOURCE
!  POINT P (INDICATED BY NODE NUMBER I) OVER A PARTICULAR ELEMENT
!  (INDICATED BY ARGUMENT M) BY GAUSSIAN QUADRATURE.
!
      USE SHAREDDATA2PQ
!
!  COORDINATES OF POINT P.
      XP=XNODE(I)
      YP=YNODE(I)
!
!  SET UP THE ELEMENT NODE LOOP.
      Each element node in turn: DO IN=1,3
      J=NODE(M,IN)
!
!  IF P IS NOT THE CURRENT ELEMENT NODE, USE NORMAL GAUSSIAN QUADRATURE.
      IF(I /= J) THEN
        Each Gauss point in turn: DO IGAUSS=1,NGAUSS
!
```

```
!  EVALUATE JACOBIAN AND UNIT NORMAL VECTOR COMPONENTS, ALSO THE KERNELS
!  AT THE PARTICULAR GAUSS POINT FOR NORMAL QUADRATURE OVER THE WHOLE
!  ELEMENT.
        IT=1
        IC=1
        CALL  JACOBI(M,IGAUSS,IT,IC)
        CALL  KERNEL(XP,YP,M,IGAUSS,AK,BK)
!
!  ACCUMULATE THE INTEGRALS.
        SFN=SF(IN,IGAUSS)
        AROW(M,IN)=AROW(M,IN)+WG(IGAUSS)*AK*SFN*JACOB
        BROW(M,IN)=BROW(M,IN)+WG(IGAUSS)*BK*SFN*JACOB
        END DO Each Gauss point in turn
      END IF
!
!  IF P IS THE CURRENT ELEMENT NODE, SOME LOGARITHMIC QUADRATURE
!  IS REQUIRED.
      IF(I == J) THEN
!
!  P AT THE FIRST NODE OF THE ELEMENT.
        IF(IN == 1) THEN
!
!  TERM INVOLVING NORMAL QUADRATURE.
          Each Gauss point in turn: DO IGAUSS=1,NGAUSS
          IT=1
          IC=1
          CALL  JACOBI(M,IGAUSS,IT,IC)
          CALL  KERN2(M,IN,IGAUSS,BK2)
          SFN=SF(IN,IGAUSS)
          BROW(M,IN)=BROW(M,IN)+WG(IGAUSS)*BK2*SFN*JACOB
          END DO Each Gauss point in turn
!
!  TERM INVOLVING LOGARITHMIC QUADRATURE.
          Each Gauss point in turn: DO IGAUSS=1,NGAUSS
          IT=2
          IC=1
          CALL  JACOBI(M,IGAUSS,IT,IC)
          SFN=SFL(IC,IN,IGAUSS)
          DZDE=2.
```

```
            BROW(M,IN)=BROW(M,IN)+WGL(IGAUSS)*SFN*JACOB*DZDE
            END DO Each Gauss point in turn
          END IF
!
!  P AT THE SECOND NODE OF THE ELEMENT.
          IF(IN == 2) THEN
!
!  TERMS INVOLVING NORMAL QUADRATURE.
            Each Gauss point in turn: DO IGAUSS=1,NGAUSS
            IT=1
            IC=1
            CALL  JACOBI(M,IGAUSS,IT,IC)
            CALL  KERN2(M,IN,IGAUSS,BK2)
            SFN=SF(IN,IGAUSS)
            BROW(M,IN)=BROW(M,IN)+WG(IGAUSS)*BK2*SFN*JACOB
            END DO Each Gauss point in turn
!
!  TERMS INVOLVING LOGARITHMIC QUADRATURE.
            Each Gauss point in turn: DO IGAUSS=1,NGAUSS
            IT=2
```

We believe that energy suppliers should be renewable, too. We are therefore looking for enthusiastic new colleagues with plenty of ideas who want to join RWE in changing the world. Visit us online to find out what we are offering and how we are working together to ensure the energy of the future.

Click on the ad to read more

```
          IC=2
          CALL  JACOBI(M,IGAUSS,IT,IC)
          SFN=SFL(IC,IN,IGAUSS)
          DZDE=1.
          BROW(M,IN)=BROW(M,IN)+WGL(IGAUSS)*SFN*JACOB*DZDE
          IC=3
          CALL  JACOBI(M,IGAUSS,IT,IC)
          SFN=SFL(IC,IN,IGAUSS)
          DZDE=1.
          BROW(M,IN)=BROW(M,IN)+WGL(IGAUSS)*SFN*JACOB*DZDE
          END DO Each Gauss point in turn
        END IF
!
!  P AT THE THIRD NODE OF THE ELEMENT.
        IF(IN == 3) THEN
!
!  TERM INVOLVING NORMAL QUADRATURE.
          Each Gauss point in turn: DO IGAUSS=1,NGAUSS
          IT=1
          IC=1
          CALL  JACOBI(M,IGAUSS,IT,IC)
          CALL  KERN2(M,IN,IGAUSS,BK2)
          SFN=SF(IN,IGAUSS)
          BROW(M,IN)=BROW(M,IN)+WG(IGAUSS)*BK2*SFN*JACOB
          END DO Each Gauss point in turn
!
!  TERM INVOLVING LOGARITHMIC QUADRATURE.
          Each Gauss point in turn: DO IGAUSS=1,NGAUSS
          IT=2
          IC=4
          CALL  JACOBI(M,IGAUSS,IT,IC)
          SFN=SFL(IC,IN,IGAUSS)
          DZDE=2.
          BROW(M,IN)=BROW(M,IN)+WGL(IGAUSS)*SFN*JACOB*DZDE
          END DO Each Gauss point in turn
        END IF
      END IF
      END DO Each element node in turn
!
      RETURN
      END SUBROUTINE INTKER
```

P not at the current node of the element containing Q

If P and Q are not in the same element, or P is not at the current node of the element containing Q (that is, I≠J) then normal Gaussian quadrature (Appendix A) over the whole element is carried out to evaluate the integrals. The required shape functions at the Gauss points have already been computed in subprogram SHAPE and stored in array SF. The Jacobians at the Gauss points are found in subprogram JACOBI, and the kernel functions in subprogram KERNEL. The last of these subprograms stores the kernel functions in variables AK and BK, respectively. For each Gauss point the products of Gaussian weighting factor (WG), kernel function (AK or BK), shape function (SFN) and Jacobian (JACOB) are then added to either arrays AROW or BROW to complete the numerical integration process.

P at the current node of the element containing Q

If P and Q are in the same element and P is at the current node of the element (that is, I= J) then, because of the singular nature of the second kernel function, some logarithmic Gaussian quadrature is required, as described in Section 2.5.2. The first kernel function does not have to be evaluated because the relevant diagonal coefficient of the matrix $[A]$ has already been found indirectly with the aid of Equation 2.66.

If P is the first node of the element the integration process follows Equations 2.68 to 2.78. The integral of the second kernel function is split into singular and non-singular parts. To the latter is applied Gaussian quadrature in the usual way, the non-singular part of the second kernel function being defined in subprogram KERN2 and returned in variable BK2. The singular part is integrated using logarithmic Gaussian quadrature (Appendix A) applied to the whole element, with the origin of the modified intrinsic co-ordinate η being at the first node. The derivative of ξ with respect to η, $|d\xi/d\eta|$, is stored in variable DZDE.

If P is the second node of the element the integration process follows Equations 2.79 to 2.90. The integral of the second kernel function is again split into singular and non-singular parts. To the latter is applied Gaussian quadrature in the usual way, the non-singular part of the second kernel function being defined in subprogram KERN2 and returned in variable BK2. The singular part is integrated using logarithmic Gaussian quadrature applied to the two halves of the element separately, in each case with the origin of the modified intrinsic co-ordinate η at the second node.

If P is the third node of the element the integration process follows Equations 2.91 to 2.101, and is very similar to the case of P (and Q) at the first node of the element.

4.1.10 Subprogram for computing the Jacobian at a point on an element

Subprogram JACOBI computes the Jacobian of co-ordinate transformation from global to local intrinsic at a particular Gauss point on an element. To do this it uses Equations 2.60, 2.57 and 2.58. The shape function derivatives required for Equation 2.58 are obtained from either array SD for a Gauss point used in normal quadrature, or array SDL for a Gauss point used in logarithmic quadrature. For the latter there are four possible alternative cases, defined by IC, according to the extent of, and origin for, the logarithmic quadrature. The distinction here between normal quadrature and the various cases of logarithmic quadrature is only important in defining the position of the Gauss point at which the Jacobian is to be computed.

```
      SUBROUTINE  JACOBI(M,IGAUSS,IT,IC)
!
!  SUBPROGRAM TO EVALUATE THE JACOBIAN AND THE COMPONENTS OF THE UNIT
!  NORMAL VECTOR AT A PARTICULAR GAUSS POINT.
!  M INDICATES THE ELEMENT NUMBER.
!  IGAUSS INDICATES THE GAUSS POINT NUMBER.
!  IT INDICATES THE TYPE OF QUADRATURE.
!     IT=1 - NORMAL GAUSSIAN QUADRATURE.
!     IT=2 - LOGARITHMIC GAUSSIAN QUADRATURE.
!  IC INDICATES THE CASE NUMBER FOR LOGARITHMIC GAUSSIAN QUADRATURE,
!  AS DEFINED IN SUBROUTINE SHAPE.
!
      USE SHAREDDATA2PQ
!
!  CALCULATE THE DERIVATIVES OF THE GLOBAL COORDINATES WITH RESPECT TO
!  THE LOCAL INTRINSIC COORDINATE.
      DX=0.
      DY=0.
      Each element node in turn: DO IN=1,3
      IF(IT == 1) SFND=SD(IN,IGAUSS)
      IF(IT == 2) SFND=SDL(IC,IN,IGAUSS)
      J=NODE(M,IN)
      DX=DX+SFND*XNODE(J)
      DY=DY+SFND*YNODE(J)
      END DO Each element node in turn
!
!  COMPONENTS OF THE LOCAL OUTWARD NORMAL VECTOR AT THE GAUSS POINT.
      UNGX=DY
      UNGY=-DX
!
!  JACOBIAN OF THE COORDINATE TRANSFORMATION.
      JACOB=SQRT(UNGX**2+UNGY**2)
!
!  SCALE THE VECTOR COMPONENTS TO GIVE THE UNIT NORMAL VECTOR.
      UNGX=UNGX/JACOB
      UNGY=UNGY/JACOB
!
      RETURN
      END SUBROUTINE JACOBI
```

By products of the calculation for a Jacobian (JACOB) are the components of the outward normal vector to the boundary at the chosen Gauss point. These are converted to components of the unit normal and stored in variables UNGX and UNGY.

4.1.11 Subprogram for computing the kernel functions when *P* is not at the current node of the element containing *Q*

Subprogram KERNEL computes the kernel functions at a particular Gauss point Q with the aid of Equation 2.64. The co-ordinates XQ and YQ of the point are first found using Equations 2.54 and 2.55, followed by the components r_x and r_y (RX and RY) of the radius vector of length r (R) joining *P* and Q. Using the components of the unit normal previously found in subprogram JACOBI, the first kernel function (AK) is computed using Equation 2.64, and the second kernel (BK) from Equation 2.33.

```
      SUBROUTINE  KERNEL(XP,YP,M,IGAUSS,AK,BK)
!
!  SUBPROGRAM TO EVALUATE THE KERNELS WHEN P IS NOT THE CURRENT
!  ELEMENT NODE.
!  XP, YP INDICATE THE GLOBAL COORDINATES OF POINT P.
!  M INDICATES THE ELEMENT NUMBER.
!  IGAUSS INDICATES THE NUMBER OF THE GAUSS POINT, Q.
!
      USE SHAREDDATA2PQ
!
!  COORDINATES OF GAUSS POINT Q.
      XQ=0.
      YQ=0.
      Each element node in turn: DO IN=1,3
      SFN=SF(IN,IGAUSS)
      J=NODE(M,IN)
      XQ=XQ+SFN*XNODE(J)
      YQ=YQ+SFN*YNODE(J)
      END DO Each element node in turn
!
!  COMPONENTS AND MAGNITUDE OF THE RADIUS VECTOR FROM P TO Q.
      RX=XQ-XP
      RY=YQ-YP
      RSQ=RX**2+RY**2
      R=SQRT(RSQ)
!
!  EVALUATE THE KERNELS.
      AK=-(RX*UNGX+RY*UNGY)/RSQ
      BK=ALOG(1./R)
!
      RETURN
      END SUBROUTINE KERNEL
```

4.1.12 Subprogram for computing the second kernel function when *P* is at the current node of the element containing *Q*

Subprogram KERN2 computes, for a particular Gauss point , the non-singular logarithmic component of the second kernel function when source point *P* is at the current node of the element containing Q. In other words, the $\ln\left[\frac{1}{R(\xi)}\right]$ term (BK2 in the subprogram) in either Equation 2.78, 2.90 or 2.101, where $R(\xi)$ (RFN in the subprogram) is defined by either Equations 2.77, 2.88 or 2.100, according to whether *P* is at the first, second or third node of the element. The relevant value of the intrinsic co-ordinate ξ (variable ZETA in the program) is obtained from array ZG for normal Gaussian quadrature.

```
      SUBROUTINE  KERN2(M,IN,IGAUSS,BK2)
!
!  SUBPROGRAM TO EVALUATE THE NON-SINGULAR LOGARITHMIC TERM IN THE
!  SECOND KERNEL WHEN P IS THE CURRENT ELEMENT NODE.
!  M INDICATES THE ELEMENT NUMBER.
!  IN INDICATES THE NUMBER OF THE ELEMENT NODE FORMING P.
!  IGAUSS INDICATES THE GAUSS POINT NUMBER.
!
      USE SHAREDDATA2PQ
!
!  ELEMENT NODE NUMBERS.
```

Brain power
The Power of Knowledge Engineering
Plug into The Power of Knowledge Engineering.
Visit us at www.skf.com/knowledge
SKF

```
      I1=NODE(M,1)
      I2=NODE(M,2)
      I3=NODE(M,3)
!
!  EVALUATE THE INTRINSIC COORDINATE.
      ZETA=ZG(IGAUSS)
!
!  P AT THE FIRST NODE OF THE ELEMENT.
      IF(IN == 1) THEN
        XCOMP=(ZETA-2.)*XNODE(I1)+2.*(1.-ZETA)*XNODE(I2)+ZETA*XNODE(I3)
        YCOMP=(ZETA-2.)*YNODE(I1)+2.*(1.-ZETA)*YNODE(I2)+ZETA*YNODE(I3)
        RFN=SQRT(XCOMP**2+YCOMP**2)
      END IF
!
!  P AT THE SECOND NODE OF THE ELEMENT.
      IF(IN == 2) THEN
        XCOMP=-0.5*(ZETA-1.)*XNODE(I1)+ZETA*XNODE(I2)
     &        -0.5*(ZETA+1.)*XNODE(I3)
        YCOMP=-0.5*(ZETA-1.)*YNODE(I1)+ZETA*YNODE(I2)
     &        -0.5*(ZETA+1.)*YNODE(I3)
        RFN=SQRT(XCOMP**2+YCOMP**2)
      END IF
!
!  P AT THE THIRD NODE OF THE ELEMENT.
      IF(IN == 3) THEN
        XCOMP=-ZETA*XNODE(I1)+2.*(1.+ZETA)*XNODE(I2)-(ZETA+2.)*XNODE(I3)
        YCOMP=-ZETA*YNODE(I1)+2.*(1.+ZETA)*YNODE(I2)-(ZETA+2.)*YNODE(I3)
        RFN=SQRT(XCOMP**2+YCOMP**2)
      END IF
!
!  EVALUATE THE KERNEL TERM.
      BK2=ALOG(1./RFN)
!
      RETURN
      END SUBROUTINE KERN2
```

4.1.13 Results output subprogram

Subprogram OUTPUT organises and writes out the primary results of the computation, but not including secondary results such as values of potentials at internal points. Firstly, potentials and potential gradients are put back into their proper arrays, and the contributions of the particular integral added to get from the solution to the Laplace equation (with modified boundary conditions) which has been solved back to the "total" solution to the original problem. If the original problem was of the Laplace type, then the contributions from the particular integral potential and gradient are both zero.

```
      SUBROUTINE  OUTPUT
!
!  SUBPROGRAM TO WRITE OUT THE NODAL POINT VALUES OF POTENTIAL AND
!  POTENTIAL GRADIENT, AND COMPUTE POTENTIAL FLOWS ACROSS THE
!  BOUNDARY SEGMENTS.
!
      USE SHAREDDATA2PQ
!
!  ARRANGE FOR PSI AND DPSI TO CONTAIN THE POTENTIALS AND POTENTIAL
!  GRADIENTS, RESPECTIVELY.
      Each node in turn: DO I=1,NNP
      IF(IBCN(I) == 1) THEN
        TEMP=PSI(I)
        PSI(I)=DPSI(I)
        DPSI(I)=TEMP
      END IF
      IF(IBCN(I) == 3) THEN
        PSITOT=PSI(I)+PSIPI(I)
        DPSI(I)=(ALPHAN(I)*PSITOT+BETAN(I))*MAXL-DPSIPI(I)
      END IF
      END DO Each node in turn
!
!  ADD CONTRIBUTIONS OF THE PARTICULAR INTEGRAL.
      WRITE(6,61)
 61   FORMAT(/ "NODAL POINT POTENTIALS AND POTENTIAL GRADIENTS" //
     &  "     I      PSI            DPSI           UNX          UNY" )
      Each node in turn: DO I=1,NNP
      PSIT(I)=PSI(I)
      DPSIT(I)=DPSI(I)
      IF(F1 /= 0.) THEN
```

```
          PSIT(I)=PSIT(I)+PSIPI(I)
          DPSIT(I)=DPSIT(I)+DPSIPI(I)
        END IF
!
!  REMOVE THE LENGTH SCALE FACTOR FROM THE GRADIENT VALUES.
        DPSIT(I)=DPSIT(I)/MAXL
!
!  OUTPUT THE NODAL VALUES OF POTENTIAL, POTENTIAL GRADIENT,
!  AND THE UNIT NORMAL COMPONENTS.
        WRITE(6,62) I,PSIT(I),DPSIT(I),UNX(I),UNY(I)
 62     FORMAT(I6,2E15.6,2E13.4)
        END DO Each node in turn
!
!  COMPUTE THE POTENTIAL FLOWS ACROSS THE BOUNDARY SEGMENTS.
        FLOWIN=0.
        FLOWOUT=0.
        Each segment in turn: DO ISEG=1,NSEGTOT
        FLOWSEG(ISEG)=0.
        END DO Each segment in turn
```

```
      Each element in turn: DO M=1,NEL
      ISEG=ISEGELEM(M)
!
!  APPLY GAUSSIAN QUADRATURE.
      FLOWELEM=0.
      Each element node in turn: DO IN=1,3
      J=NODE(M,IN)
!
!  FIND THE POTENTIAL GRADIENT AT THE NODE.
      IF(IBCE(M) == 1) DPSISTORE=DPSIT(J)
      IF(IBCE(M) == 2) DPSISTORE=STORE(M)
      IF(IBCE(M) == 3) DPSISTORE=ALPHA(M)*PSIT(J)+BETA(M)
      Each Gauss point in turn: DO IGAUSS=1,NGAUSS
      SFN=SF(IN,IGAUSS)
      IT=1
      IC=1
      CALL  JACOBI(M,IGAUSS,IT,IC)
      FLOWELEM=FLOWELEM+WG(IGAUSS)*SFN*JACOB*DPSISTORE*MAXL
      END DO Each Gauss point in turn
      END DO Each element node in turn
!
!  ACCUMULATE THE FLOW ACROSS THE SEGMENT, AND THE TOTAL FLOWS IN
!  AND OUT OF THE DOMAIN.
      FLOWSEG(ISEG)=FLOWSEG(ISEG)+FLOWELEM
      IF(FLOWELEM > 0.) FLOWIN=FLOWIN+FLOWELEM
      IF(FLOWELEM < 0.) FLOWOUT=FLOWOUT-FLOWELEM
      END DO Each element in turn
!
!  OUTPUT THE POTENTIAL FLOW RESULTS.
      WRITE(6,63) (ISEG,FLOWSEG(ISEG),ISEG=1,NSEGTOT)
 63   FORMAT(/ "POTENTIAL FLOWS INTO DOMAIN ACROSS BOUNDARY SEGMENTS"
     &      // "SEGMENT      FLOW" / (I5,E16.6))
      WRITE(6,64) FLOWIN,FLOWOUT
 64   FORMAT(/ "TOTAL POTENTIAL FLOW INTO DOMAIN =",E15.6 //
     &         "TOTAL POTENTIAL FLOW OUT OF DOMAIN =",E15.6)
!
      RETURN
      END SUBROUTINE OUTPUT
```

If the boundary condition at a particular point is of the first type (prescribed potential) the solution array PSI contains the computed potential gradient, and the relevant components of arrays PSI and DPSI must be interchanged. If the boundary condition is of the third, mixed, type then potential was the unknown and is now correctly stored in array PSI. On the other hand, the potential gradient solution (for the Laplace problem) has to be found using Equation 3.21 and stored in DPSI.

Arrays PSI and DPSI now contain all the nodal point values of potential and potential gradient, either from the boundary conditions or from the boundary element solution. But they are for the Laplace equation with the particular integral subtracted. The latter contributions, stored in PSIPI and DPSIPI are now added to PSI and DPSI to give the "total" values solving the original problem and stored in PSIT and DPSIT. Both sets are scaled values, and the gradients have to be unscaled by dividing by the length scale, MAXL. For each node, the node number, potential and potential gradient are written out.

In many problems of practical engineering interest, the "flows" of potential across different parts of the boundary of a problem are of interest. Such flows are locally proportional to the potential gradient normal to the boundary, and their totals over parts or all of the boundary can be found by integrating the potential gradient with respect to distance along the boundary. In the case of quadratic boundary element results this has to be done using Gaussian quadrature to find the flow contribution of each element (FLOWELEM) from

$$Flow = \sum_{c=1}^{3} \int_{-1}^{+1} \left(\frac{\mathrm{d}\psi}{\mathrm{d}n}\right)_c N_c(\xi)\, J(\xi) L \,\mathrm{d}\xi \tag{4.7}$$

The length scale L is necessary in this expression because the Jacobian transforms between global scaled co-ordinates and the intrinsic co-ordinate, whereas integration with respect to unscaled distance along the boundary is required. Note that it is important to use the correct value of potential gradient at every element node, given that the gradients at nodes are not necessarily unique. For an element with prescribed potential this value is the computed nodal point gradient. For an element with prescribed potential gradient the correct value is that prescribed value. For an element with a mixed boundary condition the correct value is that defined by Equation 2.32, the appropriate potential value being that at the node concerned.

The flow contributions of individual elements are summed over the required parts of the boundary. In the subprogram, the total flows over each boundary segment (FLOWSEG) are computed, together with the total flows into (FLOWIN) and out of (FLOWOUT) the domain which are found and finally written out.

4.1.14 Subprogram for computing potentials at internal points

Subprogram INTERNAL implements Equation 2.26 to find values of potential at points within the solution domain. Internal point p can be chosen freely, but the result may not be very accurate if the point is very close to the boundary. The calculation procedure only breaks down completely if p happens to be chosen to coincide with a Gauss point of an element (when the distance between P and Q would be zero in subprogram KERNEL).

```
SUBROUTINE INTERNAL
!
!  SUBPROGRAM TO COMPUTE AND OUTPUT THE POTENTIALS AT SELECTED
!  INTERNAL POINTS.
!
      USE SHAREDDATA2PQ
!
!  INPUT NUMBER OF INTERNAL POINTS.
      READ(5,*) NINT
      IF(NINT == 0) RETURN
      WRITE(6,61)
 61   FORMAT(/ "POTENTIALS AT INTERNAL POINTS" //
     &        5X,"PSI                X           Y")
```

```
!
!  INPUT THE INTERNAL POINT COORDINATES.
      Each internal point in turn: DO IINT=1,NINT
      READ(5,*) XINT,YINT
      XP=XINT/MAXL
      YP=YINT/MAXL
!
!  INTEGRATE ROUND THE BOUNDARY.
      SUM=0.
      Each element in turn: DO M=1,NEL
      Each element node in turn: DO IN=1,3
      J=NODE(M,IN)
!
!  FIND THE POTENTIAL GRADIENT AT THE NODE, EITHER AS COMPUTED
!  IF POTENTIAL WAS PRESCRIBED, OR FROM THE PRESCRIBED GRADIENT
!  OR MIXED BOUNDARY CONDITIONS.
      IF(IBCE(M) == 1) DPSISTORE=DPSI(J)
      IF(IBCE(M) == 2) DPSISTORE=STORE(M)*MAXL-DPSIPIM(M,IN)
      IF(IBCE(M) == 3) DPSISTORE=(ALPHA(M)*PSIT(J)+BETA(M))*MAXL
     &                -DPSIPIM(M,IN)
!
!  APPLY GAUSSIAN QUADRATURE.
      Each Gauss point in turn: DO IGAUSS=1,NGAUSS
      IT=1
      IC=1
      CALL JACOBI(M,IGAUSS,IT,IC)
      CALL KERNEL(XP,YP,M,IGAUSS,AK,BK)
      SFN=SF(IN,IGAUSS)
      SUM=SUM+WG(IGAUSS)*SFN*JACOB*(-AK*PSI(J)+BK*DPSISTORE)
      END DO Each Gauss point in turn
      END DO Each element node in turn
      END DO Each element in turn
      PSIIP=SUM*0.5/PI
      PSIIPT=PSIIP
!
!  ADD THE PARTICULAR INTEGRAL FOR A POISSON TYPE PROBLEM.
      IF(F1 /= 0.) THEN
        XX=XP-XC
        YY=YP-YC
```

```
          PSIIPT=PSIIPT+0.25*F1*(XX**2+YY**2)
        END IF
!
!  OUTPUT THE POTENTIAL AT THE INTERNAL POINT.
        WRITE(6,62) PSIIPT,XINT,YINT
 62     FORMAT(E14.6,5X,2E12.4)
        END DO Each internal point in turn
!
        RETURN
        END SUBROUTINE INTERNAL
```

In the subprogram the number of internal points is first read into variable NINT. Then for each point in turn its global co-ordinates are input to XINT and YINT. These are first scaled and stored in XP and YP. The integration over the boundary required by Equation 2.26 is performed by summing over each of the boundary elements in turn.

$$2\pi\psi(p) + \oint_S \left(\psi\frac{\partial\phi}{\partial n} - \phi\frac{\partial\psi}{\partial n}\right) \mathrm{d}S = 0 \tag{2.26}$$

which becomes

$$2\pi\psi(p) = \sum_{m=1}^{M}\sum_{c=1}^{3}\left(-\psi\frac{\partial\phi}{\partial n} + \phi\frac{\partial\psi}{\partial n}\right) N_c(\xi)J(\xi)d\xi \tag{4.8}$$

where the gradient of ϕ, and ϕ itself, are the first and second kernel functions, respectively. In subprogram INTERNAL three summation loops are set up in turn: over the elements, over the nodes of each element, and over the Gauss points within the element. For a particular Gauss point the Jacobian is found from JACOBI, the two kernel functions from KERNEL (returned in AK and BK), and the shape function from array SFN. Finally, the summation expressed on the right hand side of Equation 4.8 is performed, the result being stored in SUM, and finally divided by 2π to give PSIIP, the potential at internal point p. Note that it is important to use the correct value of potential gradient at every element node, given that the gradients at nodes are not necessarily unique. For an element with prescribed potential this value is the computed nodal point gradient (before the particular integral contribution is added). For an element with prescribed potential gradient the correct value is that prescribed value minus the particular integral contribution for the node. For an element with a mixed boundary condition the correct value is that defined by Equation 2.32, minus the particular integral contribution, the appropriate potential value being the total value at the node concerned. In other words, the calculation is carried out in terms of the Laplace problem variables, with if necessary the particular integral contributions to the boundary conditions subtracted. If the problem is of the Poisson type, the particular integral contribution to the potential (Equation 2.108) at is p then added, to give the total potential there, PSIIPT. Finally, this value and the (unscaled) co-ordinates of the internal point are written out.

4.1.15 Subprogram for solving the linear equations

Both the Gaussian elimination algorithm and subprogram ELIMIN for solving the linear equations are described in Appendix B. The matrix $[A^*]$ extended to include the right hand side vector $[b]$ (in Equation 2.31) is entered in array A, and the solutions returned in array X (PSI in the calling main program).

4.2 Some Test Problems for BEM2PQ

Before attempting to solve real problems for which the solutions are not known, it is essential to test any computer program as extensively as possible on problems for which exact analytical solutions are available for comparison. If necessary, test problems have to be simplified to the point where analytical solutions can be obtained. Such problems serve not only to verify that the program is working correctly, but also to examine the level of discretisation (the fineness of the boundary element mesh in this case) necessary to obtain accurate results. It is also both convenient and appropriate to solve at least some of the test problems considered in Chapter 3, thereby allowing a direct comparison between constant and quadratic boundary elements. With quadratic elements, having nodes at points where boundary conditions change, not only in magnitude but also in type, it is important to include test cases where such changes occur.

4.2.1 A simple one-dimensional potential problem

The first test is the effectively one-dimensional one shown in Figure 3.2. The region concerned is rectangular, 4 units wide and 2 deep, subject to zero derivative boundary conditions on the top and bottom edges, a potential value of 5 on the left hand edge and a value of 1 at the right hand edge. The analytical solutions to the problem, both Laplace and Poisson, are derived in Section 3.2.1.

The Laplace problem

The exact analytical solution to the Laplace problem, with $f_1 = 0$, is

$$\psi = -x + 5 \,, \qquad \frac{\mathrm{d}\psi}{\mathrm{d}x} = -1 \tag{4.9}$$

The potential "flows" into the left hand edge and out of the right hand edge of the domain are both of magnitude 2 (gradient times edge length).

To obtain a quadratic boundary element solution to the problem, the boundary has first to be divided into segments. Four segments along the four edges of the domain are appropriate, because they are straight lines between corners and the boundary condition over each one is uniform. The ends of the segments are shown numbered 1 to 4 in Figure 3.2. The choice of starting point is arbitrary, but then the direction of numbering must keep the domain to the left. The data file DATA to solve the problem using one element (three nodes) per side of the domain is as follows

```
ONE-DIMENSIONAL TEST PROBLEM
0.                                (Value of f1)
1                                 (Number of boundaries)
4                                 (Number of segments on the boundary)
0. 0. 4. 0. 4. 2. 0. 2.           (Co-ordinates of the segment ends)
0. 1 1.                           (Radii of curvature,
0. 1 1.                           number of elements
0. 1 1.                           and element length ratio
0. 1 1.                           for each of the 4 segments)
2 0 0                             (Two prescribed boundary potentials)
2 1.                              (Potential = 1 on segment 2)
4 5.                              (Potential = 5 on segment 4)
3                                 (Number of internal points)
1. 1.                             (Co-ordinates of first internal point)
2. 1.                             (Co-ordinates of second internal point)
3. 1.                             (Co-ordinates of third internal point)
```

The annotations on the right have been added for explanation, and are not part of the data file. The potential values at three internal points are called for, points E, F and G in Figure 3.2. Note that the prescribed zero values of potential gradient do not have to be entered explicitly.

The mesh data output file MESHRES is as follows

```
GEOMETRIC DATA FOR THE MESH

         NUMBER OF ELEMENTS = 4

         NUMBER OF NODAL POINTS = 8

COORDINATES OF THE NODAL POINTS

I        X          Y         I      X           Y
1  0.0000E+00  0.0000E+00   2  0.2000E+01  0.0000E+00
3  0.4000E+01  0.0000E+00   4  0.4000E+01  0.1000E+01
5  0.4000E+01  0.2000E+01   6  0.2000E+01  0.2000E+01
7  0.0000E+00  0.2000E+01   8  0.0000E+00  0.1000E+01

ELEMENT NODE NUMBERS

M   ND1   ND2   ND3         M     ND1   ND2   ND3
1   1     2     3           2     3     4     5
3   5     6     7           4     7     8     1
```

With one element per segment, 4 elements and 8 nodes are created. Both elements and nodes are numbered as for the segments, anticlockwise from the origin, as shown in Figure 4.1. Element numbers are shown circled. Element end nodes are shown as small solid circles, mid-side nodes as open circles.

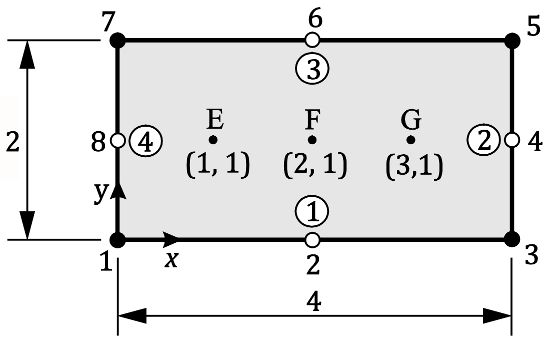


Figure 4.1 Discretisation of one-dimensional test problem

The RESULTS data file output by the program is

```
QUADRATIC BOUNDARY ELEMENT SOLUTION FOR TWO DIMENSIONAL POTENTIAL PROBLEM

ONE-DIMENSIONAL TEST PROBLEM

LAPLACE EQUATION

PRESCRIBED POTENTIAL BOUNDARY CONDITIONS
```

```
POTENTIAL = 0.1000E+01 ON ELEMENTS 2 TO 2

POTENTIAL = 0.5000E+01 ON ELEMENTS 4 TO 4

NODAL POINT POTENTIALS AND POTENTIAL GRADIENTS

   I      PSI          DPSI           UNX          UNY
   1 0.500000E+01  0.100151E+01 -0.1000E+01  0.0000E+00
   2 0.300000E+01  0.000000E+00  0.0000E+00 -0.1000E+01
   3 0.100000E+01 -0.100151E+01  0.1000E+01 -0.2666E-06
   4 0.100000E+01 -0.998856E+00  0.1000E+01  0.0000E+00
   5 0.100000E+01 -0.100151E+01  0.1000E+01 -0.2666E-06
   6 0.300000E+01  0.000000E+00  0.0000E+00  0.1000E+01
   7 0.500000E+01  0.100151E+01 -0.1000E+01  0.0000E+00
   8 0.500000E+01  0.998857E+00 -0.1000E+01  0.0000E+00

POTENTIAL FLOWS INTO DOMAIN ACROSS BOUNDARY SEGMENTS

SEGMENT             FLOW
   1             0.000000E+00
   2            -0.199948E+01
   3             0.000000E+00
   4             0.199948E+01

TOTAL POTENTIAL FLOW INTO DOMAIN = 0.199948E+01

TOTAL POTENTIAL FLOW OUT OF DOMAIN = 0.199948E+01

POTENTIALS AT INTERNAL POINTS

     PSI        X           Y
 0.399998E+01 0.1000E+01 0.1000E+01
 0.299834E+01 0.2000E+01 0.1000E+01
 0.199980E+01 0.3000E+01 0.1000E+01
```

The prescribed potential boundary conditions are a value of 1 over element 2 (nodes 3, 4 and 5) on the right end edge (segment 2), and a value of 5 over element 4 (nodes 7, 8 and 1) on the left hand edge (segment 4). Segments (and elements) 1 and 3 defining the top and bottom edges had no boundary condition prescribed by the DATA file, so zero normal derivative conditions are assumed.

The computed results show that at nodes 2 and 6 (at $x = 2$ on the bottom and top edges, respectively) the exact potential value of 3 (Equations 4.9) is computed. At node 4 at the centre of the right hand edge the computed potential gradient is -0.9989 compared with the exact -1 (Equations 4.9), and similarly at node 8 at the left hand edge it is +0.9989 compared with +1 (at this edge the normal direction is in the negative x direction). While at node 4 the computed gradient is 0.11% too low, at the adjacent nodes 3 and 5 it is 0.15% too high.

The computed potential "flows" into and out of the domain across the boundary segments are both 1.999 (to four significant figures), compared with the exact figure of 2. The computed values of potential at points E, F and G (Figure 4.1) are 4.000, 2.998, and 2.000 (to four significant figures), compared with exact values of 4, 3 and 2. Although the computed potential gradients on the left hand and right hand edges of the boundary fluctuate about their exact values, the smoothing and averaging effect of integration to produce either the flows across the boundary, or the potential at internal points, gives results of higher accuracy.

Compared to the constant element results presented in Section 3.2.1 the present ones show a massive improvement in accuracy. Using just one quadratic element and three nodes on each of the four edges of the boundary gives results which are almost correct to four significant figures. This level of accuracy was only achieved using constant elements when about a hundred or more nodes were used on each boundary. Note that in comparisons between constant and quadratic elements it is the number of nodes used for each that is the more relevant (rather than the number of elements). This is because it is the number of nodes that determines the amount of computation involved in solving the linear equations, which at least for larger problems represents the dominant computing cost.

For this very simple one-dimensional problem it is hardly necessary to refine the mesh because the accuracy of the results is already adequate for most practical engineering purposes. It is relevant, however, to examine the extent to which the potential gradient fluctuations over, say, the right hand boundary can be reduced by mesh refinement. The following is an extract from the RESULTS file for the case of 3 elements (7 nodes) per edge of the domain for the nodes on the right hand edge.

```
I        PSI              DPSI
 7  0.100000E+01     -0.100164E+01
 8  0.100000E+01     -0.999107E+00
 9  0.100000E+01     -0.100136E+01
10  0.100000E+01     -0.999213E+00
11  0.100000E+01     -0.100136E+01
12  0.100000E+01     -0.999108E+00
13  0.100000E+01     -0.100164E+01
```

With this mesh the computed flows and potentials at internal points are all exact to at least four significant figures, so the average gradient over the right hand edge is accurate. Similarly, with 15 nodes per edge

```
I        PSI               DPSI
15  0.100000E+01     -0.100171E+01
16  0.100000E+01     -0.999152E+00
17  0.100000E+01     -0.100139E+01
18  0.100000E+01     -0.999261E+00
19  0.100000E+01     -0.100137E+01
20  0.100000E+01     -0.999266E+00
21  0.100000E+01     -0.100137E+01
22  0.100000E+01     -0.999262E+00
23  0.100000E+01     -0.100137E+01
24  0.100000E+01     -0.999267E+00
25  0.100000E+01     -0.100136E+01
26  0.100000E+01     -0.999260E+00
27  0.100000E+01     -0.100140E+01
28  0.100000E+01     -0.999146E+00
29  0.100000E+01     -0.100172E+01
```

While there is some reduction in gradient fluctuation in going from 3 nodes per edge to 7 nodes per edge, there is no significant further change produced by increasing to 15 nodes per edge, or indeed by refinement to many more than 15 nodes per edge.

It turns out that the small fluctuations about the true value of gradient are caused by the differences between the (odd-numbered) end nodes and (even numbered) mid-side nodes of the elements. While mid-side nodes attract integral contributions from only their own elements, end notes attract contributions from the whole of the two elements with which they are associated. The way to reduce the fluctuations is to increase the accuracy of the Gaussian quadrature used, by increasing the number of Gauss points used in each element integration. This explains why the relatively large number of eight Gauss point is used here. Using more points is of course possible, but is probably unnecessary for most practical purposes.

The Poisson problem

Introducing a non-zero value of $f_1 = 1$, the exact analytical solution to the Poisson problem is given by Equation 3.29

$$\psi = \frac{x^2}{2} - 3x + 5, \qquad \frac{\mathrm{d}\psi}{\mathrm{d}x} = x - 3 \tag{4.10}$$

Although the analytical solution remains one-dimensional, the two-dimensional form of the particular integral means that the Laplace problem solved by the program, with boundary conditions which are now two-dimensional, is certainly two-dimensional. The only change required to file DATA is to replace the zero value (for f_1) in the second line by the value 1., followed immediately by a new line with the co-ordinates of the origin for the particular integral function. The centre of the domain at $x = 2., y = 1$ is convenient.

Again using one element on each edge of the domain as in Figure 4.1, the computed potential gradients at the centres of the left and right hand edges are 2.998 (node 8) and 1.000 (node 4), compared with the exact 3 and 1. The computed potential "flows" into the left hand and right hand edges of the domain are 6.000 and 2.001, compared with the exact 6 and 2. The computed potential at internal point E ($x = 1$) in Figure 4.1 is 2.500, which is exact to four significant figures.

Again, the accuracy is excellent, with hardly any need to refine the boundary element mesh beyond one element per edge of the domain. Clearly, the change from a Laplace to a Poisson type problem, with the introduction of a two-dimensional particular integral, does not adversely affect accuracy.

Internal points close to the boundary

In Figure 4.1 the chosen internal points E, F and G are not close to any part of the boundary. With most boundary element methods, taking an internal point close to the boundary leads to loss of accuracy, due to the singular nature of the fundamental solutions and the use of approximate numerical integration. Table 4.1 shows the effect of taking internal point E from its initial position at (1, 1) progressively closer to the boundary at (1, 0) (not varying its x co-ordinate, $x = 1$). The meshes used are 1, 10 and 100 elements per edge.

y	3 nodes per edge	21 nodes per edge	201 nodes per edge
1	2.4997	2.4999	2.5000
10^{-1}	3.2869	2.4994	2.5000
10^{-2}	2.2464	1.9037	2.5000
10^{-3}	1.5469	1.5464	2.5344
10^{-4}	1.4741	1.5083	1.7276
10^{-5}	1.4668	1.5045	1.5231
10^{-6}	1.4661	1.5041	1.5024
0	1.4660	1.5041	1.5000

Table 4.1 Potential at internal points at (1,*y*)

The effect of approaching the boundary is considerable. Using one element per edge, only when the internal point is at $y = 1$ is the computed potential accurate. Increasing the number of nodes per edge to 10, accuracy is maintained at $y = 10^{-1}$, and to 100 per edge it is maintained at $y = 10^{-2}$. In other words, internal points should not be taken closer than to within about a quarter of an element length of a boundary.

This loss of accuracy as an internal point approaching a boundary is in direct contrast to constant elements (Table 3.3), where much better accuracy was obtained both very close to and on the boundary. The reason for this is that, while constant elements employ exact analytical integration, quadratic elements rely on approximate numerical integration.

Domain aspect ratio

Rather like taking an internal point close to a boundary, making the domain very narrow and bringing opposite sides of the boundary close together is likely to cause loss of accuracy. In the present problem, the width of the domain is twice its height, giving an aspect ratio of 2. Table 4.2 shows the effects of increasing this aspect ratio, by progressively reducing the depth of the domain, using a mesh of one element (3 nodes) per edge.

Aspect ratio	Potential at node 2	Gradient at node 4
Exact	1	1
2	0.9989	1.0004
4	0.9992	0.9999
8	1.0005	0.9969
16	0.9999	0.9844
32	0.9790	0.9433
64	0.9054	0.9350

Table 4.2 Effects of varying domain aspect ratio

ie business school
#1 EUROPEAN BUSINESS SCHOOL
FINANCIAL TIMES 2013
#gobeyond
MASTER IN MANAGEMENT
Because achieving your dreams is your greatest challenge. IE Business School's Master in Management taught in English, Spanish or bilingually, trains young high performance professionals at the beginning of their career through an innovative and stimulating program that will help them reach their full potential.
Choose your area of specialization.
Customize your master through the different options offered.
Global Immersion Weeks in locations such as Rio de Janeiro, Shanghai or San Francisco.
Because you change, we change with you.
www.ie.edu/master-management
mim.admissions@ie.edu
Click on the ad to read more

The variables selected for scrutiny are the potential at node 2, at the centre of the lower edge, and the potential gradient at node 4, at the centre of the right hand edge, in both cases the exact values being 1. There is progressive deterioration of accuracy with increasing aspect ratio, the effect becoming significant beyond an aspect ratio of about 8. At this aspect ratio, the distance between the top and bottom edges of the domain is one eighth of the length of the elements on these edges. This is actually approaching quite close, thanks to using eight Gaussian quadrature points per element integration which gives relatively high accuracy. Again there is a contrast with constant elements, which are able to maintain greater accuracy with increasing domain aspect ratio thanks to exact analytical integration.

Prescribed potential gradient boundary conditions

So far, only prescribed potential and zero potential gradient boundary conditions have been used. Staying with the present Poisson problem, the condition on the left hand edge can be changed from a prescribed potential of magnitude 5 to a prescribed gradient of 3: the problem is unchanged. If this is done using a mesh with one element (3 nodes) per edge, the following are obtained

Potential at (2, 0) = 0.99915 (exact 1)

Potential gradient at (4, 1) = 1.00009 (exact 1, previously 0.99886)

Potential at (0, 1) = 5.0010 (exact 5, previously 5 prescribed)

The results are, as is to be expected, not identical to those obtained when potential rather than the gradient is prescribed, but the accuracy is not impaired. Compared with constant elements, these results with one quadratic element per edge are as good as those with 61 constant elements per edge.

This example provides a test of a situation where prescribed potential gradient boundary conditions change abruptly (at nodes 1 and 7 in Figure 4.1).

Mixed boundary conditions

Again staying with the same problem, on the left hand edge the exact solution gives the potential as 5 and gradient as 3, so in place of either the prescribed potential or prescribed gradient, a possible mixed boundary condition on the edge is

$$\frac{d\psi}{dn} = \psi - 2 \tag{4.11}$$

Prescribing values of $\alpha = 1$ and $\beta = -2$ and again using a mesh of one element (3 nodes) per edge, the following are obtained

Potential at (2, 0) = 0.99874 (exact 1)

Potential gradient at (4, 1) = 1.0004 (exact 1)

Potential at (0, 1) = 5.0004 (exact 5)

Potential gradient at (0, 1) = 3.0004 (exact 3)

Again there is no loss of accuracy, and the results are as least as good as those for constant elements using 61 nodes per edge.

This example provides a test of a situation where boundary conditions change from prescribed potential gradient to mixed type (at nodes 1 and 7 in Figure 4.1).

4.2.2 A thick-walled cylinder test problem

The second test is designed to demonstrate the ability of the program to solve problems with curved boundaries, problems with more than one boundary, and the use of elements of varying size along a boundary segment. Figure 3.3 shows the cross-section of a thick-walled circular cylinder with internal and external radii of r_1 and r_2. The boundary conditions are prescribed potentials of ψ_1 and ψ_2 at the inner and outer surfaces, respectively. A practical engineering application would be to heat conduction through the walls of a cylinder with different temperatures at the two curved surfaces.

The analytical solution to the problem is derived in Section 3.2.2, leading to Equations 3.34 and 3.35 for the potential and potential gradient distributions

$$\psi = \psi_1 + \frac{f_1}{4}(r^2 - r_1^2) + \left[(\psi_2 - \psi_1) - \frac{f_1}{4}(r_2^2 - r_1^2)\right]\frac{\ln\left(\frac{r}{r_1}\right)}{\ln\left(\frac{r_2}{r_1}\right)} \tag{4.12}$$

$$\frac{\mathrm{d}\psi}{\mathrm{d}r} = \frac{f_1 r}{2} + \left[(\psi_2 - \psi_1) - \frac{f_1}{4}(r_2^2 - r_1^2)\right]\frac{1}{r\ln\left(\frac{r_2}{r_1}\right)} \tag{4.13}$$

Laplace problem

Consider first the Laplace problem with $f_1 = 0$, corresponding to thermal conduction with no heat generation or absorption within the wall of the cylinder. Take the case where $r_1 = 1$, $r_2 = 10$, $\psi_1 = 8$, and $\psi_2 = 1$. The ratio between the outer and inner radii is chosen to be quite large for reasons that should become clearer later. With these data, the potential and potential gradient distributions are

$$\psi = 8 - 3.04006 \ln r \tag{4.14}$$

$$\frac{\mathrm{d}\psi}{\mathrm{d}r} = -\frac{3.04006}{r} \tag{4.15}$$

The potential values at typical internal points are at $r = 4$, $\psi = 3.7856$ and at $r = 7, \psi = 2.0843$.

Bearing in mind that program BEM2PQ cannot accept boundary segment specifications with angles greater than $180°$, the simplest mesh to represent the geometry shown in Figure 3.3 requires four segments, two on the outer boundary with end points arbitrarily chosen at (10, 0) and (-10, 0), and two on the inner boundary with end points at (0, -1) and (0, 1). These end points are the points numbered 1 to 4 in Figure 3.3. The following is an annotated DATA input file for this problem, for the case of four elements (9 nodes) per semi-circular segment, two per quadrant. This means that each element has to represent an arc of $45°$ by means of a parabolic (quadratic) curve passing through both the ends and midpoint of the arc. The file calls for potentials to be computed at two internal points, on the x axis at radii of 4 and 7.

`THICK-WALLED CYLINDER`	
`0.`	(Value of f_1)
`2`	(Number of boundaries)
`2`	(Number of segments on first boundary)
`10. 0. -10. 0.`	(Co-ordinates of the segment ends)
`10. 4 1.`	(Radii of curvature, numbers of elements
`10. 4 1.`	and length ratios)
`2`	(Number of segments on second boundary)
`0. -1. 0. 1.`	(Co-ordinates of the segment ends)
`-1. 4 1.`	(Radii of curvature, numbers of elements
`-1. 4 1.`	and length ratios)
`4 0 0`	(Four prescribed boundary potentials)
`1 1.`	(Potential on segment 1)
`2 1.`	(Potential on segment 2)
`3 8.`	(Potential on segment 3)
`4 8.`	(Potential on segment 4)
`2`	(Number of internal points)
`4. 0.`	(Co-ordinates of first internal point)
`7. 0.`	(Co-ordinates of second internal point)

Table 4.3 shows the results obtained for meshes ranging from one element per quadrant up to four elements per quadrant. For the computed gradients, two values are shown, the first for element end nodes, and the second for midside nodes.

Elements per quadrant	Gradient at $r = 1$	Gradient at $r = 10$	Potential at (4, 0)	Potential at (7, 0)	Flow in at $r = 1$	Flow out at $r = 10$
Exact	3.0401	-0.30401	3.7856	2.0843	19.101	19.101
1	3.069 3.047	-0.3054 -0.3033	3.7767	2.0752	19.094	19.002
2	3.046 3.038	-0.3044 -0.3037	3.7850	2.0839	19.100	19.091
3	3.044 3.038	-0.3044 -0.3037	3.7855	2.0843	19.100	19.097
4	3.044 3.038	-0.3044 -0.3038	3.7856	2.0844	19.100	19.100

Table 4.3 Results for the thick-walled cylinder Laplace problem

Since the problem is axisymmetric it is to be expected that the computed gradients at all the nodes on the inner boundary would be identical, and similarly for all the nodes on the outer boundary. This proves to be the case, but for expected fluctuations between the values at element end nodes and midside nodes. These do not improve with mesh refinement beyond two elements per quadrant. At two elements per quadrant the results, and particularly the results smoothed by integration to give potential at internal points or flows in and out of the domain, are almost correct to four significant figures, which is adequate for most practical purposes. In this case each element covers a circular arc of $45°$, which demonstrates the ability of quadratic elements to accurately model curved boundaries. Sixteen elements having 32 nodes are sufficient to model this multiply-connected solution domain.

Given the very superior performance of quadratic elements over constant ones for problems with straight boundaries, it is not at all surprising that this superiority is maintained for problems with curved boundaries, for which they are even more obviously better suited.

Poisson problem

Consider the same problem, but of the Poisson type with $f_1 = -1$, corresponding to thermal conduction with heat generation within the wall of the cylinder. Equations 4.12 and 4.13 reduce to

$$\psi = 8.25 - \frac{r^2}{4} - 7.70872 \ln r \tag{4.16}$$

$$\frac{d\psi}{dr} = -\frac{r}{2} - \frac{7.70872}{r} \tag{4.17}$$

Hence, the gradients at the inner and outer surfaces of the cylinder are 7.2087 and -4.2291, respectively. The potential values at typical internal points are at $r = 4,\ \ \psi = 14.937$ and at $r = 7\ \psi = 11.000$. The origin for the particular integral function for this Poisson problem is taken as the centre of the cylinder, which in this case coincides with the origin chosen for the global co-ordinates.

Table 4.4 again shows the results obtained for meshes ranging from one element per quadrant up to four elements per quadrant. For the computed gradients, two values are shown, the first for element end nodes, and the second for midside nodes.

Elements per quadrant	Gradient at $r = 1$	Gradient at $r = 10$	Potential at (4, 0)	Potential at (7, 0)	Flow out at $r = 1$	Flow out at $r = 10$
Exact	-7.2087	-4.2291	14.937	11.000	45.294	265.72
1	-7.283 -7.228	-4.226 -4.231	14.959	11.024	45.293	264.30
2	-7.223 -7.205	-4.228 -4.230	14.938	11.001	45.290	265.63
3	-7.220 -7.203	-4.228 -4.230	14.937	11.000	45.291	265.71
4	-7.219 -7.203	-4.228 -4.230	14.937	11.000	45.292	265.72

Table 4.4 Results for the thick-walled cylinder Poisson problem

The trends are very similar to those observed for the Laplace version, with good accuracy at two elements per quadrant, and little improvement beyond that level of mesh refinement.

Use of Symmetry

Returning to the Laplace problem, there is no need to analyse the whole of this axisymmetric problem, but only a small part of it. Figure 3.4 shows one quadrant of the thick-walled cylinder with the relevant boundary conditions on the curved surfaces. The normal gradients on the planes of symmetry that cut through the cylinder walls are zero. Of course, a slice substantially smaller than a quadrant could have been analysed. Not only is the size of the problem reduced, but it has been simplified from multiply-connected (two boundaries) to singly-connected (one boundary). It remains to be seen what benefits, if any, this offers. The boundary is divided into four segments, with the numbered end points as shown.

Table 4.5 shows the results obtained for meshes ranging from one element per segment of the cylinder quadrant (one element on each of the curved parts of the boundary) up to four elements per segment. These should be compared with Table 4.3 for the whole cylinder. Note that the internal points for potential evaluation are at $(\frac{4}{\sqrt{2}}, \frac{4}{\sqrt{2}})$ and $(\frac{7}{\sqrt{2}}, \frac{7}{\sqrt{2}})$, at $r = 4$ and $r = 7$, but no longer on the axis, which now forms part of the boundary. The magnitudes of the potential flows in and out of the domain are reduced by factors of 4 because only a quarter of the problem is now being considered. For the computed gradients, two values are shown, the first for end nodes, and the second for midside nodes, of the elements on the curved surfaces at the corner segment ends (numbered 1 and 4 for the inner surface, 2 and 3 for the outer surface, in Figure 3.4). These upper and lower values are the maximum values and minimum values for the inner and outer curved surfaces, respectively.

Elements per segment	Gradient at $r=1$	Gradient at $r=10$	Potential at $r=4$	Potential at $r=7$	Flow in at $r=1$	Flow out at $r=10$
Exact	3.0401	-0.30401	3.7856	2.0843	4.7753	4.7753
1	3.618	-0.3107	3.9083	2.1104	4.6514	4.8267
	2.620	-0.3079				
2	3.500	-0.3048	3.7899	2.0807	4.7172	4.7619
	2.857	-0.3026				
3	3.420	-0.3041	3.7805	2.0814	4.7460	4.7635
	2.906	-0.3029				
4	3.352	-0.3041	3.7814	2.0824	4.7589	4.7672
	2.934	-0.3032				

Table 4.5 Results for the thick-walled cylinder quadrant Laplace problem

The computed gradients on the inner curved part of the boundary vary by far more than the expected fluctuation from end node to midside in each element. The results overall are substantially inferior to those in Table 4.3 for the same problem.

The explanation for this behaviour is the nature of the mesh applied to this particular geometry. A thick-walled cylinder geometry with external radius ten times the internal radius was chosen deliberately so that the potential varies much more rapidly with radius at the inner surface than at the outer surface. When the whole cylinder was analysed this was not an issue. But the use of a small number of elements over the lines of symmetry of one quadrant of the cylinder does not adequately model the variation of potential there.

Experience so far suggests that two quadratic elements per quadrant provide satisfactory modelling of a circular curved boundary when the potential and potential gradient do not change significantly along the arc. Table 4.6 shows the results obtained when the mesh has two elements on each of the curved boundaries, but a number of uniformly sized elements on the straight segments of the boundary varying from 2 to 16.

Elements per straight segment	Gradient at $r = 1$	Gradient at $r = 10$	Potential at $r = 4$	Potential at $r = 7$	Flow in at $r = 1$	Flow out at $r = 10$
Exact	3.0401	-0.30401	3.7856	2.0843	4.7753	4.7753
2	3.500 2.857	-0.3048 -0.3026	3.7899	2.0807	4.7172	4.7619
4	3.176 2.985	-0.3041 -0.3029	3.7817	2.0823	4.7612	4.7667
8	3.069 3.029	-0.3044 -0.3036	3.7844	2.0836	4.7731	4.7714
16	3.048 3.038	-0.3045 -0.3036	3.7850	2.0839	4.7749	4.7723

Table 4.6 Results for the thick-walled cylinder quadrant Laplace problem,with two elements on each curved boundary

The critical result is that for the potential gradient at $r = 1$, in the first column of results. Using 16 elements on each of the straight segments (36 elements in total) gives results here similar to those in Table 4.3 for the whole problem using 2 elements per quadrant, only 16 elements in total. So, in this case there is no computational benefit in using symmetry to reduce the size of the solution domain. But, using 16 uniform elements on each of the straight segments is inefficient: while smaller elements are required at the inner surface of the cylinder, this is not necessary at the outer surface.

The use of non-uniform distributions of elements over the straight segments should be helpful. The method of varying the elements sizes which is implemented in the program is described in Sections 3.1.3 and 4.1.3. The key parameter is *S*, the ratio between the lengths of successive elements in the direction of numbering. A value of greater than unity is required on the first segment joining the end points numbered 1 and 2 in Figure 3.4, to ensure that the element size increases from point 1 towards point 2. On the other hand, on the third segment joining points 3 and 4, the corresponding ratio should be $1/S$, to ensure that element size decreases from point 3 towards point 4, in the direction of numbering.

Table 4.7 shows the effect of increasing the length ratio S applied in this way from unity up to 5.0. The mesh used has two elements on each of the curved surfaces, and 4 on each of the straight ones, 12 in total.

S	Gradient at $r = 1$	Gradient at $r = 10$	Potential at $r = 4$	Potential at $r = 7$	Flow in at $r = 1$	Flow out at $r = 10$
Exact	3.0401	-0.30401	3.7856	2.0843	4.7753	4.7753
1.0	3.176	-0.3041	3.7817	2.0823	4.7612	4.7667
	2.985	-0.3029				
1.2	3.113	-0.3043	3.7833	2.0831	4.7685	4.7695
	3.010	-0.3035				
1.4	3.079	-0.3044	3.7844	2.0836	4.7720	4.7712
	3.024	-0.3036				
1.6	3.062	-0.3045	3.7851	2.0840	4.7735	4.7722
	3.032	-0.3036				
1.8	3.055	-0.3046	3.7855	2.0842	4.7741	4.7727
	3.035	-0.3036				
2.0	3.052	-0.3047	3.7860	2.0845	4.7743	4.7731
	3.036	-0.3036				
2.2	3.051	-0.3048	3.7865	2.0848	4.7743	4.7735
	3.037	-0.3035				
5.0	3.075	-0.3081	3.8034	2.0921	4.7692	4.7815
	3.027	-0.3026				

Table 4.7 Effects of varying element length ratio for a mesh of two elements per quadrant and four elements per straight segment

While there is a steady improvement in gradient values at the inner cylinder surfaces for ratios up to 2.2, there is some deterioration in the values at the outer surface. Further increasing the ratio only makes matters worse: the last case of a ratio of 5.0 is deliberately excessive. Overall, the optimum ratio is around 2.0, at which the results are similar in accuracy to those for the whole problem using 16 elements. There is no significant computational benefit in this case in taking advantage of symmetry, but if the reduced domain is analysed it is important to use non-uniform element distributions on the lines of symmetry to give small elements in regions of the domain boundary where the variables change rapidly.

4.3 An Example: Downstream Viscous Flow in a Rectangular Channel

A more practical problem for solution by program BEM2PQ is the viscous channel flow problem described in Section 1.2.1, and solved using constant boundary elements in Section 3.3.

Analytical solution

As explained in Section 3.3, the analytical solution to the problem leads to the equation in dimensionless variables displayed in Equation 3.46

$$\pi_Q = \frac{F_D}{2} - \frac{\pi_P F_P}{12} \tag{4.18}$$

The dimensionless flow rate and pressure gradient are defined as

$$\pi_Q = \frac{Q}{WHV_z}, \qquad \pi_P = \frac{P_z H^2}{\mu V_z} \tag{4.19}$$

The drag and pressure flow shape factors are given in series form by Equations 3.44 and 3.45.

The problem

The problem is to use quadratic boundary element program BEM2PQ to compute the values of dimensionless flow rate, π_Q, for a series of values of dimensionless pressure gradient, π_P, of $-2, 0, 2$ and 4, for a channel that is five times as wide as it is deep. For $H/W = 0.2$ the particular form of Equation 4.18 is

$$\pi_Q = 0.4457 - 0.07283\,\pi_P \tag{4.20}$$

where the constants are given to four significant figures. Note again that a positive pressure gradient, indicating pressure increasing with z normal to the channel cross-section in the direction of the movement of the top boundary tends to cause flow in the opposite direction, and hence a reduction in the overall flow rate. A negative pressure gradient, on the other hand, enhances the flow rate.

Integration of potential over the domain

Before attempting to solve the problem, it is necessary to extend the program to compute the integral of potential over the solution domain in order to be able to find the volumetric flow rate.

To write a piece of program to meet this requirement for any domain, of arbitrary shape with an arbitrary number of boundaries, is not straightforward, and is beyond the scope of this book. The approach adopted is therefore to modify subprogram INTERNAL, very much as described in Section 3.3 for the constant element program, to carry out the required integration for a domain which is rectangular in shape (but of arbitrary depth to width ratio), and has values of potential prescribed on each of its four edges (the four values can be different). Figure 3.5 shows the channel cross-section with its boundary divided into four segments, with their ends numbered in order from the bottom left hand corner.

```
      SUBROUTINE INTERNAL
!
!  SUBPROGRAM TO FIND AND OUTPUT THE INTEGRAL OF POTENTIAL OVER THE
!  AREA OF THE SOLUTION DOMAIN.
!  THIS VERSION IS DESIGNED FOR A RECTANGULAR DOMAIN.
!
      USE SHAREDDATA2PQ
      REAL :: PSIVAL(300),YINTGL(300)
!
!  INPUT THE NUMBERS OF SAMPLING POINTS INSIDE THE DOMAIN IN THE X
!  AND Y DIRECTIONS, WHICH SHOULD BOTH BE ODD.
      READ(5,*) NX,NY
      IF(MOD(NX,2) == 0) THEN
        WRITE(6,61) NX
 61     FORMAT(/ "WARNING - NX =",I4," NOT ODD - INCREASE BY 1")
        NX=NX+1
      END IF
      IF(MOD(NY,2) == 0) THEN
        WRITE(6,62) NY
 62     FORMAT(/ "WARNING - NY =",I4," NOT ODD - INCREASE BY 1")
        NY=NY+1
      END IF
      IF(NX+2 > 300 .OR. NY+2 > 300) THEN
        WRITE(6,63) NX,NY
 63     FORMAT(/ "NX =",I6,5X,"NY =",I6,5X,"ARRAY DIMENSIONS EXCEEDED",
     &            " IN INTEGRATION OVER DOMAIN")
        STOP
      END IF
!
!  FIND PRESCRIBED BOUNDARY VALUES OF POTENTIAL.
      For each segment in turn: DO IBCP=1,NBCP
      IF(ISEGBC(IBCP) ==  1) PSIBOT=PSISEG(IBCP)
      IF(ISEGBC(IBCP) ==  2) PSIRIGHT=PSISEG(IBCP)
      IF(ISEGBC(IBCP) ==  3) PSITOP=PSISEG(IBCP)
      IF(ISEGBC(IBCP) ==  4) PSILEFT=PSISEG(IBCP)
      END DO For each segment in turn
!
!  FIND COORDINATES OF DOMAIN SIDES.
      YBOT=YSEND(1)
      XRIGHT=XSEND(2)
      YTOP=YSEND(3)
      XLEFT=XSEND(4)
!
!  OUTER LOOP FOR INTEGRATION IN X DIRECTION.
      HX=(XRIGHT-XLEFT)/FLOAT(NX+1)
      HY=(YTOP-YBOT)/FLOAT(NY+1)
      IXMAX=NX+2
      IYMAX=NY+2
      YINTGL(1)=PSILEFT*(YTOP-YBOT)
      YINTGL(IXMAX)=PSIRIGHT*(YTOP-YBOT)
      For each X position in turn: DO IX=2,IXMAX-1
      PSIVAL(1)=PSIBOT
      PSIVAL(IYMAX)=PSITOP
      XPOINT=XLEFT+HX*FLOAT(IX-1)
!
!  INNER LOOP FOR INTEGRATION IN Y DIRECTION.
!
!  FIRST FIND POTENTIALS AT THE INTERNAL POINTS.
      For each Y position in turn: DO IY=2,IYMAX-1
      YPOINT=YBOT+HY*FLOAT(IY-1)
      XP=XPOINT/MAXL
      YP=YPOINT/MAXL
```

```
!
!  INTEGRATE ROUND THE BOUNDARY.
      SUM=0.
      Each element in turn: DO M=1,NEL
      Each element node in turn: DO IN=1,3
      J=NODE(M,IN)
!
!  FIND THE POTENTIAL GRADIENT AT THE NODE, EITHER AS COMPUTED
!  IF POTENTIAL WAS PRESCRIBED, OR FROM THE PRESCRIBED GRADIENT
!  OR MIXED BOUNDARY CONDITIONS.
      IF(IBCE(M) == 1) DPSISTORE=DPSI(J)
      IF(IBCE(M) == 2) DPSISTORE=STORE(M)*MAXL-DPSIPIM(M,IN)
      IF(IBCE(M) == 3) DPSISTORE=(ALPHA(M)*PSIT(J)+BETA(M))*MAXL
     &                 -DPSIPIM(M,IN)
!
!  APPLY GAUSSIAN QUADRATURE.
      Each Gauss point in turn: DO IGAUSS=1,NGAUSS
      IT=1
      IC=1
      CALL JACOBI(M,IGAUSS,IT,IC)
      CALL KERNEL(XP,YP,M,IGAUSS,AK,BK)
      SFN=SF(IN,IGAUSS)
      SUM=SUM+WG(IGAUSS)*SFN*JACOB*(-AK*PSI(J)+BK*DPSISTORE)
      END DO Each Gauss point in turn
      END DO Each element node in turn
      END DO Each element in turn
      PSIIP=SUM*0.5/PI
      PSIIPT=PSIIP
!
!  ADD THE PARTICULAR INTEGRAL FOR A POISSON TYPE PROBLEM.
      IF(F1 /= 0.) THEN
        XX=XP-XC
        YY=YP-YC
        PSIIPT=PSIIPT+0.25*F1*(XX**2+YY**2)
      END IF
      PSIVAL(IY)=PSIIPT
      END DO For each Y position in turn
!
!  INTEGRATE POTENTIAL IN Y DIRECTION.
      SUM=0.
      For alternate points in the Y direction: DO IY=2,IYMAX-1,2
      SUM=SUM+PSIVAL(IY-1)+4.*PSIVAL(IY)+PSIVAL(IY+1)
      END DO For alternate points in the Y direction
      YINTGL(IX)=SUM*HY/3.
      END DO For each X position in turn
!
!  NOW INTEGRATE IN X DIRECTION.
      SUM=0.
      For alternate points in the X direction: DO IX=2,IXMAX-1,2
      SUM=SUM+YINTGL(IX-1)+4.*YINTGL(IX)+YINTGL(IX+1)
      END DO For alternate points in the X direction
      PSIINTG=SUM*HX/3.
      AREA=(XRIGHT-XLEFT)*(YTOP-YBOT)
      APSIINTG=PSIINTG/AREA
!
!  OUTPUT RESULT.
      WRITE(6,64) PSIINTG,APSIINTG,NX,NY
 64   FORMAT(/ "INTEGRAL OF POTENTIAL OVER SOLUTION DOMAIN =",E15.6,
     &       / "AREA AVERAGE OVER SOLUTION DOMAIN =",E15.6,
     &        / "NUMBERS OF INTERNAL POINTS IN X AND Y DIRECTIONS =",
     &          I5,5X,"AND", I5)
!
      RETURN
      END SUBROUTINE INTERNAL
```

This version of subprogram INTERNAL differs from that described in Section 3.3 only in the way that values of potential at internal points are computed – which now follows that established for quadratic boundary elements. The numbers of internal points in the two global co-ordinate directions to be used for numerical integration (NX and NY) are the only data that have to be read in.

Application to channel flow

The boundary conditions used are shown in Figure 3.5: a unit value of potential on the top edge, zero on the other three. The Poisson equation function is $f_1 = \pi_P$, and the dimensionless flow rate π_Q is obtained as the computed area average of the integral of potential (downstream velocity) over the channel cross-section.

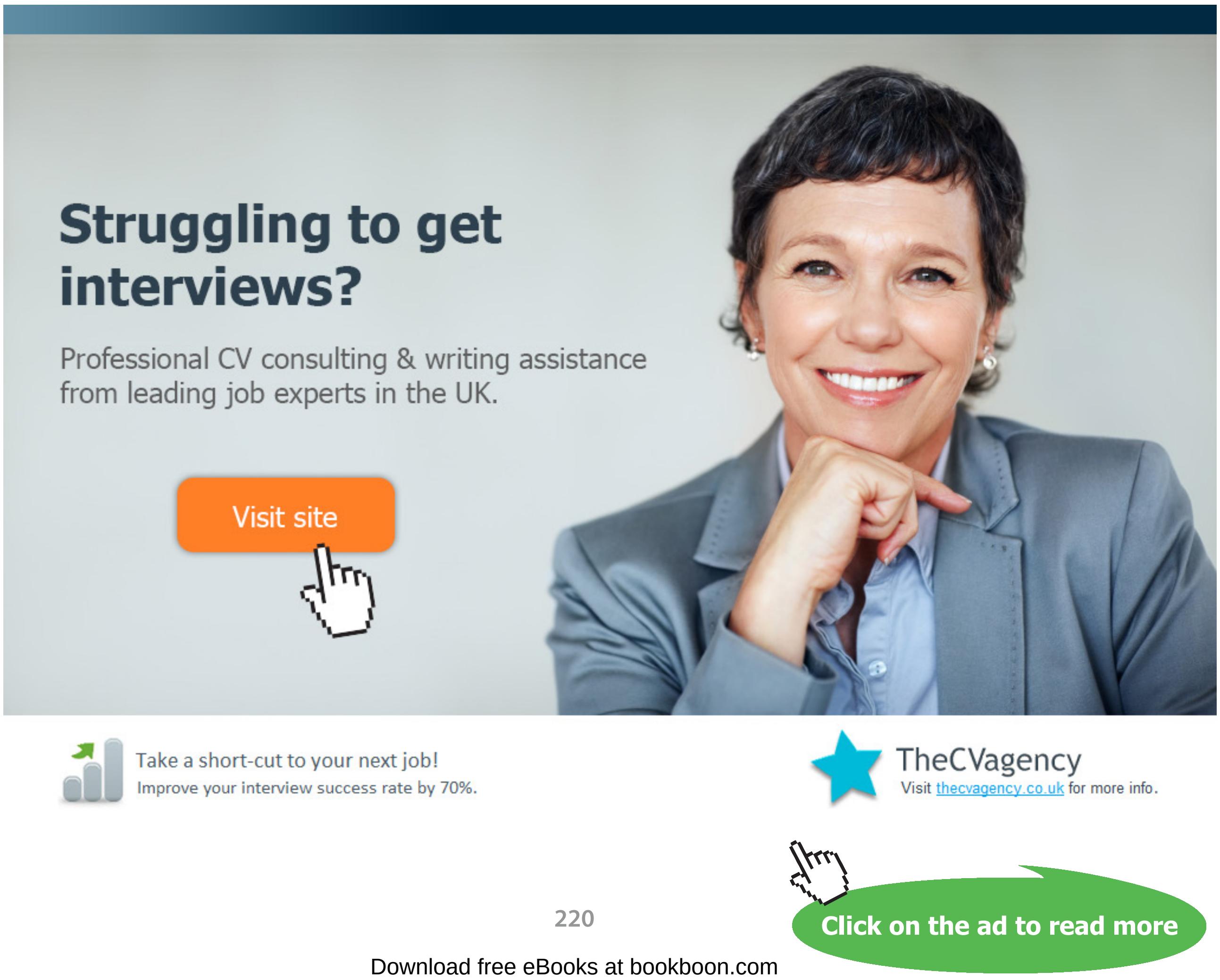
Choosing the number of internal points for integration in each of the two global co-ordinate directions (NX and NY) requires much more care than when using constant boundary elements. This is because with quadratic elements, requiring approximate numerical integration, internal points must not be located very close to domain boundaries. To avoided further complexity, for present purposes it is assumed that the same numbers of points are used in the two directions (NX=NY). Because the channel domain in the present problem is much shallower than it is wide, it is the number of internal points in the y direction (NY) that is the more significant. Consider, for example, the situation where 10 elements are used to model each of the top and bottom domain edges, a level of refinement that earlier investigations would suggest is very adequate to compute accurate results on the boundary. With a channel 5 units wide, each of the horizontal elements is 0.5 units long. If an internal point is not to approach closer than, say, a quarter of an element length to a boundary, the closest is at 0.125. The channel is only 1 unit deep, so this implies not more than 7 internal points in the y direction, which is unlikely to give an accurate value for the integral over the domain. Taking, for example, $\pi_P = -2$, for which the exact dimensionless flow rate is $\pi_Q = 0.5914$, Table 4.8 shows the flow rates obtained when using 10 quadratic elements on each of the four domain edges, and a range of numbers of internal points

NX and NY	π_Q
9	0.5859
15	0.5898
19	0.5850
29	0.5829

Table 4.8 Effect of internal point refinement on the computed domain integral

The first feature to note is the relatively poor accuracy using so many boundary elements. While increasing the number of internal points does offer some improvement, beyond about 15 points the accuracy declines again. Improvement in integration accuracy is offset by loss of accuracy in computing values of potential close to the upper and lower edges of the domain. This behaviour is typical of other values of π_P and other levels of boundary element mesh refinement: optimum accuracy of flow rate is obtained when the number of internal points (in each direction) is about 50% greater than the number of quadratic elements on each edge of the domain. This is an empirical observation, applicable only to the current problem, the determining factor being the closeness of the internal points to the boundary.

The only way to improve the computed flow rate is to refine the boundary element mesh, which allows more internal points to be used. In order to permit a direct comparison with Table 3.9 for constant elements, Table 4.9 shows the results obtained for the relationship between π_Q and π_P for the same range of number of nodes per edge of the domain from 7 to 241. The origin chosen for the particular integral function is the centre of the channel, at (2.5, 0.5).


Struggling to get interviews?
Professional CV consulting & writing assistance from leading job experts in the UK.
Visit site
Take a short-cut to your next job!
Improve your interview success rate by 70%.
TheCVagency
Visit thecvagency.co.uk for more info.
Click on the ad to read more

Nodes per edge	NX and NY	$\pi_P = -2$	$\pi_P = 0$	$\pi_P = 2$	$\pi_P = 4$
Exact π_Q		0.5914	0.4457	0.3000	0.1544
7	5	0.5588	0.4251	0.2915	0.1579
15	11	0.5897	0.4425	0.2953	0.1481
31	23	0.5908	0.4449	0.2991	0.1532
61	45	0.5911	0.4455	0.2998	0.1542
121	91	0.5913	0.4457	0.3000	0.1544
241	181	0.5914	0.4457	0.3000	0.1544

Table 4.9 Dimensionless flow rates for $H / W = 0.2$

Comparing Tables 3.9 and 4.9, it is clear that, for the same numbers of nodes per edge, the results for constant and quadratic elements are very similar, particularly for the more refined meshes. Making comparisons for the same number of nodes per edge is appropriate, because it is the number of nodes (rather than elements) which determines the size of the problem, in particular the number of equations to be solved. There is of course a slight difference between constant and quadratic elements in the total number of nodes for a given number of nodes per edge because quadratic elements share nodes at the four corners. Both Tables 3.9 and 4.9 show that four significant figure accuracy can be obtained, but only at the expense of using a great many boundary elements. The advantages of quadratic elements are lost, due to their limitations over permissible closeness of internal points to the boundary.

4.4 An Example: Heat Conduction in a Domain of Complex Shape

As a final example, involving varieties of both geometric features and boundary conditions, consider the heat conduction in the domain shown in Figure 4.2. This represents the cross-section of a long uniform component. The letters A to P in the figure label particular points on the domain boundary: either corners, changes in geometry from straight to curved, or in the cases of M, N, O and P at the ends of the horizontal hole diameters. An origin for global co-ordinates can be chosen to be at the bottom left hand corner of the figure, at the centre of arc LA. The two circular holes of radii 15 and 10 mm are centred at (35, 75) and (115, 85), respectively, and the co-ordinates (in mm) of the labelled points are as follows:

A at (30, 0)	B at (65, 0)	C at (65, 45)	D at (95, 45)	E at (95, 0)
F at (125, 0)	G at (140, 15)	H at (140, 120)	I at (110, 120)	J at (30, 120)
K at (0, 120)	L at (0, 30)	M at (50, 75)	N at (20, 75)	O at (125, 85)
P at (105, 85)				

Thermal conductivity of the material is 1.2 W/m°C. Heat is conducted into the domain at a uniformly distributed rate over the surface BCDE, the total rate being 500W per metre of length in the direction normal to the domain. The curved surface IJ is maintained at a uniform temperature of 150°C. The two circular channels represented by the holes in the cross-section carry a cooling fluid with a bulk temperature of 10°C, and the surface heat transfer coefficient for both channels is 200 W/m^2°C. Surface LA is thermally insulated. The remaining parts of the boundary are exposed to the atmosphere at a temperature of 20°C, and the surface heat transfer coefficient is 20 W/m^2°C. The problem is to find the rates of heat transfer to the cooling fluids in the two channels.

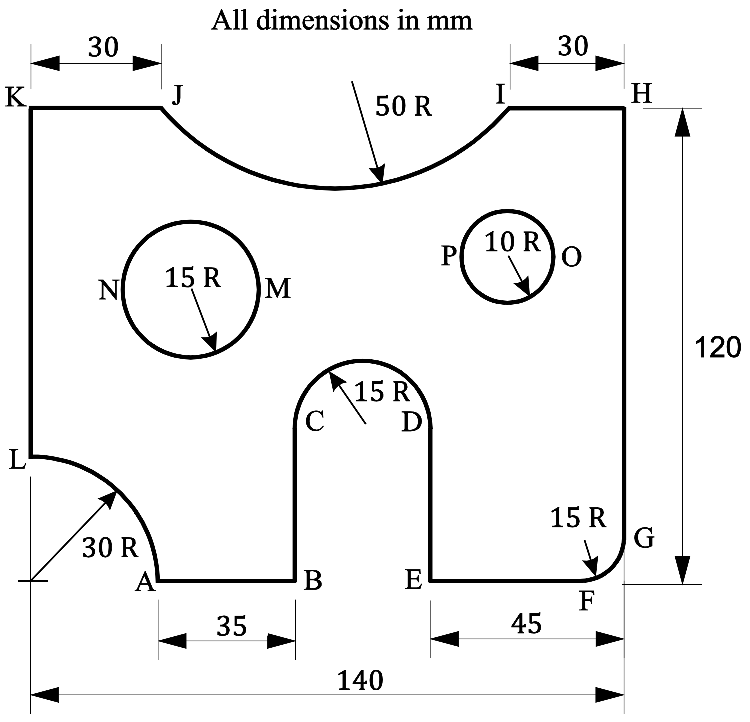


Figure 4.2 Heat conduction problem

Starting from point A, the boundary segments are defined as follows:

AB: 1	BC: 2	CD: 3	DE: 4	EF: 5	FG: 6	GH: 7
HI: 8	IJ: 9	JK: 10	KL: 11	LA: 12	MN: 13	NM: 14
OP: 15	PO: 16					

At least in the first instance, two quadratic elements per quadrant are applied to arcs CD, FG and LA, also to the two holes, and three elements to arc IJ. Segments AB, BC, DE, EF, HI and JK each have one element, while GH and KL have two.

The specified heat transfer rate must be translated into a prescribed temperature gradient over the relevant part of the boundary. The perimeter of semi-circle CD is $\pi \times 15 = 47.12$ mm, or 0.04712 m. Adding in 45 mm each for BC and DE gives a total length for BCDE of 0.1371 m. The heat transfer rate over the surface is therefore $500/0.1371 = 3647$ W/m^2, and the temperature gradient is $3647/1.2 = 3039$°C/m.

Comparing Equations 1.57 and 2.32, the coefficients α and β for a mixed boundary condition in a heat conduction problem can be expressed in terms of the surface heat transfer coefficient, thermal conductivity, and bulk temperature of the cooling medium as

$$\alpha = -\frac{h}{k}, \qquad \beta = \frac{hT_\infty}{k} \tag{4.21}$$

So, for the circular cooling channels

$$\alpha = -\frac{200}{1.2} = -166.7 \;\; 1/\text{m}, \qquad \beta = \frac{200\times 10}{1.2} = 1667 \quad {}^\circ\text{C/m} \tag{4.22}$$

and for the surfaces exposed to atmosphere

$$\alpha = -\frac{20}{1.2} = -16.67 \;\; 1/\text{m}, \qquad \beta = \frac{20\times 20}{1.2} = 333.3 \quad {}^\circ\text{C/m} \tag{4.23}$$

The input DATA file is as follows. There are three boundaries. The first is made up of 12 segments, whose ends are the points labelled A to L in Figure 4.2, and whose co-ordinates are given in metres measured from an origin at the centre of arc LA. The second and third boundaries are the two circular holes, each made up of two segments with end points M to P.

```
HEAT CONDUCTION PROBLEM
0.                                              (Laplace problem)
3                                               (Number of boundaries)
12                                              (Number of segments on 1st boundary)
0.030 0. 0.065 0. 0.065 0.045 0.095 0.045 0.095 0.
0.125 0. 0.140 0.015 0.140 0.120 0.110 0.120
0.030 0.120 0. 0.120 0. 0.030                   (Co-ordinates of segment ends)
0. 1 1.
0. 1 1.                                         (Radii of curvature,
-0.015 4 1.                                     number of elements,
0. 1 1.                                         and element length ratio
0. 1 1.                                         for each of the 12 segments in turn)
0.015 2 1.
0. 2 1.
0. 1 1.
-0.050 3 1.
0. 1 1.
0. 2 1.
-0.030 2 1.
2                                               (Number of segments on 2nd boundary)
0.050 0.075 0.020 0.075                         (Co-ordinates of segment ends)
-0.015 4 1.                                     (Segment element data)
-0.015 4 1.
2                                               (Number of segments on 3rd boundary)
0.125 0.085 0.105 0.085                         (Co-ordinates of segment ends)
-0.010 4 1.                                     (Segment element data)
-0.010 4 1.
1 3 11                                          (Numbers of boundary conditions)
9 150.                                          (Segment and prescribed potential)
2 3039.                                         (Segment numbers and
3 3039.                                         prescribed gradients)
4 3039.
1 -16.67 333.3
```

```
5 -16.67 333.3                  (Mixed boundary conditions:
6 -16.67 333.3                  segment number, with α and β,
7 -16.67 333.3                  for 11 segments)
8 -16.67 333.3
10 -16.67 333.3
11 -16.67 333.3
13 -166.7 1667.
14 -166.7 1667.
15 -166.7 1667.
16 -166.7 1667.
0                               (No internal points)
```

Segment 9 (IJ) is subject to prescribed temperature (potential), while segments 2, 3, and 4 (surface BCDE) have nonzero temperature gradient prescribed. Segment 12 (LA) does not appear in the boundary conditions because it is subject to zero temperature gradient. The remaining 11 segments have mixed boundary conditions, defined by Equations 4.22 and 4.23. No results for internal points are required.

The mesh data output file MESHRES is as follows. There are 37 elements, with 74 nodal points.

```
  GEOMETRIC DATA FOR THE MESH

           NUMBER OF ELEMENTS = 37

           NUMBER OF NODAL POINTS = 74

  COORDINATES OF THE NODAL POINTS

       I    X           Y           I     X           Y
       1 0.3000E-01  0.1074E-08  2 0.4750E-01  0.0000E+00
       3 0.6500E-01  0.0000E+00  4 0.6500E-01  0.2250E-01
       5 0.6500E-01  0.4500E-01  6 0.6614E-01  0.5074E-01
       7 0.6939E-01  0.5561E-01  8 0.7426E-01  0.5886E-01
       9 0.8000E-01  0.6000E-01  10 0.8574E-01  0.5886E-01
      11 0.9061E-01  0.5561E-01  12 0.9386E-01  0.5074E-01
      13 0.9500E-01  0.4500E-01  14 0.9500E-01  0.2250E-01
      15 0.9500E-01  0.0000E+00  16 0.1100E+00  0.0000E+00
      17 0.1250E+00  0.0000E+00  18 0.1307E+00  0.1142E-02
      19 0.1356E+00  0.4393E-02  20 0.1389E+00  0.9260E-02
      21 0.1400E+00  0.1500E-01  22 0.1400E+00  0.4125E-01
```

```
23 0.1400E+00  0.6750E-01  24 0.1400E+00  0.9375E-01
25 0.1400E+00  0.1200E+00  26 0.1250E+00  0.1200E+00
27 0.1100E+00  0.1200E+00  28 0.9898E-01  0.1093E+00
29 0.8521E-01  0.1024E+00  30 0.7000E-01  0.1000E+00
31 0.5479E-01  0.1024E+00  32 0.4102E-01  0.1093E+00
33 0.3000E-01  0.1200E+00  34 0.1500E-01  0.1200E+00
35 0.0000E+00  0.1200E+00  36 0.0000E+00  0.9750E-01
37 0.0000E+00  0.7500E-01  38 0.0000E+00  0.5250E-01
39 0.0000E+00  0.3000E-01  40 0.1148E-01  0.2772E-01
41 0.2121E-01  0.2121E-01  42 0.2772E-01  0.1148E-01
43 0.5000E-01  0.7500E-01  44 0.4886E-01  0.6926E-01
45 0.4561E-01  0.6439E-01  46 0.4074E-01  0.6114E-01
47 0.3500E-01  0.6000E-01  48 0.2926E-01  0.6114E-01
49 0.2439E-01  0.6439E-01  50 0.2114E-01  0.6926E-01
51 0.2000E-01  0.7500E-01  52 0.2114E-01  0.8074E-01
53 0.2439E-01  0.8561E-01  54 0.2926E-01  0.8886E-01
55 0.3500E-01  0.9000E-01  56 0.4074E-01  0.8886E-01
57 0.4561E-01  0.8561E-01  58 0.4886E-01  0.8074E-01
59 0.1250E+00  0.8500E-01  60 0.1242E+00  0.8117E-01
61 0.1221E+00  0.7793E-01  62 0.1188E+00  0.7576E-01
```

```
    63 0.1150E+00  0.7500E-01  64 0.1112E+00  0.7576E-01
    65 0.1079E+00  0.7793E-01  66 0.1058E+00  0.8117E-01
    67 0.1050E+00  0.8500E-01  68 0.1058E+00  0.8883E-01
    69 0.1079E+00  0.9207E-01  70 0.1112E+00  0.9424E-01
    71 0.1150E+00  0.9500E-01  72 0.1188E+00  0.9424E-01
    73 0.1221E+00  0.9207E-01  74 0.1242E+00  0.8883E-01

ELEMENT NODE NUMBERS

  M ND1   ND2   ND3         M     ND1   ND2   ND3
  1 1     2     3           2     3     4     5
  3 5     6     7           4     7     8     9
  5 9     10    11          6     11    12    13
  7 13    14    15          8     15    16    17
  9 17    18    19          10    19    20    21
 11 21    22    23          12    23    24    25
 13 25    26    27          14    27    28    29
 15 29    30    31          16    31    32    33
 17 33    34    35          18    35    36    37
 19 37    38    39          20    39    40    41
 21 41    42    1           22    43    44    45
 23 45    46    47          24    47    48    49
 25 49    50    51          26    51    52    53
 27 53    54    55          28    55    56    57
 29 57    58    43          30    59    60    61
 31 61    62    63          32    63    64    65
 33 65    66    67          34    67    68    69
 35 69    70    71          36    71    72    73
 37 73    74    59
```

The RESULTS file is as follows.

```
QUADRATIC BOUNDARY ELEMENT SOLUTION FOR TWO DIMENSIONAL POTENTIAL PROBLEM

HEAT CONDUCTION PROBLEM

LAPLACE EQUATION

PRESCRIBED POTENTIAL BOUNDARY CONDITIONS
```

```
POTENTIAL = 0.1500E+03 ON ELEMENTS 14 TO 16

PRESCRIBED POTENTIAL GRADIENT BOUNDARY CONDITIONS

GRADIENT = 0.3039E+04 ON    ELEMENTS    2     TO    2

GRADIENT = 0.3039E+04 ON    ELEMENTS    3     TO    6

GRADIENT = 0.3039E+04 ON    ELEMENTS    7     TO    7

PRESCRIBED MIXED BOUNDARY CONDITIONS

ALPHA = -0.1667E+02 BETA = 0.3333E+03 ON ELEMENTS 1 TO 1

ALPHA = -0.1667E+02 BETA = 0.3333E+03 ON ELEMENTS 8 TO 8

ALPHA = -0.1667E+02 BETA = 0.3333E+03 ON ELEMENTS 9 TO 10

ALPHA = -0.1667E+02 BETA = 0.3333E+03 ON ELEMENTS 11 TO 12

ALPHA = -0.1667E+02 BETA = 0.3333E+03 ON ELEMENTS 13 TO 13

ALPHA = -0.1667E+02 BETA = 0.3333E+03 ON ELEMENTS 17 TO 17

ALPHA = -0.1667E+02 BETA = 0.3333E+03 ON ELEMENTS 18 TO 19

ALPHA = -0.1667E+03 BETA = 0.1667E+04 ON ELEMENTS 22 TO 25

ALPHA = -0.1667E+03 BETA = 0.1667E+04 ON ELEMENTS 26 TO 29

ALPHA = -0.1667E+03 BETA = 0.1667E+04 ON ELEMENTS 30 TO 33

ALPHA = -0.1667E+03 BETA = 0.1667E+04 ON ELEMENTS 34 TO 37
```

```
NODAL POINT POTENTIALS AND POTENTIAL GRADIENTS

  I      PSI                DPSI            UNX          UNY
  1  0.838513E+02      0.000000E+00     -0.7021E+00 -0.7121E+00
  2  0.967611E+02     -0.127971E+04     -0.3068E-07 -0.1000E+01
  3  0.133967E+03      0.303900E+04      0.7071E+00 -0.7071E+00
  4  0.150549E+03      0.303900E+04      0.1000E+01  0.0000E+00
  5  0.130908E+03      0.303900E+04      0.1000E+01 -0.7036E-02
  6  0.123689E+03      0.303900E+04      0.9239E+00 -0.3827E+00
  7  0.123411E+03      0.303900E+04      0.7071E+00 -0.7071E+00
  8  0.127642E+03      0.303900E+04      0.3827E+00 -0.9239E+00
  9  0.130472E+03      0.303900E+04      0.6357E-06 -0.1000E+01
 10  0.129789E+03      0.303900E+04     -0.3827E+00 -0.9239E+00
 11  0.129134E+03      0.303900E+04     -0.7071E+00 -0.7071E+00
 12  0.132750E+03      0.303900E+04     -0.9239E+00 -0.3827E+00
 13  0.140975E+03      0.303900E+04     -0.1000E+01 -0.7034E-02
 14  0.156014E+03      0.303900E+04     -0.1000E+01  0.0000E+00
 15  0.136025E+03     -0.193424E+04     -0.7071E+00 -0.7071E+00
 16  0.101275E+03     -0.135495E+04      0.0000E+00 -0.1000E+01
 17  0.802578E+02     -0.100460E+04      0.7035E-02 -0.1000E+01
 18  0.760928E+02     -0.935168E+03      0.3827E+00 -0.9239E+00
```

```
  19  0.739148E+02      -0.898860E+03      0.7071E+00 -0.7071E+00
  20  0.730442E+02      -0.884347E+03      0.9239E+00 -0.3827E+00
  21  0.733541E+02      -0.889513E+03      0.1000E+01 -0.7034E-02
  22  0.682552E+02      -0.804514E+03      0.1000E+01 -0.2023E-06
  23  0.448672E+02      -0.414637E+03      0.1000E+01 -0.2023E-06
  24  0.381535E+02      -0.302719E+03      0.1000E+01  0.0000E+00
  25  0.505920E+02      -0.510069E+03      0.7071E+00  0.7071E+00
  26  0.674110E+02      -0.790441E+03     0.0000E+00   0.1000E+01
  27  0.150000E+03       0.131650E+05     -0.7958E+00  0.6056E+00
  28  0.150000E+03       0.494709E+04     -0.5796E+00  0.8149E+00
  29  0.150000E+03       0.409609E+04     -0.3042E+00  0.9526E+00
  30  0.150000E+03       0.320071E+04      0.3492E-06  0.1000E+01
  31  0.150000E+03       0.594284E+04      0.3042E+00  0.9526E+00
  32  0.150000E+03       0.525909E+04      0.5796E+00  0.8149E+00
  33  0.150000E+03       0.123646E+05      0.7958E+00  0.6056E+00
  34  0.719538E+02      -0.866170E+03     -0.7081E-06  0.1000E+01
  35  0.539120E+02      -0.565413E+03     -0.7071E+00  0.7071E+00
  36  0.469374E+02      -0.449147E+03     -0.1000E+01  0.0000E+00
  37  0.311227E+02      -0.185515E+03     -0.1000E+01  0.0000E+00
  38  0.370928E+02      -0.285037E+03     -0.1000E+01  0.0000E+00
  39  0.470644E+02       0.000000E+00     -0.7121E+00 -0.7021E+00
  40  0.571549E+02       0.000000E+00     -0.3827E+00 -0.9239E+00
  41  0.747607E+02       0.000000E+00     -0.7071E+00 -0.7071E+00
  42  0.877977E+02       0.000000E+00     -0.9239E+00 -0.3827E+00
  43  0.427965E+02      -0.546718E+04     -0.1000E+01  0.2030E-06
  44  0.407786E+02      -0.513080E+04     -0.9239E+00  0.3827E+00
  45  0.384964E+02      -0.475034E+04     -0.7071E+00  0.7071E+00
  46  0.348074E+02      -0.413539E+04     -0.3827E+00  0.9239E+00
  47  0.302071E+02      -0.336853E+04      0.2133E-06  0.1000E+01
  48  0.256852E+02      -0.261473E+04      0.3827E+00  0.9239E+00
  49  0.221622E+02      -0.202744E+04      0.7071E+00  0.7071E+00
  50  0.202331E+02      -0.170585E+04      0.9239E+00  0.3827E+00
  51  0.204366E+02      -0.173978E+04      0.1000E+01  0.5635E-07
  52  0.231309E+02      -0.218892E+04      0.9239E+00 -0.3827E+00
  53  0.284982E+02      -0.308366E+04      0.7071E+00 -0.7071E+00
  54  0.362383E+02      -0.437392E+04      0.3827E+00 -0.9239E+00
  55  0.443025E+02      -0.571823E+04      0.2105E-06 -0.1000E+01
  56  0.490269E+02      -0.650578E+04     -0.3827E+00 -0.9239E+00
  57  0.489175E+02      -0.648755E+04     -0.7071E+00 -0.7071E+00
  58  0.458112E+02      -0.596972E+04     -0.9239E+00 -0.3827E+00
```

```
59  0.239710E+02      -0.232897E+04     -0.1000E+01  0.6399E-06
60  0.246855E+02      -0.244808E+04     -0.9239E+00  0.3827E+00
61  0.271660E+02      -0.286157E+04     -0.7071E+00  0.7071E+00
62  0.308030E+02      -0.346786E+04     -0.3827E+00  0.9239E+00
63  0.349101E+02      -0.415251E+04     -0.3134E-06  0.1000E+01
64  0.387237E+02      -0.478824E+04      0.3827E+00  0.9239E+00
65  0.418642E+02      -0.531177E+04      0.7071E+00  0.7071E+00
66  0.444542E+02      -0.574351E+04      0.9239E+00  0.3827E+00
67  0.469923E+02      -0.616662E+04      0.1000E+01 -0.6352E-06
68  0.492539E+02      -0.654362E+04      0.9239E+00 -0.3827E+00
69  0.499708E+02      -0.666313E+04      0.7071E+00 -0.7071E+00
70  0.475250E+02      -0.625542E+04      0.3827E+00 -0.9239E+00
71  0.419051E+02      -0.531859E+04     -0.6534E-06 -0.1000E+01
72  0.349964E+02      -0.416691E+04     -0.3827E+00 -0.9239E+00
73  0.291235E+02      -0.318788E+04     -0.7071E+00 -0.7071E+00
74  0.253855E+02      -0.256476E+04     -0.9239E+00 -0.3827E+00

POTENTIAL FLOWS INTO DOMAIN ACROSS BOUNDARY SEGMENTS

SEGMENT         FLOW
 1         -0.471524E+02
 2          0.136755E+03
 3          0.143155E+03
 4          0.136755E+03
 5         -0.417933E+02
 6         -0.215332E+02
 7         -0.582556E+02
 8         -0.291952E+02
 9          0.512356E+03
 10        -0.309865E+02
 11        -0.324333E+02
 12         0.000000E+00
 13        -0.160658E+03
 14        -0.223623E+03
 15        -0.129467E+03
 16        -0.153033E+03

TOTAL POTENTIAL FLOW INTO DOMAIN = 0.929021E+03

TOTAL POTENTIAL FLOW OUT OF DOMAIN = 0.928131E+03
```

Firstly, some points of detail concerning the normals at the ends of segments. At points I and J (nodes 27 and 33), where there is a change from prescribed temperature to mixed boundary conditions, the local normals are normal to the curved surface, the directions of the unknown temperature gradients. On the other hand, the normals at corner points A, B, E, H, K and L (nodes 1, 3, 15, 25, 35, and 39) are in the average directions of the normals to the segments that meet there, which have gradient type boundary conditions.

The total potential flows (conductive heat fluxes) into and out of the domain of this Laplace type problem balance to within 0.1%. The potential flow (integration of temperature gradients) into the domain over BCDE (adding flows for segments 2, 3 and 4) is 416.7 ^{o}C. Multiplying by the thermal conductivity of 1.2 W/m^{o}C gives the prescribed value of 500.0 W/m. There is also heat input over IJ (segment 9) of some 614 W/m. The cooling rate into the large hole (segments 13 and 14) is 160.7 + 223.6 = 384.3 ^{o}C, or 461.2 W/m, while the rate into the small hole (segments 15 and 16) is 339.0 W/m. The effects of mesh refinement can be explored by progressively doubling the number of elements on each of the segments, with the results shown in Table 4.10.

Number of elements	Large hole cooling, W/m	Small hole cooling W/m
37	461.2	339.0
74	459.9	336.7
148	459.5	336.1
296	459.3	335.9

Table 4.10 Effect of mesh refinement on cooling rates

Refining the mesh has only a modest effect on the computed cooling rates, and for practical purposes the rates for the two holes could be taken as 459 and 336 W/m, respectively.

4.5 Discussion

It is clear that quadratic boundary elements applied to potential problems offer much better accuracy than do constant elements. Under favourable conditions, as few as a tenth as many nodes need be employed. This applies not only in problems where the boundary geometry is curved, and quadratic elements have an obvious advantage, but also in problems involving straight boundaries, because the variables (potential and potential gradient) are also allowed to vary quadratically over each element.

But, constant elements do offer significant benefits in problems where either values of potential within the solution domain are required to be computed at points close to the domain boundary, or opposite sides of the domain are close together. This is because constant elements use only analytical integration, which is always more accurate than the numerical integration required by quadratic elements. As a consequence, in problems where the integral of potential over the solution domain is required, very refined meshes of quadratic boundary elements are required to maintain internal point accuracy for domain integration, and their advantages over constant elements are lost.

Problems

4.1 Solve Problem 3.1 using program BEM2PQ.

4.2 Solve Problem 3.2 using program BEM2PQ.

4.3 Solve Problem 3.3 using program BEM2PQ.

4.4 Solve Problem 3.4 using program BEM2PQ.

4.5 Solve Problem 3.5 using program BEM2PQ.

4.6 Solve Problem 3.6 using program BEM2PQ.

4.7 Solve Problem 3.7 using program BEM2PQ.

4.8 Solve Problem 3.8 using program BEM2PQ. Why is the computed maximum temperature slightly higher than that obtained using constant boundary elements?

4.9 Solve Problem 3.9 using program BEM2PQ.

4.10 Solve Problem 3.10 using program BEM2PQ.

4.11 Figure 4.3 shows a small part ABCD of an engine component where a fin is used to promote convection cooling to the surrounding atmosphere. There are other similar fins both above and below the region show, so that the lines AB and CD can be regarded as representing planes of symmetry. The surface represented by AD is maintained at a temperature of 300°C, and the thermal conductivity of the steel from which the component is made is 50 W/m°C. The heat transfer coefficient of the surface represented by BC, and including the fin, which is exposed to the atmosphere at a temperature of 25 is 30 W/m^2. Use program BEM2PQ to find the percentage increases in cooling rate obtained using fin lengths L of (a) 40mm, (b) 80 mm, and (c) 120 mm, relative to not using a fin at all $(L = 0)$. Find also the temperature at the tip of the fin in each case.

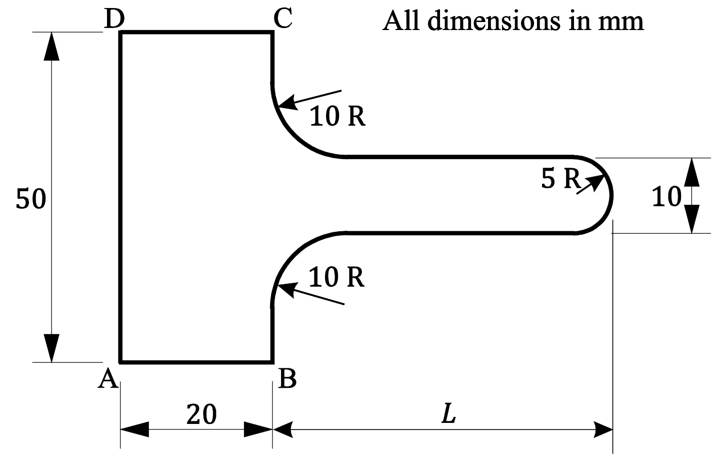


Figure 4.3 Cooling fin problem

4.12 A number of pipes are used to carry hot water at 90°C underground. The pipes are 100 mm external diameter laid parallel to each other at a distance 2 m apart, and with their centres 1 m below ground level. The thermal conductivity of the ground can be taken to be 1.25 W/m°C, and the surface heat transfer coefficient to the air at 10°C is 5 W/m^2°C. Use program BEM2PQ to find both the rate of heat loss from the pipe and the maximum temperature of the ground surface.

Appendix A: Gaussian Quadrature

The aim of this appendix is to explain the Gaussian quadrature method of numerical integration, particularly as applied to boundary elements.

When a function cannot be integrated analytically, some form of numerical integration or quadrature must be used. Two of the simplest methods are the familiar trapezium and Simpson's rules. Given a function $f(x)$, values of the function are obtained for a number of values of the independent variable x over the required range of integration, and an approximate value of the integral obtained from them. An important feature (and limitation) of these simple methods is that the values of the independent variable at which the function is sampled are often taken at constant intervals, and certainly little attention is paid to the optimum choice of sampling points.

If there is no restriction on the positions at which the function can be evaluated, then there is a class of numerical integration procedures referred to as Gaussian quadrature, which is to be preferred. The advantage over the trapezium and Simpson's rules is that greater accuracy for a given number of function evaluations, and therefore of overall computational effort, can be obtained.

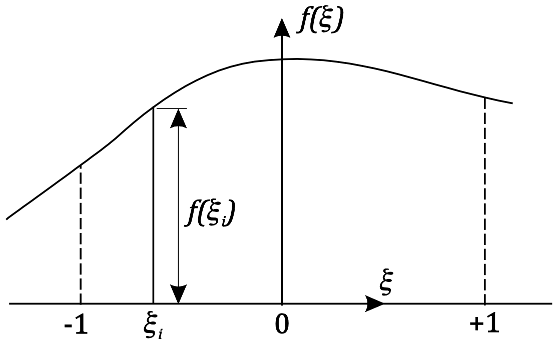


Figure A.1 Typical function to be integrated numerically

Figure A.1 shows a typical function to be integrated: the area between the curve $f(\xi)$ and the ξ axis between the limits $\xi = -1$ and $\xi = +1$ is required. It is convenient to transform the actual independent variable, such as x, into a dimensionless variable with these limits. This is the reason for introducing the local intrinsic co-ordinate ξ for quadratic boundary elements. Assuming that the function is evaluated at n points, known as the Gauss points, the required integral can be expressed approximately as

$$\int_{-1}^{+1} f(\xi)\mathrm{d}\xi \approx \sum_{p=1}^{n} \omega_p f(\xi_p) \tag{A.1}$$

where ω_p the are weighting factors. Simpson's rule (defined in Equation 3.49) is actually a particular form of this result in which, for a single application of the rule

$$\begin{gathered} n = 3 \\ \xi_1 = -1 \qquad \xi_2 = 0 \qquad \xi_3 = +1 \\ \omega_1 = \omega_3 = \frac{1}{3} \qquad \omega_2 = \frac{4}{3} \end{gathered} \tag{A.2}$$

Given the freedom to choose the ξ_p, however, the values of these and the corresponding weighting factors can be optimised for the integration of polynomial functions to give better accuracy. These are now well established and are available for a wide range of n, the number of Gauss points (for example, Stroud & Secrest 1966).

For any numerical integration, choosing the number of Gauss points involves a compromise between accuracy and the amount of computation involved. In boundary element methods the number of points is normally at least $n = 4$. In this book a higher value of $n = 8$ is preferred, for which the values of ξ and ω are

$$\begin{aligned} \xi_8 = -\xi_1 &= 0.9602898564 \qquad \omega_1 = \omega_8 = 0.1012285362 \\ \xi_7 = -\xi_2 &= 0.7966664774 \qquad \omega_2 = \omega_7 = 0.2223810344 \\ \xi_6 = -\xi_3 &= 0.5255324099 \qquad \omega_3 = \omega_6 = 0.3137066458 \\ \xi_5 = -\xi_4 &= 0.1834346424 \qquad \omega_4 = \omega_5 = 0.3626837833 \end{aligned} \tag{A.3}$$

to ten decimal places.

Gaussian quadrature formulae have also been worked out for functions which involve particular forms of functions as multipliers, including cases of multipliers which are singular within the range of integration. One such case which is very relevant to boundary element methods is

$$\int_0^1 \ln\left(\frac{1}{\eta}\right) f(\eta)\mathrm{d}\eta \approx \sum_{p=1}^{n} \omega_p^* f(\eta_p^*) \tag{A.4}$$

the numerical values for $n = 8$ being

$$\begin{aligned}
\eta_1^* &= 0.013320244 \qquad & \omega_1^* &= 0.164416605 \\
\eta_2^* &= 0.079750429 \qquad & \omega_2^* &= 0.237525610 \\
\eta_3^* &= 0.197871029 \qquad & \omega_3^* &= 0.226841984 \\
\eta_4^* &= 0.354153994 \qquad & \omega_4^* &= 0.175754079 \\
\eta_5^* &= 0.529458575 \qquad & \omega_5^* &= 0.112924030 \\
\eta_6^* &= 0.701814530 \qquad & \omega_6^* &= 0.057872211 \\
\eta_7^* &= 0.849379320 \qquad & \omega_7^* &= 0.020979074 \\
\eta_8^* &= 0.953326450 \qquad & \omega_8^* &= 0.003686407
\end{aligned} \tag{A.5}$$

Reference

Stroud, A.H. & Secrest, D. 1966, *Gaussian Quadrature Formulas*, Prentice-Hall.

Appendix B: Gaussian Elimination

The aim of this appendix is to explain Gaussian elimination, which is a direct method for solving sets of simultaneous linear algebraic equations of the general form

$$\begin{array}{l} a_{11}x_1 + a_{12}x_2 + \cdots \cdots \cdots \cdots \cdots \cdots \cdots \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots \cdots \cdots \cdots \cdots \cdots \cdots \cdots + a_{2n}x_n = b_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots \cdots \cdots \cdots \cdots \cdots \cdots \cdots + a_{nn}x_n = b_n \end{array} \tag{B.1}$$

where $x_1, x_2, \cdots, x_n$ are the unknowns, and the coefficients a_{ij} and b_i are all known constants. This set of equations can be expressed in matrix form as follows

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \cdot \\ \cdot \\ b_n \end{bmatrix} \tag{B.2}$$

$$[A][x] = [b] \tag{B.3}$$

The unknowns are successively eliminated by algebraic manipulation. The first equation can be used to eliminate x_1 from the remaining $n-1$ equations. The modified second equation is then used to eliminate x_2 from the remaining $n-2$ equations, and so on until the last equation contains only x_n. Thus may be found, followed by all the other unknowns, by back substitution. Let the coefficients of $[A]$ and $[b]$ shown in Equations B.1 and B.2 be given the notation $a_{ij}^{(1)}$ and $b_i^{(1)}$. After the *n*th elimination, the modified coefficients are

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \emptyset a_{kj}^{(k)} \tag{B.4}$$

$$b_i^{(k+1)} = b_i^{(k)} - \emptyset b_k^{(k)}$$

where $\emptyset = a_{ik}^{(k)}/a_{kk}^{(k)}$; $i = k+1, k+2, \dots, n$ and $j = k, k+1, \dots, n$. Note that the column vector $[b]$ is treated just like a column of $[A]$, and advantage can be taken of this fact to simplify the computer programming of the process. The final set of equations is

$$\begin{aligned} a_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + \dots\dots\dots\dots + a_{1n}^{(1)}x_n &= b_1^{(1)} \\ a_{22}^{(2)}x_2 + \dots\dots\dots\dots + a_{2n}^{(2)}x_n &= b_2^{(2)} \\ \dots\dots\dots\dots\dots & \\ \dots\dots\dots\dots\dots & \\ a_{nn}^{(n)}x_n &= b_n^{(n)} \end{aligned} \tag{B.5}$$

Expressed in matrix terminology, the elimination process triangularises $[A]$. The unknowns are obtained in reverse order

$$x_n = b_n^{(n)}/a_{nn}^{(n)} \tag{B.6}$$

$$x_i = \left(b_i^{(i)} - \sum_{j=i+1}^{n} a_{ij}^{(i)}x_j \right)/a_{ii}^{(i)}$$

where $i = n-1, n-2, \dots., 1$.

A great many arithmetic operations are involved in solving large sets of equations by elimination. Any errors introduced, such as roundoff errors caused by representing numbers with only a finite number of significant figures, tend to be magnified and may become unacceptably large. Equations B.4 show that the elimination process involves many multiplications by the factors $\emptyset$. In order to minimize the effects of roundoff errors, these factors should be made as small as possible, and certainly less than one. Therefore, the pivotal coefficient $a_{kk}^{(k)}$ should be the largest coefficient in the leading column of the remaining submatrix

$$\left|a_{kk}^{(k)}\right| > \left|a_{ik}^{(k)}\right| \quad \text{for } i = k+1, k+2, \dots, n \tag{B.7}$$

This condition also helps to avoid division by zero in Equations B.6, and can be achieved by a process known as partial pivoting. Immediately before each elimination, the leading column is searched for the largest coefficient. By interchanging equations, this can be made the pivotal coefficient to satisfy Equation B.7. The idea of partial pivoting can be extended to searching the whole of the remaining submatrix for the largest coefficient. Such complete pivoting involves interchanging both rows and columns, and is more difficult to program. Since it offers only modest advantages in terms of accuracy over partial pivoting, it is rarely used.

Another refinement which helps to minimize the effects of roundoff errors is to scale the equations to make their coefficients similar in magnitude. One way to do this is to normalize each equation so that the largest coefficient in each row of $[A]$ is of magnitude one. Scaling can be particularly important when corresponding coefficients in different equations differ by several orders of magnitude.

Any given set of equations may provide either duplicate (for example, two equations identical) or inconsistent information, and cannot be solved. In either case, the matrix of coefficients $[A]$ is said to be singular, and its determinant is zero. Such a condition can be detected during partial pivoting if it is impossible to find a non-zero pivotal value at some stage of the elimination process or at the start of the back substitution process. What is more difficult to detect, however, is when a set of equations is nearly singular or ill-conditioned. This arises when, for example, two equations provide very nearly the same information about the unknowns. Alternatively, the effect of roundoff errors may be to make what is a singular set of equations apparently ill-conditioned, in that rather than a zero pivotal coefficient, a very small value is obtained. Indeed, the detection of a very small pivotal coefficient can be used as a convenient test for either a singular or very ill-conditioned set of equations, but without being able to distinguish between them. By very small pivotal coefficient is meant a value very small in relation to the magnitudes of the coefficients of the matrix at the start of the elimination process, which are of order unity if scaling has been applied.

The following subprogram ELIMIN provides a Fortran implementation of Gaussian elimination.

```
      SUBROUTINE ELIMIN(A,X,NEQN,NROW,NCOL,IFLAG)
!
!  SUBROUTINE FOR SOLVING SIMULTANEOUS LINEAR EQUATIONS BY GAUSSIAN
!  ELIMINATION WITH PARTIAL PIVOTING.
!
      REAL :: A(NROW,NCOL),X(NROW)
!
!  INITIALIZE ILL-CONDITIONING FLAG.
      IFLAG=0
!
!  SCALE EACH EQUATION TO HAVE A MAXIMUM COEFFICIENT MAGNITUDE OF UNITY.
      JMAX=NEQN+1
      Each equation in turn: DO I=1,NEQN
      AMAX=0.
      Search for maximum: DO J=1,NEQN
      ABSA=ABS(A(I,J))
      IF(ABSA > AMAX) AMAX=ABSA
      END DO Search for maximum
!
      Scale coefficients: DO J=1,JMAX
      A(I,J)=A(I,J)/AMAX
      END DO Scale coefficients
!
      END DO Each equation in turn
!
!  COMMENCE ELIMINATION PROCESS.
      Eliminate each variable in turn: DO K=1,NEQN-1
!
!  SEARCH LEADING COLUMN OF THE COEFFICIENT MATRIX FROM THE DIAGONAL
!  DOWNWARDS FOR THE LARGEST VALUE.
      IMAX=K
      Search for largest value: DO I=K+1,NEQN
      IF(ABS(A(I,K)) > ABS(A(IMAX,K))) IMAX=I
      END DO Search for largest value
!
!  IF NECESSARY, INTERCHANGE EQUATIONS TO MAKE THE LARGEST COEFFICIENT
!  BECOME THE PIVOTAL COEFFICIENT.
      IF(IMAX /= K) THEN
        Interchange coefficients: DO J=K,JMAX
```

```
         ATEMP=A(K,J)
         A(K,J)=A(IMAX,J)
         A(IMAX,J)=ATEMP
         END DO Interchange coefficients
       END IF
!
!  ELIMINATE X(K) FROM EQUATIONS (K+1) TO NEQN, FIRST TESTING FOR
!  EXCESSIVELY SMALL PIVOTAL COEFFICIENT (ASSOCIATED WITH A SINGULAR
!  OR VERY ILL-CONDITIONED MATRIX).
       IF(ABS(A(K,K)) < 1.E-5) THEN
         IFLAG=1
         RETURN
       END IF
       Each of remaining equations: DO I=K+1,NEQN
       FACT=A(I,K)/A(K,K)
       Modify coefficients: DO J=K,JMAX
       A(I,J)=A(I,J)-FACT*A(K,J)
       END DO Modify coefficients
       END DO Each of remaining equations
!
```

```
        END DO Eliminate each variable in turn
  !
  !  SOLVE THE EQUATIONS BY BACK SUBSTITUTION, FIRST TESTING
  !  FOR AN EXCESSIVELY SMALL LAST DIAGONAL COEFFICIENT.
        IF(ABS(A(NEQN,NEQN)) < 1.E-5) THEN
          IFLAG=1
          RETURN
        END IF
        X(NEQN)=A(NEQN,JMAX)/A(NEQN,NEQN)
        Then each unknown in turn backwards: DO I=NEQN-1,1,-1
        SUM=A(I,JMAX)
        Sum products: DO J=I+1,NEQN
        SUM=SUM-A(I,J)*X(J)
        END DO Sum products
        X(I)=SUM/A(I,I)
        END DO Then each unknown in turn backwards
        RETURN
        END SUBROUTINE ELIMIN
```

The arguments of ELIMIN include the array of equation coefficients, A, which incorporates the vector of constants $[b]$ as its last column, and the solution vector, X. The argument NEQN enters the number of equations to be solved, while IFLAG returns an integer flag value to the calling program to warn of a singular or very ill-conditioned coefficient matrix.

The first action of the program is to scale the coefficients of the equations to give a maximum coefficient magnitude of unity in each row of $[A]$. The elimination process is then started: each equation, with the exception of the last, is used in turn to eliminate the corresponding unknown from the subsequent equations. The current elimination number is given by K, and is equivalent to k in Equations B.4. Before performing the necessary eliminations with a particular equation, however, a search is made down the leading column of the remaining submatrix to find the coefficient of greatest magnitude, as defined by Equation B.7. The search technique locates the row number of the largest coefficient, IMAX, by first assuming that it corresponds to the coefficient on the diagonal, and only changing this if a larger coefficient is found. If the pivotal coefficient is not the largest, the relevant equations are interchanged by interchanging all their coefficients. In computing terms, this movement of data between storage locations is inefficient. While it could be avoided by keeping a record of the revised order in which the equations are to be considered, this makes the program significantly more difficult to understand. For present purposes, some computational efficiency is sacrificed in the interests of clarity.

Despite the search for the largest coefficient, it is still possible for the pivotal coefficient to be extremely small or zero if the coefficient matrix is singular or very ill-conditioned. Bearing in mind that the equations were scaled initially, an appropriate test of magnitude is $|a_{kk}| < 10^{-5}$. If this condition is satisfied at any stage of the elimination process, the problem is rejected. Rejection is indicated by setting the value of IFLAG to one, which may be detected by the calling program. If the partial pivoting is successful, however, the eliminations defined by Equations B.4 are performed, with the variable FACT being used to store the values of the factor $\emptyset$. After testing the magnitude of the last diagonal coefficient $(a_{nn}^{(n)}$ in Equations B.5), the back substitutions defined in Equations B.6 are performed to find the required solutions.

Appendix C: Matlab Version of Constant Boundary Element Program for Potential Problems

The aim of this appendix is to present a Matlab version of the program BEM2PC which is described in detail in its Fortran form in Chapter 3. Matlab is now very widely used by engineers for many purposes.

The Matlab version is as far as possible a literal translation of the Fortran. This means that the detailed explanations provided in Chapter 3 are also applicable to the Matlab script. On the other hand, no attempt is made to take advantage of special built-in features of Matlab such as simplified handling of matrices and vectors, and the solution of sets of linear equations.

Fortran is a compiler which converts the source code into machine code before computation starts. In contrast, Matlab is an interpreter which goes through the code line-by-line, translating and executing each line as it goes. Strictly speaking therefore, Matlab works with scripts rather than programs, although scripts are often loosely referred to as programs. A consequence of this difference between a compiler and an interpreter is that for large computationally intensive problems Fortran generally executes noticeably more quickly than Matlab.

The same function and variable names are used in the Fortran and Matlab programs: upper case names in Fortran become lower case in Matlab. The definitions of the names given in Some Program Variable Names at the beginnings of both Parts of the book are equally valid for both. DO loops in Fortran become "for" loops in Matlab, and functions are called in somewhat different ways. Also, STOP in Fortran is replaced by an error message to the computer screen in Matlab. The main difference, however, is to be seen in the details of the handling of input and output, although the same approach of inputting data from one pre-defined file and outputting to other files is adopted. In particular, the required input DATA files used in Fortran are unchanged, and the output files are effectively identical.

The Matlab version of BEM2PC, stored as the file BEM2PC.m, is as follows.

```
function BEM2PC(varargin)
%
%  PROGRAM FOR SOLVING TWO DIMENSIONAL POTENTIAL PROBLEMS BY THE BOUNDARY
%  ELEMENT METHOD USING CONSTANT ELEMENTS.
%
clear global; clear functions;
global fid5 fid6 fid7
```

```
%
shareddata2pc
fid5=fopen('DATA','r');
fid6=fopen('RESULTS','w+');
fid7=fopen('MESHRES','w+');
%
%  DEFINE THE MAXIMUM PROBLEM SIZE PERMITTED BY THE ARRAY DIMENSIONS.
maxnnp=1000;
maxnb=10;
%
%  INPUT THE PROBLEM TITLE AND TYPE.
intitle;
%
%  INPUT OR GENERATE THE MESH DATA.
meshc;
%
%  OUTPUT THE MESH DATA.
mshout;
%
%  INPUT, PROCESS AND OUTPUT THE BOUNDARY CONDITIONS.
bcs;
%
%  IF GOVERNING EQUATION IS POISSON TYPE, MODIFY THE BOUNDARY CONDITIONS.
poisson;
%
%  FORM THE COEFFICIENT MATRIX AND APPLY THE BOUNDARY CONDITIONS.
frmtrx;
%
%  SOLVE THE LINEAR EQUATIONS.
[psi,iflag]=elimin(a,nnp);
if(iflag == 1)
  fprintf(fid6,['\n', ...
    'MATRIX ILL-CONDITIONING DETECTED IN EQUATION SOLVER']);
  error('MATRIX ILL-CONDITIONING DETECTED IN EQUATION SOLVER')
end;
%
%  OUTPUT NODAL POINT VALUES OF POTENTIAL AND POTENTIAL GRADIENT,
%  ALSO POTENTIAL FLOWS ACROSS BOUNDARY SEGMENTS.
output;
```

```
%
%  COMPUTE VALUES OF POTENTIAL AT INTERNAL POINTS.
internal;
%
end %program BEM2PC


function intitle(varargin)
%
%  SUBPROGRAM TO INPUT PROBLEM TITLE AND TYPE (LAPLACE OR POISSON).
%
global yc xc f1 fid5 fid6
%
%  INPUT THE PROBLEM TITLE.
title=fgetl(fid5);
fprintf(fid6,['CONSTANT BOUNDARY ELEMENT SOLUTION FOR', ...
     ' TWO DIMENSIONAL POTENTIAL PROBLEM','\n','\n','%s','\n'],title);
%
%  INPUT THE VALUE OF THE (CONSTANT) F1 FUNCTION IN THE GOVERNING
%  EQUATION.
```

```
line=fgetl(fid5);
f1=str2num(line);
if(f1 == 0.)
  fprintf(fid6,['\n','LAPLACE EQUATION','\n']);
end;
if(f1 ~= 0.)
  fprintf(fid6,['\n','POISSON EQUATION,  F1 = ','%12.4e', ...
      '  CONSTANT','\n'],f1);
%
%  INPUT THE COORDINATES OF THE ORIGIN FOR THE PARTICULAR INTEGRAL.
  line=fgetl(fid5);
  vec=str2num(line);
  xc=vec(1); yc=vec(2);
  fprintf(fid6,['\n','ORIGIN FOR PARTICULAR INTEGRAL:     ', ...
      'X =','%12.4e','     Y =','%12.4e','\n'],xc,yc);
end;
%
return;
end %function intitle


function meshc(varargin)
%
%  SUBPROGRAM TO READ IN AND GENERATE THE GEOMETRIC DATA FOR A MESH OF
%  CONSTANT ELEMENTS.
%
global maxl ynode xnode nnp elength uny unx yeend xeend nep2 nep1 ...
    isegend isegelem nnpb nbound nel neend ysend xsend ifirst ...
    ilast maxnnp nsegb nsegtot maxnb fid5 fid6
%
%  INPUT THE NUMBER OF SEPARATE BOUNDARIES.
line=fgetl(fid5);
nbound=str2num(line);
%
%  TEST THE NUMBER OF BOUNDARIES.
if(nbound < 1 || nbound > maxnb)
  fprintf(fid6,['\n','NBOUND =','%4i', ...
      '  OUTSIDE PERMITTED RANGE 1 TO','%4i','\n'],nbound,maxnb);
  error('nbound error');
end;
```

```
%
%  FOR EACH BOUNDARY IN TURN INPUT THE NUMBER OF SEGMENTS.
nnp=0;
ieend=0;
nsegtot=0;
for ibound=1:nbound; % each boundary in turn
  nnpb(ibound)=0;
  line=fgetl(fid5);
  nsegb(ibound)=str2num(line);
  nsegtot=fix(nsegtot+nsegb(ibound));
%
%  TEST THE NUMBER OF SEGMENTS.
  if(nsegtot < 1 || nsegtot > maxnnp)
    fprintf(fid6,['\n','NSEGTOT =','%6i', ...
        '  OUTSIDE PERMITTED RANGE 1 TO','%6i','\n'], nsegtot,maxnnp);
    error('nsegtot error');
  end;
%
%  INPUT THE CARTESIAN GLOBAL COORDINATES OF THE END POINTS OF THE
%  SEGMENTS.  TAKE THE END POINTS CONSECUTIVELY, KEEPING THE DOMAIN
%  TO THE LEFT OF THE DIRECTION OF NUMBERING.
%  Each line of input data is assumed to contain an arbitrary number
%  of pairs of x,y coordiates.
  isend=0;
  while isend < nsegb(ibound);
    line=fgetl(fid5);
    vec=str2num(line);
    veclength=length(vec);
    np=veclength/2;
    for ip=1:np;
      isend=isend+1;
      xsend(isend)=vec(2*ip-1);
      ysend(isend)=vec(2*ip);
    end;
  end;
%
%  DEFINE THE FIRST END POINT ON THE CURRENT BOUNDARY.
  ieend=ieend+1;
  xeend(ieend)=xsend(1);
  yeend(ieend)=ysend(1);
%
```

```
%  FOR EACH OF THE SEGMENTS (BETWEEN ENDS 1 AND 2, 2 AND 3, ETC.)
%  INPUT THE RADIUS OF CURVATURE (+VE FOR CONVEX WITH CENTRE OF
%  CURVATURE INSIDE DOMAIN, -VE FOR CONCAVE), THE NUMBER OF
%  ELEMENTS IN THE SEGMENT, AND THE LENGTH RATIO BETWEEN SUCCESSIVE
%  ELEMENTS IN THE DIRECTION OF NUMBERING.
  isegmax=nsegtot;
  isegmin=isegmax-nsegb(ibound)+1;
  for iseg=isegmin:isegmax; % each segment in turn
    line=fgetl(fid5);
    vec=str2num(line);
    rseg=vec(1); nelseg=vec(2); ratseg=vec(3);
%
%  FIND AND TEST NUMBER OF NODES/ELEMENTS SO FAR.
    nnp=fix(nnp+nelseg);
    nnpb(ibound)=fix(nnpb(ibound)+nelseg);
    if(nnp < 1 || nnp > maxnnp)
      fprintf(fid6,['\n','NNP =','%6i','  OUTSIDE PERMITTED', ...
          ' RANGE 1 TO','%6i','\n'],nnp,maxnnp);
      error('nnp error');
    end;
```

```
%
%  FIRST AND LAST NODES ON CURRENT SEGMENT.
    ilast(iseg)=fix(nnp);
    ifirst(iseg)=fix(nnp-nelseg+1);
%
%  COORDINATES OF THE FIRST END POINT OF THE SEGMENT.
    isend=iseg-isegmin+1;
    xfirst=xsend(isend);
    yfirst=ysend(isend);
%
%  COORDINATES OF THE LAST END POINT OF THE SEGMENT.
    isend=isend+1;
    if(iseg == isegmax)
      isend=1;
    end;
    xlast=xsend(isend);
    ylast=ysend(isend);
%
%  GENERATE ELEMENT DATA FOR A STRAIGHT SEGMENT.
    if(rseg == 0.)
%
%  DEFINE THE ELEMENT END POINT COORDINATES ON THE SEGMENT.
      for m=1:nelseg; % each element in turn
        ieend=ieend+1;
        isegend(ieend)=fix(iseg);
        if(ratseg == 1.)
          xeend(ieend)=xfirst+(xlast-xfirst)*(m)/(nelseg);
          yeend(ieend)=yfirst+(ylast-yfirst)*(m)/(nelseg);
        end;
        if(ratseg ~= 1.)
          xeend(ieend)=xfirst+(xlast-xfirst)*(1.-ratseg^m)/ ...
              (1.-ratseg^nelseg);
          yeend(ieend)=yfirst+(ylast-yfirst)*(1.-ratseg^m)/ ...
              (1.-ratseg^nelseg);
        end;
      end; % each element in turn
    end;
%
%  GENERATE ELEMENT DATA FOR A SEGMENT IN THE FORM OF A CIRCULAR ARC.
    if(rseg ~= 0.)
```

```
%
%  LOCATE THE CENTRE OF THE ARC.
      xmid=(xfirst+xlast)/2.;
      ymid=(yfirst+ylast)/2.;
      alseg=sqrt((xlast-xfirst)^2+(ylast-yfirst)^2);
      alperp2=rseg^2-(alseg/2.)^2;
      if(abs(alperp2) < 1.0e-6*rseg^2)
        alperp2=0.;
      end;
      if(alperp2 < -1.0e-6*rseg^2)
        fprintf(fid6,['\n','DATA ERROR FOR SEGMENT NUMBER','%6i', ...
        '\n','NOT POSSIBLE TO CREATE A CIRCULAR ARC','\n'],iseg);
        error('circular arc error');
      end;
      alperp=sqrt(alperp2);
      uvflx=(xlast-xfirst)/alseg;
      uvfly=(ylast-yfirst)/alseg;
      fact=1.;
      if(rseg < 0.)
        fact=-1.;
      end;
      xcent=xmid-alperp*uvfly*fact;
      ycent=ymid+alperp*uvflx*fact;
%
%  FIND THE ANGLE SUBTENDED THERE BY THE SEGMENT.
      if(alperp ~= 0.)
        angseg=2.*atan(alseg*0.5/alperp);
      end;
      if(alperp == 0.)
        angseg=pi;
      end;
%
%  DEFINE THE ELEMENT END POINT COORDINATES ON THE SEGMENT.
      angfir=atan2(yfirst-ycent,xfirst-xcent);
      for m=1:nelseg; % each element in turn
        ieend=ieend+1;
        isegend(ieend)=fix(iseg);
        if(ratseg == 1.)
          ang=angseg*(m)/(nelseg);
        end;
        if(ratseg ~= 1.)
          ang=angseg*(1.-ratseg^m)/(1.-ratseg^nelseg);
        end;
        if(rseg < 0.)
          ang=-ang;
        end;
        xeend(ieend)=xcent+abs(rseg)*cos(angfir+ang);
        yeend(ieend)=ycent+abs(rseg)*sin(angfir+ang);
      end; % each element in turn
    end;
%
  end; % each segment in turn
end; % each boundary in turn
neend=fix(ieend);
nel=fix(nnp);
%
%  GENERATE THE ELEMENT POINT DATA AND THE NODAL POINT COORDINATES.
mmin=1;
iep1=0;
```

```
iep2=1;
for ibound=1:nbound; % each boundary in turn
  mmax=mmin+nnpb(ibound)-1;
  for m=mmin:mmax; % each element on current boundary
    iep1=iep1+1;
    iep2=iep2+1;
    nep1(m)=fix(iep1);
    nep2(m)=fix(iep2);
    xnode(m)=0.5*(xeend(iep1)+xeend(iep2));
    ynode(m)=0.5*(yeend(iep1)+yeend(iep2));
    isegelem(m)=fix(isegend(iep2));
  end; % each element on current boundary
  mmin=mmax+1;
  iep1=iep1+1;
  iep2=iep2+1;
end; % each boundary in turn
%
%  FIND AND STORE THE COMPONENTS OF THE UNIT OUTWARD NORMALS AT THE
NODES,
%  ALSO THE ELEMENT LENGTHS.
for i=1:nnp; % each node in turn
  iep1=nep1(i);
  iep2=nep2(i);
  evx=xeend(iep2)-xeend(iep1);
  evy=yeend(iep2)-yeend(iep1);
  unx(i)=evy;
  uny(i)=-evx;
  denom=sqrt(unx(i)^2+uny(i)^2);
  unx(i)=unx(i)/denom;
  uny(i)=uny(i)/denom;
  elength(i)=sqrt(evx^2+evy^2);
end; % each node in turn
%
%  DETERMINE THE MAXIMUM DIMENSION OF THE SOLUTION DOMAIN.
maxl=0.;
for i=1:nnp; % each node in turn
  for j=1:nnp; % each other node in turn
    dist=sqrt((xnode(i)-xnode(j))^2+(ynode(i)-ynode(j))^2);
    if(dist > maxl)
```

```
      maxl=dist;
    end;
  end; % each other node in turn
end; % each node in turn
%
return;
end %function meshc

function mshout(varargin)
%
%  SUBPROGRAM TO WRITE OUT THE MESH DATA.
%
global maxl ynode xnode nnp yeend xeend neend nel uny unx nep2 nep1 fid7
%
%  OUTPUT THE NUMBERS OF ELEMENTS AND NODES, ALSO THE ELEMENT END
%  POINT COORDINATE DATA.
fprintf(fid7,['\n','GEOMETRIC DATA FOR THE MESH','\n','\n', ...
    '          NUMBER OF ELEMENTS =','%6i','\n','\n', ...
    '          NUMBER OF NODAL POINTS =','%6i','\n','\n', ...
    '          NUMBER OF ELEMENT END POINTS =','%6i','\n','\n', ...
    'COORDINATES OF ELEMENT END POINTS','\n','\n', ...
    '     I     X           Y      ', ...
    '     I     X           Y      '],nel,nnp,neend);
for ieend=1:neend;
  if(mod(ieend,2) == 1);
    fprintf(fid7,['\n','%6i','%12.4e','%12.4e'], ...
        ieend,xeend(ieend),yeend(ieend));
  end;
  if(mod(ieend,2) == 0);
    fprintf(fid7,['%6i','%12.4e','%12.4e'], ...
        ieend,xeend(ieend),yeend(ieend));
  end;
end;
%
%
%  OUTPUT THE ELEMENT END POINT NUMBERS, NODAL COORDINATES AND
%  COMPONENTS OF THE UNIT OUTWARD NORMALS AT THE NODES.
fprintf(fid7,['\n','\n','ELEMENT END POINT NUMBERS,', ...
    ' NODAL COORDINATES AND UNIT NORMAL COMPONENTS','\n','\n', ...
'     M   NEP1  NEP2   X(NODE)     Y(NODE)','     UNX         UNY']);
```

```
for m=1:nel;
  fprintf(fid7,['\n','%6i','%6i','%6i','%12.4e','%12.4e','%12.4e', ...
      '%12.4e'],m,nep1(m),nep2(m),xnode(m),ynode(m),unx(m),uny(m));
end;
%
%  SCALE THE ELEMENT END POINT AND NODAL POINT COORDINATES.
for ieend=1:neend; % each element end point in turn
  xeend(ieend)=xeend(ieend)/maxl;
  yeend(ieend)=yeend(ieend)/maxl;
end; % each element end point in turn
for i=1:nnp; % each node in turn
  xnode(i)=xnode(i)/maxl;
  ynode(i)=ynode(i)/maxl;
end; % each node in turn
%
return;
end %function mshout
```

```
function bcs(varargin)
%
%  SUBPROGRAM TO INPUT, PROCESS AND OUTPUT THE BOUNDARY CONDITIONS.
%
%  INPUT THE NUMBERS OF SEGMENTS SUBJECT TO EACH TYPE OF BOUNDARY
%  CONDITION.
%     NBCP - PRESCRIBED POTENTIAL.
%     NBCD - NON-ZERO PRESCRIBED NORMAL DERIVATIVE OF POTENTIAL.
%     NBCM - MIXED BOUNDARY CONDITION.
%  ANY SEGMENT NOT INCLUDED IS ASSUMED TO BE SUBJECT TO A ZERO
%  NORMAL DERIVATIVE OF POTENTIAL.
%
global maxl beta dpsi ibc store nnp ilast ifirst betaseg alphaseg ...
 alpha nsegtot isegbc nbcm dpsiseg nbcd psiseg nbcp maxnnp nbct fid5 fid6
%
line=fgetl(fid5);
vec=str2num(line);
nbcp=vec(1); nbcd=vec(2); nbcm=vec(3);
%
%  TEST THESE BOUNDARY CONDITION NUMBERS.
nbct=fix(nbcp+nbcd+nbcm);
if(nbcp < 0 || nbcp > maxnnp || nbcd < 0 || nbcd > maxnnp|| nbcm < 0 ...
        || nbcm > maxnnp || nbct < 0 ||nbct > maxnnp)
  fprintf(fid6,['\n','NBCP =','%6i','   NBCD =','%6i','   NBCM =', ...
      '%6i','\n','NBCT =','%6i','   OUTSIDE PERMITTED RANGE 0 TO', ...
      '%6i','\n'], nbcp,nbcd,nbcm,nbct,maxnnp);
  error('numbers of boundary conditions error');
end;
%
%  INITIALISE THE BOUNDARY CONDITION STORAGE ARRAYS.
for i=1:nnp; % each node in turn
  ibc(i)=2;
  store(i)=0.;
  alpha(i)=0.;
  beta(i)=0.;
end; % each node in turn
%
%  INPUT, STORE AND OUTPUT THE PRESCRIBED POTENTIAL BOUNDARY CONDITIONS.
if(nbcp > 0)
```

```
  for ibcp=1:nbcp;
    line=fgetl(fid5);
    vec=str2num(line);
    isegbc(ibcp)=vec(1); psiseg(ibcp)=vec(2);
  end;
  fprintf(fid6,['\n','PRESCRIBED POTENTIAL BOUNDARY CONDITIONS','\n']);
  for ibcp=1:nbcp; % each segment with prescribed potential
    iseg=isegbc(ibcp);
    if(iseg < 1 || iseg > nsegtot)
      fprintf(fid6,['\n','ISEG = ','%6i','  OUTSIDE PERMITTED RANGE', ...
          ' 1 TO','%6i','\n'],iseg,nsegtot);
      error('iseg error');
    end;
    for i=ifirst(iseg):ilast(iseg); % each node on current segment
      ibc(i)=1;
      store(i)=psiseg(ibcp);
    end; % each node on current segment
    fprintf(fid6,['\n','POTENTIAL =','%12.4e','     AT NODES ','%6i', ...
        '     TO ','%6i','\n'],psiseg(ibcp),ifirst(iseg),ilast(iseg));
  end; % each segment with prescribed potential
end;
%
%  INPUT, STORE AND OUTPUT THE PRESCRIBED POTENTIAL GRADIENT BOUNDARY
%  CONDITIONS.
if(nbcd > 0)
  for ibcd=1:nbcd;
    line=fgetl(fid5);
    vec=str2num(line);
    isegbc(ibcd)=vec(1);
    dpsiseg(ibcd)=vec(2);
  end;
  fprintf(fid6,['\n', ...
      'PRESCRIBED POTENTIAL GRADIENT BOUNDARY CONDITIONS','\n']);
  for ibcd=1:nbcd; % each segment with prescribed potential gradient
    iseg=isegbc(ibcd);
    if(iseg < 1 || iseg > nsegtot)
      fprintf(fid6,['\n','ISEG = ','%6i','  OUTSIDE PERMITTED RANGE', ...
          ' 1 TO','%6i','\n'],iseg,nsegtot);
      error('iseg error');
    end;
    for i=ifirst(iseg):ilast(iseg); % each node on current segment
      ibc(i)=2;
      store(i)=dpsiseg(ibcd);
    end; % each node on current segment
    fprintf(fid6,['\n','GRADIENT =','%12.4e','     AT NODES ','%6i', ...
        '     TO ','%6i','\n'],dpsiseg(ibcd),ifirst(iseg),ilast(iseg));
  end; % each segment with prescribed potential gradient
end;
%
%  INPUT, STORE AND OUTPUT THE MIXED BOUNDARY CONDITIONS.
%  ASSUMED FORM - DPSI=ALPHA*PSI+BETA.
if(nbcm > 0)
  for ibcm=1:nbcm;
    line=fgetl(fid5);
    vec=str2num(line);
    isegbc(ibcm)=vec(1);
    alphaseg(ibcm)=vec(2);
    betaseg(ibcm)=vec(3);
  end;
  fprintf(fid6,['\n','PRESCRIBED MIXED BOUNDARY CONDITIONS','\n']);
```

ie business school
#1 EUROPEAN BUSINESS SCHOOL
FINANCIAL TIMES 2013
#gobeyond
MASTER IN MANAGEMENT
Because achieving your dreams is your greatest challenge. IE Business School's Master in Management taught in English, Spanish or bilingually, trains young high performance professionals at the beginning of their career through an innovative and stimulating program that will help them reach their full potential.
Choose your area of specialization.
Customize your master through the different options offered.
Global Immersion Weeks in locations such as Rio de Janeiro, Shanghai or San Francisco.
Because you change, we change with you.
www.ie.edu/master-management
mim.admissions@ie.edu

```
    for ibcm=1:nbcm; % each segment with prescribed mixed condition
      iseg=isegbc(ibcm);
      if(iseg < 1 || iseg > nsegtot)
        fprintf(fid6,['\n','ISEG = ','%6i','  OUTSIDE PERMITTED RANGE', ...
            ' 1 TO','%6i','\n'],iseg,nsegtot);
      error('iseg error');
      end;
      for i=ifirst(iseg):ilast(iseg); % each node on current segment
        ibc(i)=3;
        alpha(i)=alphaseg(ibcm);
        beta(i)=betaseg(ibcm);
      end; % each node on current segment
      fprintf(fid6,['\n','ALPHA =','%12.4e','     BETA =','%12.4e', ...
          '     AT NODES ','%6i','     TO ','%6i','\n'],alphaseg(ibcm), ...
          betaseg(ibcm),ifirst(iseg),ilast(iseg));
    end; % each segment with prescribed mixed condition
  end;
  %
  %  ASSEMBLE THE VECTOR OF KNOWN VARIABLES, SCALING ANY PRESCRIBED
  GRADIENTS.
  for i=1:nnp; % each node in turn
    if(ibc(i) == 1)
      dpsi(i)=store(i);
    end;
    if(ibc(i) == 2)
      dpsi(i)=store(i)*maxl;
    end;
    if(ibc(i) == 3)
      dpsi(i)=beta(i)*maxl;
    end;
  end; % each node in turn
  %
  return;
  end %function bcs


  function poisson(varargin)
  %
  %  MODIFY BOUNDARY CONDITIONS IF GOVERNING EQUATION IS POISSON TYPE.
  %
```

```
global maxl psipi alpha dpsipi dpsi ibc nnp uny unx f1 yc ynode xc xnode
%
%  INITIALISE THE PARTICULAR INTEGRAL ARRAYS.
for i=1:nnp; % each node in turn
  psipi(i)=0.;
  dpsipi(i)=0.;
end; % each node in turn
if(f1 == 0.)
  return;
end;
%
%  FIND AND STORE THE POTENTIAL AND POTENTIAL GRADIENT AT EVERY NODE
%  DUE TO THE PARTICULAR INTEGRAL.
xc=xc/maxl;
yc=yc/maxl;
f1=f1*maxl^2;
for i=1:nnp; % each node in turn
  xx=xnode(i)-xc;
  yy=ynode(i)-yc;
  psipi(i)=0.25*f1*(xx^2+yy^2);
  dpsipi(i)=0.5*f1*(xx*unx(i)+yy*uny(i));
end; % each node in turn
%
%  SUBTRACT THE PARTICULAR INTEGRAL CONTRIBUTIONS FROM THE PRESCRIBED
%  BOUNDARY CONDITIONS.
for i=1:nnp; % each node in turn
  if(ibc(i) == 1)
    dpsi(i)=dpsi(i)-psipi(i);
  end;
  if(ibc(i) == 2)
    dpsi(i)=dpsi(i)-dpsipi(i);
  end;
  if(ibc(i) == 3)
    dpsi(i)=dpsi(i)-dpsipi(i)+alpha(i)*psipi(i)*maxl;
  end;
end; % each node in turn
%
return;
end %function poisson
```

```
function frmtrx(varargin)
%
%  SUBPROGRAM TO FORM THE COEFFICIENT MATRIX AND RIGHT HAND SIDE VECTOR,
%  MODIFIED TO SUIT THE BOUNDARY CONDITIONS.
%
global a dpsi maxl alpha ibc nnp
%
%  DEFINE THE NUMBER OF COLUMNS IN THE EXTENDED COEFFICIENT MATRIX A.
jmax=nnp+1;
%
%  FORM THE COEFFICIENT MATRIX A, AND THE RIGHT HAND SIDE VECTOR B*DPSI.
for i=1:nnp; % take each node in turn as P
  bdpsi=0.;
  for j=1:nnp; % take each element in turn to contain Q
%
%  EVALUATE THE KERNELS WHEN P IS NOT IN THE ELEMENT CONTAINING Q.
    if(i ~= j)
      [aij,bij]=kernel(i,j);
    end;
%
```

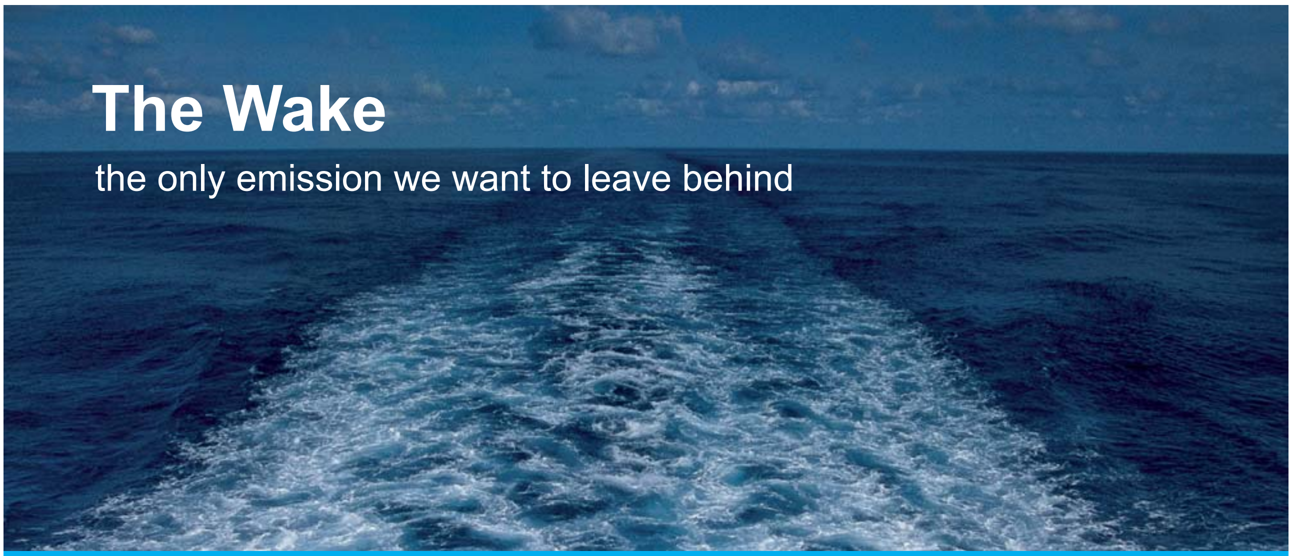
```
%  EVALUATE THE KERNELS WHEN P IS IN THE ELEMENT CONTAINING Q.
    if(i == j)
      aij=pi;
      bij=kern2(j);
    end;
%
%  IF POTENTIAL IS PRESCRIBED OVER THE ELEMENT CONTAINING Q,
%  INTERCHANGE THE A AND B COEFFICIENTS.
    if(ibc(j) == 1)
      temp=aij;
      aij=-bij;
      bij=-temp;
    end;
%
%  IF THE BOUNDARY CONDITION OVER THE ELEMENT CONTAINING Q IS MIXED,
%  MODIFY THE A COEFFICIENT.
    if(ibc(j) == 3)
      aij=aij-bij*alpha(j)*maxl;
    end;
%
%  STORE THE A COEFFICIENT AND ACCUMULATE THE B*DPSI VECTOR COEFFICIENT.
    a(i,j)=aij;
    bdpsi=bdpsi+bij*dpsi(j);
  end; % take each element in turn to contain Q
%
%  STORE THE B*DPSI COEFFICIENT AS AN EXTENSION OF MATRIX A.
  a(i,jmax)=bdpsi;
end; % take each node in turn as P
%
return;
end %function frmtrx

function [aij,bij]=kernel(i,j)
%
%  SUBPROGRAM TO EVALUATE THE INTEGRALS OF THE KERNEL FUNCTIONS
%  WHEN P IS NOT IN THE ELEMENT CONTAINING Q.
%
global uny unx ynode yeend xnode xeend nep2 nep1 fid6
%
```

```
%  FIND THE COMPONENTS OF THE RADIAL DISTANCES OF P FROM THE ENDS
%  OF THE ELEMENT CONTAINING Q, AND THE DISTANCES THEMSELVES.
iep1=nep1(j);
iep2=nep2(j);
r1x=xeend(iep1)-xnode(i);
r2x=xeend(iep2)-xnode(i);
r1y=yeend(iep1)-ynode(i);
r2y=yeend(iep2)-ynode(i);
r1=sqrt(r1x^2+r1y^2);
r2=sqrt(r2x^2+r2y^2);
if(r1 < 1.0e-6 || r2 < 1.0e-6)
  fprintf(fid6,['\n','ERROR - INTERNAL POINT COINCIDENT WITH AN END', ...
      ' POINT OF ELEMENT','%6i','\n'],j);
  error('coincident points');
end;
%
%  FIND THE COMPONENTS OF THE UNIT VECTORS FROM P TO THE ENDS OF
%  THE ELEMENT CONTAINING Q.
ur1x=r1x/r1;
ur2x=r2x/r2;
ur1y=r1y/r1;
ur2y=r2y/r2;
%
%  FIND THE PERPENDICULAR DISTANCE OF P FROM THE ELEMENT CONTAINING Q.
d=r1x*unx(j)+r1y*uny(j);
%
%  EVALUATE THE INTEGRALS OF THE KERNELS.
aij=asin(ur1y*ur2x-ur1x*ur2y);
costh=ur1x*ur2x+ur1y*ur2y;
if(costh < 0. && aij > 0.)
  aij=pi-aij;
end;
if(costh < 0. && aij <= 0.)
  aij=-pi-aij;
end;
sinth1=unx(j)*ur1y-uny(j)*ur1x;
sinth2=unx(j)*ur2y-uny(j)*ur2x;
blim1=r1*sinth1*(log(1./r1)+1.);
blim2=r2*sinth2*(log(1./r2)+1.);
bij=blim2-blim1+d*aij;
```

```
%
return;
end %function kernel

function bij=kern2(j)
%
%  SUBPROGRAM TO EVALUATE THE INTEGRAL OF THE SECOND KERNEL
%  FUNCTION WHEN P IS IN THE ELEMENT CONTAINING Q.
%
global maxl elength
%
%  EVALUATE THE SEMI LENGTH OF THE ELEMENT CONTAINING Q.
r1=0.5*elength(j)/maxl;
%
%  HENCE THE INTEGRAL OF THE KERNEL.
bij=2.*r1*(log(1./r1)+1.);
%
return;
end %function kern2
```



```
 function output(varargin)
%
%  SUBPROGRAM TO WRITE OUT THE NODAL POINT VALUES OF POTENTIAL AND
%  POTENTIAL GRADIENT, AND COMPUTE POTENTIAL FLOWS ACROSS THE
%  BOUNDARY SEGMENTS.
%
global nsegtot flowseg elength dpsit isegelem nel psit maxl dpsipi ...
    psipi f1 dpsi psi nnp beta alpha ibc fid6
%
%  ARRANGE FOR PSI AND DPSI TO CONTAIN THE POTENTIALS AND POTENTIAL
%  GRADIENTS, RESPECTIVELY.
for i=1:nnp; % each node in turn
  if(ibc(i) == 1)
    temp=psi(i);
    psi(i)=dpsi(i);
    dpsi(i)=temp;
  end;
  if(ibc(i) == 3)
    psitot=psi(i)+psipi(i);
    dpsi(i)=(alpha(i)*psitot+beta(i))*maxl-dpsipi(i);
  end;
end; % each node in turn
%
%  ADD CONTRIBUTIONS OF THE PARTICULAR INTEGRAL.
fprintf(fid6,['\n','NODAL POINT POTENTIALS AND POTENTIAL GRADIENTS', ...
    '\n','\n','      I      PSI            DPSI      ','\n']);
for i=1:nnp; % each node in turn
  psit(i)=psi(i);
  dpsit(i)=dpsi(i);
  if(f1 ~= 0.)
    psit(i)=psit(i)+psipi(i);
    dpsit(i)=dpsit(i)+dpsipi(i);
  end;
%
%  REMOVE THE SCALE FACTOR FROM THE GRADIENT VALUES.
  dpsit(i)=dpsit(i)/maxl;
%
%  OUTPUT THE NODAL VALUES OF POTENTIAL AND POTENTIAL GRADIENT.
  fprintf(fid6,[' ','%6i','%15.6e','%15.6e','\n'],i,psit(i),dpsit(i));
end; % each node in turn
```

```
%
%  COMPUTE THE POTENTIAL FLOWS ACROSS THE BOUNDARY SEGMENTS.
flowin=0.;
flowout=0.;
for iseg=1:nsegtot; % each segment in turn
  flowseg(iseg)=0.;
end; % each segment in turn
for m=1:nel; % each element in turn
  iseg=isegelem(m);
  flowelem=dpsit(m)*elength(m);
  flowseg(iseg)=flowseg(iseg)+flowelem;
  if(flowelem > 0.)
    flowin=flowin+flowelem;
  end;
  if(flowelem < 0.)
    flowout=flowout-flowelem;
  end;
end; % each element in turn
fprintf(fid6,['\n', ...
  'POTENTIAL FLOWS INTO DOMAIN ACROSS BOUNDARY SEGMENTS','\n','\n', ...
  'SEGMENT      FLOW','\n']);
for iseg=1:nsegtot;
  fprintf(fid6,['%5i','%16.6e','\n'],iseg,flowseg(iseg));
end;
fprintf(fid6,['\n','TOTAL POTENTIAL FLOW INTO DOMAIN =','%15.6e', ...
    '\n','\n','TOTAL POTENTIAL FLOW OUT OF DOMAIN =','%15.6e', ...
    '\n'],flowin,flowout);
%
return;
end %function output


function internal(varargin)
%
%  SUBPROGRAM TO COMPUTE AND OUTPUT THE POTENTIALS AT SELECTED
%  INTERNAL POINTS.
%
global f1 yc ynode xc xnode dpsi psi nnp maxl fid5 fid6
%
%  INPUT NUMBER OF INTERNAL POINTS.
```

```
line=fgetl(fid5);
nint=str2num(line);
if(nint == 0)
  return;
end;
fprintf(fid6,['\n','POTENTIALS AT INTERNAL POINTS','\n','\n', ...
    '     PSI                X            Y']);
%
%  INPUT THE INTERNAL POINT COORDINATES.
for iint=1:nint; % each internal point in turn
  line=fgetl(fid5);
  vec=str2num(line);
  xint=vec(1); yint=vec(2);
  xnode(1)=xint/maxl;
  ynode(1)=yint/maxl;
%
%  INTEGRATE ROUND THE BOUNDARY.
  sum=0.;
  for j=1:nnp; % each node in turn
    i=1;
```

```
      [aij,bij]=kernel(i,j);
      sum=sum-aij*psi(j)+bij*dpsi(j);
    end; % each node in turn
    psiip=sum*0.5/pi;
    psiipt=psiip;
  %
  %  ADD THE PARTICULAR INTEGRAL FOR A POISSON TYPE PROBLEM.
    if(f1 ~= 0.)
      xx=xnode(1)-xc;
      yy=ynode(1)-yc;
      psiipt=psiipt+0.25*f1*(xx^2+yy^2);
    end;
  %
  %  OUTPUT THE POTENTIAL AT THE INTERNAL POINT.
    fprintf(fid6,['\n','%14.6e','     ','%12.4e','%12.4e'], ...
        psiipt,xint,yint);
  end; % each internal point in turn
  %
  return;
  end %function internal

  function [x,iflag]=elimin(a,neqn)
  %
  %  SUBROUTINE FOR SOLVING SIMULTANEOUS LINEAR EQUATIONS BY GAUSSIAN
  %  ELIMINATION WITH PARTIAL PIVOTING.
  %
  %  INITIALIZE ILL-CONDITIONING FLAG.
  iflag=0;
  for i=1:neqn; x(i)=0.; end;
  %
  %  SCALE EACH EQUATION TO HAVE A MAXIMUM COEFFICIENT MAGNITUDE OF UNITY.
  jmax=neqn+1;
  for i=1:neqn; % each equation in turn
    amax=0.;
    for j=1:neqn; % search for maximum
      absa=abs(a(i,j));
      if(absa > amax)
        amax=absa;
      end;
    end; % search for maximum
  %
    for j=1:jmax; % scale coefficients
      a(i,j)=a(i,j)/amax;
    end; % scale coefficients
  %
  end; % each equation in turn
  %
  %  COMMENCE ELIMINATION PROCESS.
  for k=1:neqn-1; % eliminate each variable in turn
  %
  %  SEARCH LEADING COLUMN OF THE COEFFICIENT MATRIX FROM THE DIAGONAL
  %  DOWNWARDS FOR THE LARGEST VALUE.
    imax=k;
    for i=k+1:neqn; % search for largest value
      if(abs(a(i,k)) > abs(a(imax,k)))
        imax=i;
      end;
    end; % search for largest value
  %
  %  IF NECESSARY, INTERCHANGE EQUATIONS TO MAKE THE LARGEST COEFFICIENT
  %  BECOME THE PIVOTAL COEFFICIENT.
    if(imax ~= k)
      for j=k:jmax; % interchange coefficients
        atemp=a(k,j);
        a(k,j)=a(imax,j);
        a(imax,j)=atemp;
      end; % interchange coefficients
    end;
  %
  %  ELIMINATE X(K) FROM EQUATIONS (K+1) TO NEQN, FIRST TESTING FOR
  %  EXCESSIVELY SMALL PIVOTAL COEFFICIENT (ASSOCIATED WITH A SINGULAR
  %  OR VERY ILL-CONDITIONED MATRIX).
    if(abs(a(k,k)) < 1.0e-5)
      iflag=1;
      return;
    end;
    for i=k+1:neqn; % each of remaining equations
      fact=a(i,k)/a(k,k);
      for j=k:jmax; % modify coefficients
        a(i,j)=a(i,j)-fact*a(k,j);
      end; % modify coefficient
    end; % each of remaining equations
  %
  end; % eliminate each variable in turn
  %
  %  SOLVE THE EQUATIONS BY BACK SUBSTITUTION, FIRST TESTING
  %  FOR AN EXCESSIVELY SMALL LAST DIAGONAL COEFFICIENT.
  if(abs(a(neqn,neqn)) < 1.0e-5)
    iflag=1;
    return;
  end;
  x(neqn)=a(neqn,jmax)/a(neqn,neqn);
  for i=neqn-1:-1:1; % then each unknown in turn backwards
    sum=a(i,jmax);
    for j=i+1:neqn; % sum products
      sum=sum-a(i,j)*x(j);
    end; % sum products
    x(i)=sum/a(i,i);
  end; % then each unknown in turn backwards
  return;
  end %function elimin
```

The equivalent of the Fortran SHAREDDDATA2PC is shareddata2pc, stored as the file shareddata2pc.m, is as follows.

```
%%%--- module shareddata2pc;
%
% module STORING SHARED DATA.
%
 global xeend; if isempty(xeend), xeend=zeros(1,1010); end;
 global yeend; if isempty(yeend), yeend=zeros(1,1010); end;
 global xnode; if isempty(xnode), xnode=zeros(1,1000); end;
 global ynode; if isempty(ynode), ynode=zeros(1,1000); end;
 global xsend; if isempty(xsend), xsend=zeros(1,1000); end;
 global ysend; if isempty(ysend), ysend=zeros(1,1000); end;
 global unx; if isempty(unx), unx=zeros(1,1000); end;
 global uny; if isempty(uny), uny=zeros(1,1000); end;
 global f1; if isempty(f1), f1=0; end;
 global xc; if isempty(xc), xc=0; end;
 global yc; if isempty(yc), yc=0; end;
 global psi; if isempty(psi), psi=zeros(1,1000); end;
 global dpsi; if isempty(dpsi), dpsi=zeros(1,1000); end;
 global alpha; if isempty(alpha), alpha=zeros(1,1000); end;
 global beta; if isempty(beta), beta=zeros(1,1000); end;
 global psiseg; if isempty(psiseg), psiseg=zeros(1,1000); end;
 global dpsiseg; if isempty(dpsiseg), dpsiseg=zeros(1,1000); end;
 global alphaseg; if isempty(alphaseg), alphaseg=zeros(1,1000); end;
 global betaseg; if isempty(betaseg), betaseg=zeros(1,1000); end;
 global store; if isempty(store), store=zeros(1,1000); end;
 global psipi; if isempty(psipi), psipi=zeros(1,1000); end;
 global dpsipi; if isempty(dpsipi), dpsipi=zeros(1,1000); end;
 global psit; if isempty(psit), psit=zeros(1,1000); end;
 global dpsit; if isempty(dpsit), dpsit=zeros(1,1000); end;
 global flowseg; if isempty(flowseg), flowseg=zeros(1,1000); end;
 global a; if isempty(a), a=zeros(1000,1001); end;
 global maxl; if isempty(maxl), maxl=0; end;
 global elength; if isempty(elength), elength=zeros(1,1000); end;
 global nel; if isempty(nel), nel=0; end;
 global nnp; if isempty(nnp), nnp=0; end;
 global maxnnp; if isempty(maxnnp), maxnnp=0; end;
 global maxnb; if isempty(maxnb), maxnb=0; end;
 global nep1; if isempty(nep1), nep1=zeros(1,1000); end;
 global nep2; if isempty(nep2), nep2=zeros(1,1000); end;
 global neend; if isempty(neend), neend=0; end;
 global nbound; if isempty(nbound), nbound=0; end;
 global nsegtot; if isempty(nsegtot), nsegtot=0; end;
 global nnpb; if isempty(nnpb), nnpb=zeros(1,10); end;
 global nsegb; if isempty(nsegb), nsegb=zeros(1,10); end;
 global nbcp; if isempty(nbcp), nbcp=0; end;
 global nbcd; if isempty(nbcd), nbcd=0; end;
 global nbcm; if isempty(nbcm), nbcm=0; end;
 global nbct; if isempty(nbct), nbct=0; end;
 global ibc; if isempty(ibc), ibc=zeros(1,1000); end;
 global isegbc; if isempty(isegbc), isegbc=zeros(1,1000); end;
 global ifirst; if isempty(ifirst), ifirst=zeros(1,1000); end;
 global ilast; if isempty(ilast), ilast=zeros(1,1000); end;
 global isegend; if isempty(isegend), isegend=zeros(1,1000); end;
 global isegelem; if isempty(isegelem), isegelem=zeros(1,1000); end;
```

Finally, the alternative form of function "internal", which is required to carry out area integration over a rectangular solution domain (described in Section 3.3), is as follows.

```
function internal(varargin)
%
%  SUBPROGRAM TO FIND AND OUTPUT THE INTEGRAL OF POTENTIAL OVER THE
%  AREA OF THE SOLUTION DOMAIN.
%  THIS VERSION IS DESIGNED FOR A RECTANGULAR DOMAIN.
%
global nbcp isegbc psiseg xsend ysend xnode ynode maxl nnp psi ...
    dpsi xc yc f1 fid5 fid6
global psival; if isempty(psival), psival=zeros(1,300); end;
global yintgl; if isempty(yintgl), yintgl=zeros(1,300); end;
%
%  INPUT THE NUMBERS OF SAMPLING POINTS INSIDE THE DOMAIN IN THE X
%  AND Y DIRECTIONS, WHICH SHOULD BOTH BE ODD.
line=fgetl(fid5);
vec=str2num(line);
nx=vec(1); ny=vec(2);
if(rem(nx,2) == 0)
  fprintf(fid6,['\n','WARNING - NX =','%4i', ...
      ' NOT ODD - INCREASE BY 1','\n'],nx);
  nx=nx+1;
end;
if(rem(ny,2) == 0)
  fprintf(fid6,['\n','WARNING - NY =','%4i', ...
      ' NOT ODD - INCREASE BY 1','\n'],ny);
  ny=ny+1;
end;
if(nx+2 > 300 || ny+2 > 300)
  fprintf(fid6,['\n','NX =','%6i','     ','NY =','%6i','     ', ...
      'ARRAY DIMENSIONS EXCEEDED',' IN INTEGRATION OVER DOMAIN', ...
      '\n'],nx,ny);
  error('array dimension error in internal');
end;
%
%  FIND PRESCRIBED BOUNDARY VALUES OF POTENTIAL.
for ibcp=1:nbcp % each segment in turn
  if(isegbc(ibcp) ==  1)
    psibot=psiseg(ibcp);
  end;
  if(isegbc(ibcp) ==  2)
```

```
      psiright=psiseg(ibcp);
    end;
    if(isegbc(ibcp) ==  3)
      psitop=psiseg(ibcp);
    end;
    if(isegbc(ibcp) ==  4)
      psileft=psiseg(ibcp);
    end;
  end % each segment in turn;
  %
  %  FIND COORDINATES OF DOMAIN SIDES.
  ybot=ysend(1);
  xright=xsend(2);
  ytop=ysend(3);
  xleft=xsend(4);
  %
  %  OUTER LOOP FOR INTEGRATION IN X DIRECTION.
  hx=(xright-xleft)/(nx+1);
  hy=(ytop-ybot)/(ny+1);
  ixmax=nx+2;
  iymax=ny+2;
  yintgl(1)=psileft*(ytop-ybot);
  yintgl(ixmax)=psiright*(ytop-ybot);
  for ix=2:ixmax-1; % each x position in turn
    psival(1)=psibot;
    psival(iymax)=psitop;
    xpoint=xleft+hx*(ix-1);
  %
  %  INNER LOOP FOR INTEGRATION IN Y DIRECTION.
  %
  %  FIRST FIND POTENTIALS AT THE INTERNAL POINTS.
    for iy=2:iymax-1; % each y position in turn
      ypoint=ybot+hy*(iy-1);
      xnode(1)=xpoint/maxl;
      ynode(1)=ypoint/maxl;
  %
  %  INTEGRATE ROUND THE BOUNDARY.
      sum=0.;
      for j=1:nnp; % each node in turn
        i=1;
        [aij,bij]=kernel(i,j);
        sum=sum-aij*psi(j)+bij*dpsi(j);
      end; % each node in turn;
      psiip=sum*0.5/pi;
      psiipt=psiip;
  %
  %  ADD PARTICULAR INTEGRAL.
      if(f1 ~= 0.)
        xx=xnode(1)-xc;
        yy=ynode(1)-yc;
        psiipt=psiipt+0.25*f1*(xx^2+yy^2);
      end;
      psival(iy)=psiipt;
    end; % each y position in turn
  %
  %  INTEGRATE POTENTIAL IN Y DIRECTION.
    sum=0.;
    for iy=2:2:iymax-1; % alternate points in the y direction
      sum=sum+psival(iy-1)+4.*psival(iy)+psival(iy+1);
    end; % alternate points in the y direction
    yintgl(ix)=sum*hy/3.;
  end; % each x position in turn
  %
  %  NOW INTEGRATE IN X DIRECTION.
  sum=0.;
  for ix=2:2:ixmax-1; % alternate points in the x direction
    sum=sum+yintgl(ix-1)+4.*yintgl(ix)+yintgl(ix+1);
  end; % alternate points in the x direction
  psiintg=sum*hx/3.;
  area=(xright-xleft)*(ytop-ybot);
  apsiintg=psiintg/area;
  %
  %  OUTPUT RESULT.
  fprintf(fid6,['\n','INTEGRAL OF POTENTIAL OVER SOLUTION DOMAIN =', ...
      '%15.6e','\n','AREA AVERAGE OVER SOLUTION DOMAIN =','%15.6e', ...
      '\n','NUMBERS OF INTERNAL POINTS IN X AND Y DIRECTIONS =', ...
      '%5i','     ','AND','%5i','\n'],psiintg,apsiintg,nx,ny);
  %
  return;
  end %function internal
```

Appendix D: Matlab Version of Quadratic Boundary Element Program for Potential Problems

The aim of this appendix is to present a Matlab version of the program BEM2PQ which is described in detail in its Fortran form in Chapter 4.

The Matlab version is as far as possible a literal translation of the Fortran. This means that the detailed explanations provided in Chapter 4 are also applicable to the Matlab script. The same function and variable names are used in the Fortran and Matlab programs: upper case names in Fortran become lower case in Matlab. The definitions of the names given in Some Program Variable Names at the beginnings of both Parts of the book are equally valid for both. The introduction to Appendix C provides further comments on the differences between Fortran and Matlab, particularly as they affect the present programs.

The Matlab version of BEM2PQ, stored as the file BEM2PQ.m, is as follows.

```
function BEM2PQ(varargin)
%
%  PROGRAM FOR SOLVING TWO DIMENSIONAL POTENTIAL PROBLEMS BY THE BOUNDARY
%  ELEMENT METHOD USING QUADRATIC ELEMENTS.
%
clear global; clear functions;
global fid5 fid6 fid7
%
shareddata2pq
fid5=fopen('DATA','r');
fid6=fopen('RESULTS','w+');
fid7=fopen('MESHRES','w+');
%
%  DEFINE THE MAXIMUM PROBLEM SIZE PERMITTED BY THE ARRAY DIMENSIONS.
maxnel=500;
maxnnp=1000;
maxnb=10;
%
%  INPUT THE PROBLEM TITLE AND TYPE.
intitle;
%
%  INPUT AND GENERATE THE MESH DATA.
meshq;
%
%  OUTPUT THE MESH DATA.
mshout;
%
%  INPUT, PROCESS AND OUTPUT THE BOUNDARY CONDITIONS.
bcs;
%
%  EVALUATE AND STORE VALUES OF THE SHAPE FUNCTIONS AND THEIR DERIVATIVES
%  AT THE GAUSS POINTS AND NODES.
shape;
%
%  IF GOVERNING EQUATION IS POISSON TYPE, MODIFY THE BOUNDARY CONDITIONS.
poisson;
%
%  FORM THE COEFFICIENT MATRIX AND APPLY THE BOUNDARY CONDITIONS.
frmtrx;
%
```

```
%  SOLVE THE LINEAR EQUATIONS.
[psi,iflag]=elimin(a,nnp);
if(iflag == 1)
  fprintf(fid6,['\n' , ...
    'MATRIX ILL-CONDITIONING DETECTED IN EQUATION SOLVER','\n']);
  error('MATRIX ILL-CONDITIONING DETECTED IN EQUATION SOLVER');
end;
%
%  OUTPUT NODAL POINT VALUES OF POTENTIAL AND POTENTIAL GRADIENT,
%  ALSO POTENTIAL FLOWS ACROSS BOUNDARY SEGMENTS.
output;
%
%  COMPUTE VALUES OF POTENTIAL AT INTERNAL POINTS.
internal;
%
end %program BEM2PQ


function intitle(varargin)
%
%  SUBPROGRAM TO INPUT PROBLEM TITLE AND TYPE (LAPLACE OR POISSON).
%
global yc xc f1 fid5 fid6
%
%  INPUT THE PROBLEM TITLE.
title=fgetl(fid5);
fprintf(fid6,['QUADRATIC BOUNDARY ELEMENT SOLUTION FOR', ...
    ' TWO DIMENSIONAL POTENTIAL PROBLEM','\n','\n','%s','\n'],title);
%
%  INPUT THE VALUE OF THE (CONSTANT) F1 FUNCTION IN THE GOVERNING
%  EQUATION.
line=fgetl(fid5);
f1=str2num(line);
if(f1 == 0.)
  fprintf(fid6,['\n','LAPLACE EQUATION','\n']);
end;
if(f1 ~= 0.)
  writef(fid6,['\n','POISSON EQUATION,  F1 = ','%12.4e', ...
      '  CONSTANT','\n'],f1);
%
```

```
%  INPUT THE COORDINATES OF THE ORIGIN FOR THE PARTICULAR INTEGRAL.
  line=fgetl(fid5);
  vec=str2num(line);
  xc=vec(1); yc=vec(2);
  fprintf(fid6,['\n','ORIGIN FOR PARTICULAR INTEGRAL:     ', ...
      'X =','%12.4e','     Y =','%12.4e','\n'],xc,yc);
end;
%
return;
end %function intitle


function meshq(varargin)
%
%  SUBPROGRAM TO READ IN AND GENERATE THE GEOMETRIC DATA FOR A MESH OF
%  QUADRATIC ELEMENTS.
%
global node maxl ynode xnode nnp nel neend m3 m1 angstore yeend xeend ...
    isegelem mfirst isegend ysend xsend mlast maxnel nelb nsegb ...
    nsegtot nbound maxnb fid5 fid6
```

```
%
%  INPUT THE NUMBER OF SEPARATE BOUNDARIES.
line=fgetl(fid5);
nbound=str2num(line);
%
%  TEST THE NUMBER OF BOUNDARIES.
if(nbound < 1 || nbound > maxnb)
  fprintf(fid6,['\n','NBOUND =','%4i', ...
      '  OUTSIDE PERMITTED RANGE 1 TO','%4i','\n'],nbound,maxnb);
  error('nbound error');
end;
%
%  FOR EACH BOUNDARY IN TURN INPUT THE NUMBER OF SEGMENTS.
nel=0;
ieend=0;
nsegtot=0;
mmaxb=0;
for ibound=1:nbound; % each boundary in turn
  nelb(ibound)=0;
  mminb=mmaxb+1;
  line=fgetl(fid5);
  nsegb(ibound)=str2num(line);
  nsegtot=fix(nsegtot+nsegb(ibound));
%
%  TEST THE NUMBER OF SEGMENTS.
  if(nsegtot < 1 || nsegtot > maxnel)
    fprintf(fid6,['\n','NSEGTOT =','%6i', ...
        '  OUTSIDE PERMITTED RANGE 1 TO','%6i','\n'],nsegtot,maxnel);
    error('nsegtot error');
  end;
%
%  INPUT THE CARTESIAN GLOBAL COORDINATES OF THE END POINTS OF THE
%  SEGMENTS.  TAKE THE END POINTS CONSECUTIVELY, KEEPING THE DOMAIN
%  TO THE LEFT OF THE DIRECTION OF NUMBERING.
%  Each line of input data is assumed to contain an arbitrary number
%  of pairs of x,y coordinates.
  isend=0;
  while isend < nsegb(ibound);
    line=fgetl(fid5);
    vec=str2num(line);
    veclength=length(vec);
    np=veclength/2;
    for ip=1:np;
      isend=isend+1;
      xsend(isend)=vec(2*ip-1);
      ysend(isend)=vec(2*ip);
    end;
  end;
%
%  DEFINE THE FIRST END POINT ON THE CURRENT BOUNDARY.
  ieend=ieend+1;
  xeend(ieend)=xsend(1);
  yeend(ieend)=ysend(1);
%
%  FOR EACH OF THE SEGMENTS (BETWEEN ENDS 1 AND 2, 2 AND 3, ETC.)
%  INPUT THE RADIUS OF CURVATURE (+VE FOR CONVEX WITH CENTRE OF
%  CURVATURE INSIDE DOMAIN, -VE FOR CONCAVE), THE NUMBER OF
%  ELEMENTS IN THE SEGMENT, AND THE LENGTH RATIO BETWEEN SUCCESSIVE
%  ELEMENTS IN THE DIRECTION OF NUMBERING.
  isegmax=nsegtot;
  isegmin=isegmax-nsegb(ibound)+1;
  for iseg=isegmin:isegmax; % each segment in turn
    line=fgetl(fid5);
    vec=str2num(line);
    rseg=vec(1); nelseg=vec(2); ratseg=vec(3);
%
%  FIND AND TEST THE NUMBER OF ELEMENTS SO FAR.
    nel=fix(nel+nelseg);
    nelb(ibound)=fix(nelb(ibound)+nelseg);
    if(nel < 1 || nel > maxnel)
      fprintf(fid6,['\n','NEL =','%6i','  OUTSIDE PERMITTED' ...
          ' RANGE 1 TO','%6i','\n'],nel,maxnel);
      error('nel error');
    end;
%
%  FIRST AND LAST ELEMENTS ON CURRENT SEGMENT.
    mlast(iseg)=fix(nel);
    mfirst(iseg)=fix(nel-nelseg+1);
    mmaxb=mmaxb+nelseg;
```

```
%
%  COORDINATES OF THE FIRST END POINT OF THE SEGMENT.
    isend=iseg-isegmin+1;
    xfirst=xsend(isend);
    yfirst=ysend(isend);
%
%  COORDINATES OF THE LAST END POINT OF THE SEGMENT.
    isend=isend+1;
    if(iseg == isegmax)
      isend=1;
    end;
    xlast=xsend(isend);
    ylast=ysend(isend);
%
%  GENERATE ELEMENT DATA FOR A STRAIGHT SEGMENT.
    if(rseg == 0.)
%
%  DEFINE THE ELEMENT END POINT COORDINATES ON THE SEGMENT.
      for m=1:nelseg; % each element in turn
        ieend=ieend+1;
```

```
          isegend(ieend)=fix(iseg);
          if(ratseg == 1.)
            xeend(ieend)=xfirst+(xlast-xfirst)*(m)/(nelseg);
            yeend(ieend)=yfirst+(ylast-yfirst)*(m)/(nelseg);
          end;
          if(ratseg ~= 1.)
            xeend(ieend)=xfirst+(xlast-xfirst)*(1.-ratseg^m)/ ...
                (1.-ratseg^nelseg);
            yeend(ieend)=yfirst+(ylast-yfirst)*(1.-ratseg^m)/ ...
                (1.-ratseg^nelseg);
          end;
        end; % each element in turn
%
%  DEFINE THE ELEMENT NODES AND COORDINATES.
        ieend=ieend-nelseg;
        for ielseg=1:nelseg; % each element in turn
          ieend=ieend+1;
          m=mfirst(iseg)+ielseg-1;
          i1=2*m-1;
          i2=i1+1;
          i3=i1+2;
          if(iseg == isegmax && ielseg == nelseg)
            i3=node(mminb,1);
          end;
          node(m,1)=fix(i1);
          node(m,2)=fix(i2);
          node(m,3)=fix(i3);
          isegelem(m)=fix(iseg);
          if(iseg == isegmin && ielseg == 1)
            xnode(i1)=xeend(ieend-1);
            ynode(i1)=yeend(ieend-1);
          end;
          xnode(i3)=xeend(ieend);
          ynode(i3)=yeend(ieend);
          xnode(i2)=0.5*(xnode(i1)+xnode(i3));
          ynode(i2)=0.5*(ynode(i1)+ynode(i3));
%
%  STORE THE NUMBERS OF THE ADJACENT ELEMENTS.
          m1(m)=fix(m-1);
          m3(m)=fix(m+1);
```

```
      end; % each element in turn
    end;
%
%  GENERATE ELEMENT DATA FOR A SEGMENT IN THE FORM OF A CIRCULAR ARC.
    if(rseg ~= 0.)
%
%  LOCATE THE CENTRE OF THE ARC.
      xmid=(xfirst+xlast)/2.;
      ymid=(yfirst+ylast)/2.;
      alseg=sqrt((xlast-xfirst)^2+(ylast-yfirst)^2);
      alperp2=rseg^2-(alseg/2.)^2;
      if(abs(alperp2) < 1.0e-6*rseg^2)
        alperp2=0.;
      end;
      if(alperp2 < -1.0e-6*rseg^2)
        fprintf(fid6,['\n','DATA ERROR FOR SEGMENT NUMBER','%6i', ...
        '\n','NOT POSSIBLE TO CREATE A CIRCULAR ARC','\n'],iseg);
        error('circular arc error');
      end;
      alperp=sqrt(alperp2);
      uvflx=(xlast-xfirst)/alseg;
      uvfly=(ylast-yfirst)/alseg;
      fact=1.;
      if(rseg < 0.)
        fact=-1.;
      end;
      xcent=xmid-alperp*uvfly*fact;
      ycent=ymid+alperp*uvflx*fact;
%
%  FIND THE ANGLE SUBTENDED THERE BY THE SEGMENT.
      if(alperp ~= 0.)
        angseg=2.*atan(alseg*0.5/alperp);
      end;
      if(alperp == 0.)
        angseg=pi;
      end;
%
%  DEFINE THE ELEMENT END POINT COORDINATES ON THE SEGMENT.
      angfir=atan2(yfirst-ycent,xfirst-xcent);
      angstore(ieend)=0.;
      for m=1:nelseg; % each element in turn
        ieend=ieend+1;
        isegend(ieend)=fix(iseg);
        if(ratseg == 1.)
          ang=angseg*(m)/(nelseg);
        end;
        if(ratseg ~= 1.)
          ang=angseg*(1.-ratseg^m)/(1.-ratseg^nelseg);
        end;
        if(rseg < 0.)
          ang=-ang;
        end;
        xeend(ieend)=xcent+abs(rseg)*cos(angfir+ang);
        yeend(ieend)=ycent+abs(rseg)*sin(angfir+ang);
        angstore(ieend)=ang;
      end; % each element in turn
%
%  DEFINE THE ELEMENT NODES AND COORDINATES.
      ieend=ieend-nelseg;
      for ielseg=1:nelseg; % each element in turn
```

```
        ieend=ieend+1;
        m=mfirst(iseg)+ielseg-1;
        i1=2*m-1;
        i2=i1+1;
        i3=i1+2;
        if(iseg == isegmax && ielseg == nelseg)
          i3=node(mminb,1);
        end;
        node(m,1)=fix(i1);
        node(m,2)=fix(i2);
        node(m,3)=fix(i3);
        isegelem(m)=fix(iseg);
        if(iseg == isegmin && ielseg == 1)
          xnode(i1)=xeend(ieend-1);
          ynode(i1)=yeend(ieend-1);
        end;
        xnode(i3)=xeend(ieend);
        ynode(i3)=yeend(ieend);
        ang=0.5*(angstore(ieend-1)+angstore(ieend));
        xnode(i2)=xcent+abs(rseg)*cos(angfir+ang);
        ynode(i2)=ycent+abs(rseg)*sin(angfir+ang);
%
%  STORE THE NUMBERS OF THE ADJACENT ELEMENTS.
        m1(m)=fix(m-1);
        m3(m)=fix(m+1);
      end; % each element in turn
    end;
%
  end; % each segment in turn
%
%  ADJACENT ELEMENTS FOR END ELEMENTS OF THE BOUNDARY.
  m1(mminb)=fix(mmaxb);
  m3(mmaxb)=fix(mminb);
end; % each boundary in turn
neend=fix(ieend);
nnp=fix(nel*2);
%
%  DETERMINE THE MAXIMUM DIMENSION OF THE SOLUTION DOMAIN.
maxl=0.;
```

```
for i=1:nnp; % each node in turn
  for j=1:nnp; % each other node in turn
    dist=sqrt((xnode(i)-xnode(j))^2+(ynode(i)-ynode(j))^2);
      if(dist > maxl)
        maxl=dist;
      end;
  end; % each other node in turn
end; % each node in turn
%
return;
end %function meshq


function mshout(varargin)
%
%  SUBPROGRAM TO WRITE OUT THE MESH DATA.
%
global node maxl ynode xnode nnp nel fid7
%
%  OUTPUT THE NUMBERS OF ELEMENTS AND NODES, ALSO THE NODAL
%  COORDINATES.
fprintf(fid7,['GEOMETRIC DATA FOR THE MESH','\n','\n', ...
    '           NUMBER OF ELEMENTS =','%6i','\n','\n', ...
    '           NUMBER OF NODAL POINTS =','%6i','\n','\n', ...
    'COORDINATES OF THE NODAL POINTS','\n','\n', ...
    '     I      X           Y     ', ...
    '     I      X           Y     '],nel,nnp);
for i=1:nnp;
  if(mod(i,2) == 1);
    fprintf(fid7,['\n','%6i','%12.4e','%12.4e'], ...
        i,xnode(i),ynode(i));
  end;
  if(mod(i,2) == 0);
    fprintf(fid7,['%6i','%12.4e','%12.4e'], ...
        i,xnode(i),ynode(i));
  end;
end;
%
%  OUTPUT THE ELEMENT NODE NUMBERS.
```

```
fprintf(fid7,['\n','\n','ELEMENT NODE NUMBERS','\n','\n', ...
    '           M    ND1   ND2   ND3', ...
     '          M    ND1   ND2   ND3']);
for m=1:nel;
  if(mod(m,2) == 1)
    fprintf(fid7,['\n','       ','%5i'],m);
  end;
  if(mod(m,2) == 0)
    fprintf(fid7,['       ','%5i'],m);
  end;
  for in=1:3;
    fprintf(fid7,'%5i',node(m,in));
  end;
end;
%
%  SCALE THE NODAL POINT COORDINATES.
for i=1:nnp; % each node in turn
  xnode(i)=xnode(i)/maxl;
  ynode(i)=ynode(i)/maxl;
end; % each node in turn
```

```
%
return;
end %function mshout


function bcs(varargin)
%
%  SUBPROGRAM TO INPUT, PROCESS AND OUTPUT THE BOUNDARY CONDITIONS.
%
global node maxl beta dpsi ibcn ibce store m3 m1 nel nnp mlast mfirst ...
    betaseg alphaseg betan alpha alphan nsegtot isegbc nbcm dpsiseg ...
    nbcd psiseg nbcp maxnel nbct fid5 fid6
%
%  INPUT THE NUMBERS OF SEGMENTS SUBJECT TO EACH TYPE OF BOUNDARY
%  CONDITION.
%     NBCP - PRESCRIBED POTENTIAL.
%     NBCD - NON-ZERO PRESCRIBED NORMAL DERIVATIVE OF POTENTIAL.
%     NBCM - MIXED BOUNDARY CONDITION.
%  ANY SEGMENT NOT INCLUDED IS ASSUMED TO BE SUBJECT TO A ZERO
%  NORMAL DERIVATIVE OF POTENTIAL.
%
line=fgetl(fid5);
vec=str2num(line);
nbcp=vec(1); nbcd=vec(2); nbcm=vec(3);
%
%  TEST THESE BOUNDARY CONDITION NUMBERS.
nbct=fix(nbcp+nbcd+nbcm);
if(nbcp < 0 || nbcp > maxnel || nbcd < 0 || nbcd > maxnel ...
    || nbcm < 0 || nbcm > maxnel || nbct < 0 ||nbct > maxnel)
  fprintf(fid6,['\n','NBCP =','%6i','   NBCD =','%6i','   NBCM =', ...
    '%6i','\n','NBCT =','%6i','   OUTSIDE PERMITTED RANGE 0 TO', ...
    '%6i','\n'],nbcp,nbcd,nbcm,nbct,maxnel);
  error('numbers of boundary conditions error');
end;
%
%  INITIALISE THE BOUNDARY CONDITION STORAGE ARRAYS.
for m=1:nel; % each element in turn
  ibce(m)=2;
  store(m)=0.;
```

```
    alpha(m)=0.;
    beta(m)=0.;
  end; % each element in turn
  %
  %  INPUT, STORE AND OUTPUT THE PRESCRIBED POTENTIAL BOUNDARY CONDITIONS.
  if(nbcp > 0)
    for ibcp=1:nbcp;
      line=fgetl(fid5);
      vec=str2num(line);
      isegbc(ibcp)=vec(1); psiseg(ibcp)=vec(2);
    end;
    fprintf(fid6,['\n','PRESCRIBED POTENTIAL BOUNDARY CONDITIONS','\n']);
    for ibcp=1:nbcp; % each segment with prescribed potential
      iseg=isegbc(ibcp);
      if(iseg < 1 || iseg > nsegtot)
        fprintf(fid6,['\n','ISEG = ','%6i','  OUTSIDE PERMITTED RANGE' ...
            ' 1 TO','%6i','\n'],iseg,nsegtot);
        error('iseg error');
      end;
    for m=mfirst(iseg):mlast(iseg); %each element on current segment
      ibce(m)=1;
      store(m)=psiseg(ibcp);
    end; % each element on current segment
  %
    fprintf(fid6,['\n','POTENTIAL =','%12.4e','     ON ELEMENTS ', ...
      '%6i','     TO ','%6i','\n'],psiseg(ibcp),mfirst(iseg),mlast(iseg));
    end; % each segment with prescribed potential
  end;
  %
  %  INPUT, STORE AND OUTPUT THE PRESCRIBED POTENTIAL GRADIENT BOUNDARY
  %  CONDITIONS.
  if(nbcd > 0)
    for ibcd=1:nbcd;
      line=fgetl(fid5);
      vec=str2num(line);
      isegbc(ibcd)=vec(1);
      dpsiseg(ibcd)=vec(2);
    end;
    fprintf(fid6,['\n', ...
        'PRESCRIBED POTENTIAL GRADIENT BOUNDARY CONDITIONS','\n']);
    for ibcd=1:nbcd; % each segment with prescribed potential gradient
      iseg=isegbc(ibcd);
      if(iseg < 1 || iseg > nsegtot)
        fprintf(fid6,['\n','ISEG = ','%6i','  OUTSIDE PERMITTED RANGE' ...
            ' 1 TO','%6i','\n'],iseg,nsegtot);
        error('iseg error');
      end;
      for m=mfirst(iseg):mlast(iseg); % each element on current segment
        ibce(m)=2;
        store(m)=dpsiseg(ibcd);
      end; % each element on current segment
  %
      fprintf(fid6,['\n','GRADIENT =','%12.4e','     ON ELEMENTS ','%6i',
  ...
          '     TO ','%6i','\n'],dpsiseg(ibcd),mfirst(iseg),mlast(iseg));
    end; % each segment with prescribed potential gradient
  end;
  %
  %  INPUT, STORE AND OUTPUT THE MIXED BOUNDARY CONDITIONS.
  %  ASSUMED FORM - DPSI=ALPHA*PSI+BETA.
```

```
if(nbcm > 0)
  for ibcm=1:nbcm;
    line=fgetl(fid5);
    vec=str2num(line);
    isegbc(ibcm)=vec(1);
    alphaseg(ibcm)=vec(2);
    betaseg(ibcm)=vec(3);
  end;
  fprintf(fid6,['\n','PRESCRIBED MIXED BOUNDARY CONDITIONS','\n']);
  for ibcm=1:nbcm; % each segment with prescribed mixed condition
    iseg=isegbc(ibcm);
    if(iseg < 1 || iseg > nsegtot)
      fprintf(fid6,['\n','ISEG = ','%6i','  OUTSIDE PERMITTED RANGE' ...
          ' 1 TO','%6i','\n'],iseg,nsegtot);
      error('iseg error');
    end;
    for m=mfirst(iseg):mlast(iseg); % each element on current segment
      ibce(m)=3;
      alpha(m)=alphaseg(ibcm);
      beta(m)=betaseg(ibcm);
      for in=1:3; % each element node in turn
        i=node(m,in);
        alphan(i)=alpha(m);
        betan(i)=beta(m);
      end; % each element node in turn
    end; % each element on current segment
%
    fprintf(fid6,['\n','ALPHA =','%12.4e','  BETA =','%12.4e', ...
        '  ON ELEMENTS ','%3i','  TO ','%3i','\n'],alphaseg(ibcm), ...
        betaseg(ibcm),mfirst(iseg),mlast(iseg));
  end; % each segment with prescribed mixed condition
end;
%
%  ASSEMBLE THE VECTOR OF KNOWN VARIABLES, SCALING ANY PRESCRIBED
GRADIENTS.
for i=1:nnp; % each node in turn
  dpsi(i)=0.;
end; % each node in turn
for m=1:nel; % each element in turn
```

```
    for in=1:3; % each element node in turn
      i=node(m,in);
      ibcn(i)=fix(ibce(m));
  %
  %  CHECK THE CONDITION APPLIED TO THE ADJACENT ELEMENT FOR AN ELEMENT END
  %  NODE.  IF THE CONDITION IS PRESCRIBED POTENTIAL BUT THE EXISTING
  %  CONDITION AT THE NODE IS NOT, THEN IMPOSE PRESCRIBED POTENTIAL.
      madj=m;
      if(in == 1)
        madj=m1(m);
      end;
      if(in == 3)
        madj=m3(m);
      end;
      if(ibce(madj) == 1 && ibcn(i) ~= 1)
        ibcn(i)=1;
      end;
  %
  %  STORE KNOWN VARIABLE.
      if(ibcn(i) == 1)
        if(ibce(m) == 1)
          if(dpsi(i) ==  0.)
            dpsi(i)=store(m);
          end;
          if(dpsi(i) ~=  0.)
            dpsi(i)=0.5*(dpsi(i)+store(m));
          end;
        end;
      end;
      if(ibce(m) == 2 && ibcn(i) == 2)
        dpsi(i)=store(m)*maxl;
      end;
      if(ibce(m) == 3 && ibcn(i) == 3)
        dpsi(i)=beta(m)*maxl;
      end;
    end; % each element node in turn
  end; % each element in turn
  %
  return;
  end %function bcs
```

```
function poisson(varargin)
%
%  MODIFY BOUNDARY CONDITIONS IF GOVERNING EQUATION IS POISSON TYPE.
%
global node maxl psipi alphan dpsipi dpsi ibcn nnp unmy unmx f1 ...
    dpsipim yc ynode xc xnode nel uny unx ibce m3 m1 ungy ungx
%
%  INITIALISE THE PARTICULAR INTEGRAL ARRAYS.
for i=1:nnp; % each node in turn
  psipi(i)=0.;
  dpsipi(i)=0.;
end; % each node in turn
for m=1:nel; % each element in turn
  for in=1:3; % each element node in turn
    dpsipim(m,in)=0.;
  end; % each element node in turn
end; % each element in turn
%
%  UNIT NORMALS AT THE NODES ARE REQUIRED FOR PARTICULAR INTEGRAL
%  DERIVATIVES THERE.
%
%  FIRST FIND THE NORMALS AT THE ELEMENT NODES.
for m=1:nel; % each element in turn
  for in=1:3; % each element node in turn
    igauss=in+10;
    it=1;
    ic=1;
    jacobi(m,igauss,it,ic);
    unmx(m,in)=ungx;
    unmy(m,in)=ungy;
  end; % each element node in turn
end; % each element in turn
%
%  MERGE ELEMENT NODE NORMALS TO FIND NORMALS AT THE NODES.
for i=1:nnp; % each node in turn
  unx(i)=0.;
  uny(i)=0.;
end; % each node in turn
for m=1:nel; % each element in turn
%
%  FIRST NODE.
  i=node(m,1);
  madj=m1(m);
%
%  ICASE=1 BOTH CURRENT AND ADJACENT ELEMENT HAVE EITHER PRESCRIBED
%          POTENTIAL OR PRESCRIBED GRADIENT/MIXED BOUNDARY CONDITIONS.
%  ICASE=2 CURRENT ELEMENT HAS PRESCRIBED POTENTIAL, ADJACENT HAS
%          PRESCIBED GRADIENT/MIXED.
%  ICASE=3 CURRENT ELEMENT HAS PRESCRIBED GRADIENT/MIXED CONDITION,
%          ADJACENT HAS PRESCRIBED POTENTIAL.
  if(ibce(m) == 1 && ibce(madj) == 1)
    icase=1;
  end;
  if(ibce(m) ~= 1 && ibce(madj) ~= 1)
    icase=1;
  end;
  if(ibce(m) == 1 && ibce(madj) ~= 1)
    icase=2;
  end;
```

```
    if(ibce(m) ~= 1 && ibce(madj) == 1)
      icase=3;
    end;
    if(icase == 1)
      unx(i)=unx(i)+unmx(m,1);
      uny(i)=uny(i)+unmy(m,1);
    end;
    if(icase == 2)
      unx(i)=unmx(m,1);
      uny(i)=unmy(m,1);
    end;
  %
  %  SECOND NODE.
    i=node(m,2);
    unx(i)=unmx(m,2);
    uny(i)=unmy(m,2);
  %
  %  THIRD NODE.
    i=node(m,3);
    madj=m3(m);
    if(ibce(m) ~= 1 && ibce(madj) ~= 1)
      icase=1;
    end;
    if(ibce(m) == 1 && ibce(madj) == 1)
      icase=1;
    end;
    if(ibce(m) == 1 && ibce(madj) ~= 1)
      icase=2;
    end;
    if(ibce(m) ~= 1 && ibce(madj) == 1)
      icase=3;
    end;
    if(icase == 1)
      unx(i)=unx(i)+unmx(m,3);
      uny(i)=uny(i)+unmy(m,3);
    end;
    if(icase == 2)
      unx(i)=unmx(m,3);
      uny(i)=unmy(m,3);
    end;
  end; % each element in turn
```

```
%
%  FOR ELEMENT END NODES WITH ICASE=1, CONTRIBUTIONS TO THE NORMALS
%  WILL HAVE COME FROM TWO ELEMENTS, EFFECTIVELY GIVING AVERAGE
%  (VECTOR SUM) NORMALS, BUT WHICH MUST NOW BE REDUCED TO UNIT
%  NORMALS.
for i=1:2:nnp; % each node in turn
  denom=sqrt(unx(i)^2+uny(i)^2);
  unx(i)=unx(i)/denom;
  uny(i)=uny(i)/denom;
end; % each node in turn
%
%  FIND AND STORE THE POTENTIAL AND POTENTIAL GRADIENT AT EVERY NODE
%  DUE TO THE PARTICULAR INTEGRAL.
if(f1 == 0.)
  return;
end;
xc=xc/maxl;
yc=yc/maxl;
f1=f1*maxl^2;
for i=1:nnp; % each node in turn
```

```
    xx=xnode(i)-xc;
    yy=ynode(i)-yc;
    psipi(i)=0.25*f1*(xx^2+yy^2);
    dpsipi(i)=0.5*f1*(xx*unx(i)+yy*uny(i));
  end; % each node in turn
  %
  %  FIND AND STORE THE POTENTIAL GRADIENT AT EVERY NODE OF EVERY
  %  ELEMENT DUE TO THE PARTICULAR INTEGRAL.
  for m=1:nel; % each element in turn
    for in=1:3; % each element node in turn
      i=node(m,in);
      xx=xnode(i)-xc;
      yy=ynode(i)-yc;
      dpsipim(m,in)=0.5*f1*(xx*unmx(m,in)+yy*unmy(m,in));
    end; % each element node in turn
  end; % each element in turn
  %
  %  SUBTRACT THE PARTICULAR INTEGRAL CONTRIBUTIONS FROM THE PRESCRIBED
  %  BOUNDARY CONDITIONS.
  for i=1:nnp; % each node in turn
    if(ibcn(i) == 1)
      dpsi(i)=dpsi(i)-psipi(i);
    end;
    if(ibcn(i) == 2)
      dpsi(i)=dpsi(i)-dpsipi(i);
    end;
    if(ibcn(i) == 3)
      dpsi(i)=dpsi(i)-dpsipi(i)+alphan(i)*psipi(i)*maxl;
    end;
  end; % each node in turn
  %
  return;
  end %function poisson


  function frmtrx(varargin)
  %
  %  SUBPROGRAM TO FORM THE COEFFICIENT MATRIX AND RIGHT HAND SIDE VECTOR,
  %  MODIFIED TO SUIT THE BOUNDARY CONDITIONS.
  %
```

```
global node a maxl dpsi alpha brow arow ibcn dpsipim beta psipi ...
    ibce store nel nnp
%
%  DEFINE THE NUMBER OF COLUMNS IN THE EXTENDED COEFFICIENT MATRIX A.
jmax=nnp+1;
%
%  FORM THE COEFFICIENT MATRIX A, AND THE RIGHT HAND SIDE VECTOR B*DPSI.
for i=1:nnp; % take each node in turn as P
%
%  INITIALISE CURRENT ROW OF MATRIX A.
  for j=1:jmax; % each coefficient in turn
    a(i,j)=0.;
  end; % each coefficient in turn
%
%  INITIALISE ELEMENT CONTRIBUTIONS TO CURRENT ROWS OF A AND B MATRICES.
  for m=1:nel; % each element in turn
    for in=1:3; % each element node in turn
      arow(m,in)=0.;
      brow(m,in)=0.;
    end; % each element node in turn
  end; % each element in turn
%
%  SET UP THE LOOP TO INTEGRATE OVER EACH ELEMENT IN TURN.
  for m=1:nel; % each element in turn
%
%  INTEGRATE THE KERNEL PRODUCTS OVER THE CURRENT ELEMENT.
    intker(i,m);
  end; % each element in turn
%
%  EVALUATE THE DIAGONAL COEFFICIENT OF MATRIX A.
  aii=0.;
  for m=1:nel; % each element in turn
    for in=1:3; % each element node in turn
      aii=aii-arow(m,in);
    end; % each element node in turn
  end; % each element in turn
  if(ibcn(i) ~= 1)
    a(i,i)=aii;
  end;
%
```

```
%  INITIALISE THE B*DPSI VECTOR COEFFICIENT.
  bdpsi=0.;
  if(ibcn(i) == 1)
    bdpsi=-aii*dpsi(i);
  end;
%
%  APPLY THE BOUNDARY CONDITIONS TO THE CURRENT ROWS OF A AND B, BY
%  CONSIDERING EACH ELEMENT IN TURN.
  for m=1:nel; % each element in turn
    for in=1:3; % each element node in turn
      j=node(m,in);
%
%  IF POTENTIAL IS PRESCRIBED OVER THE ELEMENT, INTERCHANGE THE A AND
%  B COEFFICIENTS.
      if(ibce(m) == 1)
        a(i,j)=a(i,j)-brow(m,in);
        bdpsi=bdpsi-arow(m,in)*(store(m)-psipi(j));
      end;
%
%  IF POTENTIAL GRADIENT IS PRESCRIBED OVER THE ELEMENT (WHICH MAY
%  INCLUDE A CORNER NODE WITH MIXED BOUNDARY CONDITION), STORE THE A
%  MATRIX COEFFICIENTS, EXCEPT AT A CORNER NODE WHERE PRESCRIBED
%  POTENTIAL HAS BEEN IMPOSED.
    if(ibce(m) == 2)
      bdpsi=bdpsi+brow(m,in)*(store(m)*maxl-dpsipim(m,in));
      if(ibcn(j) ~= 1)
        a(i,j)=a(i,j)+arow(m,in);
      end;
      if(ibcn(j) == 1)
        bdpsi=bdpsi-arow(m,in)*dpsi(j);
      end;
    end;
%
%  IF BOUNDARY CONDITION OVER THE ELEMENT IS MIXED (WHICH MAY
%  INCLUDE A CORNER NODE WITH PRESCRIBED POTENTIAL GRADIENT),
%  MODIFY THE A MATRIX COEFFICIENTS, EXCEPT AT A CORNER NODE
%  WHERE PRESCRIBED POTENTIAL HAS BEEN IMPOSED.
    if(ibce(m) == 3)
      dpsistore=alpha(m)*psipi(j)+beta(m);
      dpsistore=dpsistore*maxl-dpsipim(m,in);
      bdpsi=bdpsi+brow(m,in)*dpsistore;
      if(ibcn(j) ~= 1)
        a(i,j)=a(i,j)+arow(m,in)-brow(m,in)*alpha(m)*maxl;
      end;
      if(ibcn(j) == 1)
        bdpsi=bdpsi-arow(m,in)*dpsi(j);
        bdpsi=bdpsi+brow(m,in)*alpha(m)*dpsi(j)*maxl;
      end;
    end;
%
    end; % each element node in turn
  end; % each element in turn
%
%  STORE THE B*DPSI COEFFICIENT AS AN EXTENSION OF MATRIX A.
  a(i,jmax)=bdpsi;
end; % take each node in turn as P
%
return;
end %function frmtrx
```

```
function shape(varargin)
%
%  SUBPROGRAM TO EVALUATE AND STORE VALUES OF THE SHAPE FUNCTIONS AND
%  THEIR DERIVATIVES, AT THE GAUSS POINTS AND NODES.
%
global sd sdl sfl egl ngauss sf zg wgl wg
%
%  STORE APPROPRIATE COORDINATES AND WEIGHT FACTORS FOR NORMAL GAUSSIAN
%  QUADRATURE IN ARRAYS XG AND CG, ALSO THOSE FOR LOGARITHMIC FUNCTION
%  INTEGRATION IN XGL AND CGL.
ngauss=8;
zg(1)=-0.9602898564;
zg(2)=-0.7966664774;
zg(3)=-0.5255324099;
zg(4)=-0.1834346424;
zg(5)=-zg(4);
zg(6)=-zg(3);
zg(7)=-zg(2);
zg(8)=-zg(1);
wg(1)=0.1012285362;
wg(2)=0.2223810344;
wg(3)=0.3137066458;
wg(4)=0.3626837833;
wg(5)=wg(4);
wg(6)=wg(3);
wg(7)=wg(2);
wg(8)=wg(1);
egl(1)=0.013320244;
egl(2)=0.079750429;
egl(3)=0.197871029;
egl(4)=0.354153994;
egl(5)=0.529458575;
egl(6)=0.701814530;
egl(7)=0.849379320;
egl(8)=0.953326450;
wgl(1)=0.164416605;
wgl(2)=0.237525610;
wgl(3)=0.226841984;
wgl(4)=0.175754079;
wgl(5)=0.112924030;
wgl(6)=0.057872211;
wgl(7)=0.020979074;
wgl(8)=0.003686407;
%
%  NORMAL GAUSSIAN QUADRATURE.
for igauss=1:ngauss; % each gauss point in turn
  zeta=zg(igauss);
  sf(1,igauss)=0.5*zeta*(zeta-1.);
  sf(2,igauss)=1.-zeta^2;
  sf(3,igauss)=0.5*zeta*(zeta+1.);
  sd(1,igauss)=zeta-0.5;
  sd(2,igauss)=-2.*zeta;
  sd(3,igauss)=zeta+0.5;
end; % each gauss point in turn
%
%  FOUR CASES OF LOGARITHMIC GAUSSIAN QUADRATURE TO CONSIDER.
%     IC=1 - INTEGRATION OVER WHOLE ELEMENT FROM FIRST TO THIRD NODE.
%      IC=2 - INTEGRATION OVER HALF ELEMENT FROM SECOND TO THIRD NODE.
%      IC=3 - INTEGRATION OVER HALF ELEMENT FROM SECOND TO FIRST NODE.
%      IC=4 - INTEGRATION OVER WHOLE ELEMENT FROM THIRD TO FIRST NODE.
%
for ic=1:4; % each case in turn
  for igauss=1:ngauss; % each gauss point in turn
    eta=egl(igauss);
    if(ic == 1)
      zeta=2.*eta-1.;
    end;
    if(ic == 2)
      zeta=eta;
    end;
    if(ic == 3)
      zeta=-eta;
    end;
    if(ic == 4)
      zeta=1.-2.*eta;
    end;
    sfl(ic,1,igauss)=0.5*zeta*(zeta-1.);
    sfl(ic,2,igauss)=1.-zeta^2;
    sfl(ic,3,igauss)=0.5*zeta*(zeta+1.);
    sdl(ic,1,igauss)=zeta-0.5;
    sdl(ic,2,igauss)=-2.*zeta;
    sdl(ic,3,igauss)=zeta+0.5;
  end; % each gauss point in turn
end; % each case in turn
%
%  SHAPE FUNCTION DERIVATIVES AT THE NODES, STORED AS THOUGH THEY
%  ARE FOR GAUSS POINTS NUMBERED 11, 12 AND 13.
for igauss=11:13; % each element node in turn
  if(igauss == 11)
    zeta=-1.;
  end;
  if(igauss == 12)
    zeta=0.;
  end;
  if(igauss == 13)
    zeta=1.;
  end;
  sd(1,igauss)=zeta-0.5;
  sd(2,igauss)=-2.*zeta;
  sd(3,igauss)=zeta+0.5;
end; % each element node in turn
%
return;
end %function shape


function intker(i,m)
%
%  SUBPROGRAM TO INTEGRATE THE KERNEL PRODUCTS FOR A PARTICULAR SOURCE
%  POINT P (INDICATED BY NODE NUMBER I) OVER A PARTICULAR ELEMENT
%  (INDICATED BY ARGUMENT M) BY GAUSSIAN QUADRATURE.
%
global node jacob wgl brow sfl ngauss wg sf arow ynode xnode
%
%  COORDINATES OF POINT P.
xp=xnode(i);
yp=ynode(i);
%
%  SET UP THE ELEMENT NODE LOOP.
for in=1:3; % each element node in turn
  j=node(m,in);
%
%  IF P IS NOT THE CURRENT ELEMENT NODE, USE NORMAL GAUSSIAN QUADRATURE.
  if(i ~= j)
    for igauss=1:ngauss; % each gauss point in turn
%
%  EVALUATE JACOBIAN AND UNIT NORMAL VECTOR COMPONENTS, ALSO THE KERNELS
%  AT THE PARTICULAR GAUSS POINT FOR NORMAL QUADRATURE OVER THE WHOLE
%  ELEMENT.
      it=1;
      ic=1;
      jacobi(m,igauss,it,ic);
      [ak,bk]=kernel(xp,yp,m,igauss);
%
```

```
%  ACCUMULATE THE INTEGRALS.
      sfn=sf(in,igauss);
      arow(m,in)=arow(m,in)+wg(igauss)*ak*sfn*jacob;
      brow(m,in)=brow(m,in)+wg(igauss)*bk*sfn*jacob;
    end; % each gauss point in turn
  end;
%
%  IF P IS THE CURRENT ELEMENT NODE, SOME LOGARITHMIC QUADRATURE
%  IS REQUIRED.
  if(i == j)
%
%  P AT THE FIRST NODE OF THE ELEMENT.
    if(in == 1)
%
%  TERM INVOLVING NORMAL QUADRATURE.
      for igauss=1:ngauss; % each gauss point in turn
        it=1;
        ic=1;
        jacobi(m,igauss,it,ic);
        bk2=kern2(m,in,igauss);
        sfn=sf(in,igauss);
        brow(m,in)=brow(m,in)+wg(igauss)*bk2*sfn*jacob;
      end; % each gauss point in turn
%
%  TERM INVOLVING LOGARITHMIC QUADRATURE.
      for igauss=1:ngauss; % each gauss point in turn
        it=2;
        ic=1;
        jacobi(m,igauss,it,ic);
        sfn=sfl(ic,in,igauss);
        dzde=2.;
        brow(m,in)=brow(m,in)+wgl(igauss)*sfn*jacob*dzde;
      end; % each gauss point in turn
    end;
%
%  P AT THE SECOND NODE OF THE ELEMENT.
    if(in == 2)
%
%  TERMS INVOLVING NORMAL QUADRATURE.
```

```
      for igauss=1:ngauss; % each gauss point in turn
        it=1;
        ic=1;
        jacobi(m,igauss,it,ic);
        bk2=kern2(m,in,igauss);
        sfn=sf(in,igauss);
        brow(m,in)=brow(m,in)+wg(igauss)*bk2*sfn*jacob;
      end; % each gauss point in turn
%
%  TERMS INVOLVING LOGARITHMIC QUADRATURE.
      for igauss=1:ngauss; % each gauss point in turn
        it=2;
        ic=2;
        jacobi(m,igauss,it,ic);
        sfn=sfl(ic,in,igauss);
        dzde=1.;
        brow(m,in)=brow(m,in)+wgl(igauss)*sfn*jacob*dzde;
        ic=3;
        jacobi(m,igauss,it,ic);
        sfn=sfl(ic,in,igauss);
        dzde=1.;
        brow(m,in)=brow(m,in)+wgl(igauss)*sfn*jacob*dzde;
      end; % each gauss point in turn
    end;
%
%  P AT THE THIRD NODE OF THE ELEMENT.
    if(in == 3)
%
%  TERM INVOLVING NORMAL QUADRATURE.
      for igauss=1:ngauss; % each gauss point in turn
        it=1;
        ic=1;
        jacobi(m,igauss,it,ic);
        bk2=kern2(m,in,igauss);
        sfn=sf(in,igauss);
        brow(m,in)=brow(m,in)+wg(igauss)*bk2*sfn*jacob;
      end; % each gauss point in turn
%
%  TERM INVOLVING LOGARITHMIC QUADRATURE.
      for igauss=1:ngauss; % each gauss point in turn
```

```
         it=2;
         ic=4;
         jacobi(m,igauss,it,ic);
         sfn=sfl(ic,in,igauss);
         dzde=2.;
         brow(m,in)=brow(m,in)+wgl(igauss)*sfn*jacob*dzde;
       end; % each gauss point in turn
     end;
   end;
 end; % each element node in turn
 %
 return;
 end %function intker


 function jacobi(m,igauss,it,ic)
 %
 %  SUBPROGRAM TO EVALUATE THE JACOBIAN AND THE COMPONENTS OF THE UNIT
 %  NORMAL VECTOR AT A PARTICULAR GAUSS POINT.
 %  M INDICATES THE ELEMENT NUMBER.
 %  IGAUSS INDICATES THE GAUSS POINT NUMBER.
 %  IT INDICATES THE TYPE OF QUADRATURE.
 %     IT=1 - NORMAL GAUSSIAN QUADRATURE.
 %     IT=2 - LOGARITHMIC GAUSSIAN QUADRATURE.
 %  IC INDICATES THE CASE NUMBER FOR LOGARITHMIC GAUSSIAN QUADRATURE,
 %  AS DEFINED IN SUBROUTINE SHAPE.
 %
 global jacob ungy ungx node ynode xnode sdl sd
 %
 %  CALCULATE THE DERIVATIVES OF THE GLOBAL COORDINATES WITH RESPECT TO
 %  THE LOCAL INTRINSIC COORDINATE.
 dx=0.;
 dy=0.;
 for in=1:3; % each element node in turn
   if(it == 1)
     sfnd=sd(in,igauss);
   end;
   if(it == 2)
     sfnd=sdl(ic,in,igauss);
   end;
```

```
  j=node(m,in);
  dx=dx+sfnd*xnode(j);
  dy=dy+sfnd*ynode(j);
end; % each element node in turn
%
%  COMPONENTS OF THE LOCAL OUTWARD NORMAL VECTOR AT THE GAUSS POINT.
ungx=dy;
ungy=-dx;
%
%  JACOBIAN OF THE COORDINATE TRANSFORMATION.
jacob=sqrt(ungx^2+ungy^2);
%
%  SCALE THE VECTOR COMPONENTS TO GIVE THE UNIT NORMAL VECTOR.
ungx=ungx/jacob;
ungy=ungy/jacob;
%
return;
end %function jacobi


function [ak,bk]=kernel(xp,yp,m,igauss)
%
%  SUBPROGRAM TO EVALUATE THE KERNELS WHEN P IS NOT THE CURRENT
%  ELEMENT NODE.
%  XP, YP INDICATE THE GLOBAL COORDINATES OF POINT P.
%  M INDICATES THE ELEMENT NUMBER.
%  IGAUSS INDICATES THE NUMBER OF THE GAUSS POINT, Q.
%
global ungy ungx node ynode xnode sf;
%
%  COORDINATES OF GAUSS POINT Q.
xq=0.;
yq=0.;
for in=1:3; % each element node in turn
  sfn=sf(in,igauss);
  j=node(m,in);
  xq=xq+sfn*xnode(j);
  yq=yq+sfn*ynode(j);
end; % each element node in turn
```

```
%
%  COMPONENTS AND MAGNITUDE OF THE RADIUS VECTOR FROM P TO Q.
rx=xq-xp;
ry=yq-yp;
rsq=rx^2+ry^2;
r=sqrt(rsq);
%
%  EVALUATE THE KERNELS.
ak=-(rx*ungx+ry*ungy)/rsq;
bk=log(1./r);
%
return;
end %function kernel


function bk2=kern2(m,in,igauss)
%
%  SUBPROGRAM TO EVALUATE THE NON-SINGULAR LOGARITHMIC TERM IN THE
%  SECOND KERNEL WHEN P IS THE CURRENT ELEMENT NODE.
%  M INDICATES THE ELEMENT NUMBER.
%  IN INDICATES THE NUMBER OF THE ELEMENT NODE FORMING P.
%  IGAUSS INDICATES THE GAUSS POINT NUMBER.
%
global ynode xnode zg node
%
%  ELEMENT NODE NUMBERS.
i1=node(m,1);
i2=node(m,2);
i3=node(m,3);
%
%  EVALUATE THE INTRINSIC COORDINATE.
zeta=zg(igauss);
%
%  P AT THE FIRST NODE OF THE ELEMENT.
if(in == 1)
  xcomp=(zeta-2.)*xnode(i1)+2.*(1.-zeta)*xnode(i2)+zeta*xnode(i3);
  ycomp=(zeta-2.)*ynode(i1)+2.*(1.-zeta)*ynode(i2)+zeta*ynode(i3);
  rfn=sqrt(xcomp^2+ycomp^2);
end;
```

```
%
%  P AT THE SECOND NODE OF THE ELEMENT.
if(in == 2)
  xcomp=-0.5*(zeta-1.)*xnode(i1)+zeta*xnode(i2)-0.5*(zeta+1.)*xnode(i3);
  ycomp=-0.5*(zeta-1.)*ynode(i1)+zeta*ynode(i2)-0.5*(zeta+1.)*ynode(i3);
  rfn=sqrt(xcomp^2+ycomp^2);
end;
%
%  P AT THE THIRD NODE OF THE ELEMENT.
if(in == 3)
  xcomp=-zeta*xnode(i1)+2.*(1.+zeta)*xnode(i2)-(zeta+2.)*xnode(i3);
  ycomp=-zeta*ynode(i1)+2.*(1.+zeta)*ynode(i2)-(zeta+2.)*ynode(i3);
  rfn=sqrt(xcomp^2+ycomp^2);
end;
%
%  EVALUATE THE KERNEL TERM.
bk2=log(1./rfn);
%
return;
end %function kern2


function output(varargin)
%
%  SUBPROGRAM TO WRITE OUT THE NODAL POINT VALUES OF POTENTIAL AND
%  POTENTIAL GRADIENT, AND COMPUTE POTENTIAL FLOWS ACROSS THE
%  BOUNDARY SEGMENTS.
%
global nsegtot flowseg node maxl jacob wg sf ngauss beta psit alpha ...
    ibce store dpsit isegelem nel uny unx dpsipi psipi f1 dpsi psi ...
    nnp betan alphan ibcn fid6
%
%  ARRANGE FOR PSI AND DPSI TO CONTAIN THE POTENTIALS AND POTENTIAL
%  GRADIENTS, RESPECTIVELY.
for i=1:nnp; % each node in turn
  if(ibcn(i) == 1)
    temp=psi(i);
    psi(i)=dpsi(i);
    dpsi(i)=temp;
  end;
  if(ibcn(i) == 3)
    psitot=psi(i)+psipi(i);
    dpsi(i)=(alphan(i)*psitot+betan(i))*maxl-dpsipi(i);
  end;
end; % each node in turn
%
%  ADD CONTRIBUTIONS OF THE PARTICULAR INTEGRAL.
fprintf(fid6,['\n','NODAL POINT POTENTIALS AND POTENTIAL GRADIENTS' ...
    ,'\n','\n','     I      PSI            DPSI           UNX' ...
    '          UNY','\n']);
for i=1:nnp; % each node in turn
  psit(i)=psi(i);
  dpsit(i)=dpsi(i);
  if(f1 ~= 0.)
    psit(i)=psit(i)+psipi(i);
    dpsit(i)=dpsit(i)+dpsipi(i);
  end;
%
%  REMOVE THE LENGTH SCALE FACTOR FROM THE GRADIENT VALUES.
  dpsit(i)=dpsit(i)/maxl;
%
%  OUTPUT THE NODAL VALUES OF POTENTIAL, POTENTIAL GRADIENT,
%  AND THE UNIT NORMAL COMPONENTS.
  fprintf(fid6,['%6i','%15.6e','%15.6e','%13.4e','%13.4e','\n'], ...
      i,psit(i),dpsit(i),unx(i),uny(i));
end; % each node in turn
%
%  COMPUTE THE POTENTIAL FLOWS ACROSS THE BOUNDARY SEGMENTS.
flowin=0.;
flowout=0.;
for iseg=1:nsegtot; % each segment in turn
  flowseg(iseg)=0.;
end; % each segment in turn
for m=1:nel; % each element in turn
  iseg=isegelem(m);
%
%  APPLY GAUSSIAN QUADRATURE.
  flowelem=0.;
```

```
    for in=1:3; % each element node in turn
      j=node(m,in);
  %
  %  FIND THE POTENTIAL GRADIENT AT THE NODE.
      if(ibce(m) == 1)
        dpsistore=dpsit(j);
      end;
      if(ibce(m) == 2)
        dpsistore=store(m);
      end;
      if(ibce(m) == 3)
        dpsistore=alpha(m)*psit(j)+beta(m);
      end;
      for igauss=1:ngauss; % each gauss point in turn
        sfn=sf(in,igauss);
        it=1;
        ic=1;
        jacobi(m,igauss,it,ic);
        flowelem=flowelem+wg(igauss)*sfn*jacob*dpsistore*maxl;
      end; % each gauss point in turn
    end; % each element node in turn
  %
  %  ACCUMULATE THE FLOW ACROSS THE SEGMENT, AND THE TOTAL FLOWS IN
  %  AND OUT OF THE DOMAIN.
    flowseg(iseg)=flowseg(iseg)+flowelem;
    if(flowelem > 0.)
      flowin=flowin+flowelem;
    end;
    if(flowelem < 0.)
      flowout=flowout-flowelem;
    end;
  end; % each element in turn
  %
  %  OUTPUT THE POTENTIAL FLOW RESULTS.
  fprintf(fid6,['\n', ...
      'POTENTIAL FLOWS INTO DOMAIN ACROSS BOUNDARY SEGMENTS','\n','\n', ...
      'SEGMENT      FLOW','\n']);
  for iseg=1:nsegtot;
    fprintf(fid6,['%5i','%16.6e','\n'],iseg,flowseg(iseg));
  end;
```

```
fprintf(fid6,['\n','TOTAL POTENTIAL FLOW INTO DOMAIN =','%15.6e', ...
    '\n','\n','TOTAL POTENTIAL FLOW OUT OF DOMAIN =','%15.6e', ...
    '\n'],flowin,flowout);
%
return;
end %function output


function internal(varargin)
%
%  SUBPROGRAM TO COMPUTE AND OUTPUT THE POTENTIALS AT SELECTED
%  INTERNAL POINTS.
%
global f1 yc xc node psi jacob wg sf ngauss dpsipim maxl beta psit ...
    alpha ibce store dpsi nel fid5 fid6
%
%  INPUT NUMBER OF INTERNAL POINTS.
line=fgetl(fid5);
nint=str2num(line);
if(nint == 0)
  return;
end;
fprintf(fid6,['\n','POTENTIALS AT INTERNAL POINTS','\n','\n', ...
    '     PSI                X           Y']);
%
%  INPUT THE INTERNAL POINT COORDINATES.
for iint=1:nint; % each internal point in turn
  line=fgetl(fid5);
  vec=str2num(line);
  xint=vec(1); yint=vec(2);
  xp=xint/maxl;
  yp=yint/maxl;
%
%  INTEGRATE ROUND THE BOUNDARY.
  sum=0.;
  for m=1:nel; % each element in turn
    for in=1:3; % each element node in turn
      j=node(m,in);
%
```

```
%  FIND THE POTENTIAL GRADIENT AT THE NODE, EITHER AS COMPUTED
%  IF POTENTIAL WAS PRESCRIBED, OR FROM THE PRESCRIBED GRADIENT
%  OR MIXED BOUNDARY CONDITIONS.
      if(ibce(m) == 1)
        dpsistore=dpsi(j);
      end;
      if(ibce(m) == 2)
        dpsistore=store(m)*maxl-dpsipim(m,in);
      end;
      if(ibce(m) == 3)
        dpsistore=(alpha(m)*psit(j)+beta(m))*maxl-dpsipim(m,in);
      end;
%
%  APPLY GAUSSIAN QUADRATURE.
      for igauss=1:ngauss; % each gauss point in turn
        it=1;
        ic=1;
        jacobi(m,igauss,it,ic);
        [ak,bk]=kernel(xp,yp,m,igauss);
        sfn=sf(in,igauss);
        sum=sum+wg(igauss)*sfn*jacob*(-ak*psi(j)+bk*dpsistore);
      end; % each gauss point in turn
    end; % each element node in turn
  end; % each element in turn
  psiip=sum*0.5/pi;
  psiipt=psiip;
%
%  ADD THE PARTICULAR INTEGRAL FOR A POISSON TYPE PROBLEM.
  if(f1 ~= 0.)
    xx=xp-xc;
    yy=yp-yc;
    psiipt=psiipt+0.25*f1*(xx^2+yy^2);
  end;
%
%  OUTPUT THE POTENTIAL AT THE INTERNAL POINT.
  fprintf(fid6,['\n','%14.6e','     ','%12.4e','%12.4e'], ...
      psiipt,xint,yint);
end; % each internal point in turn
%
return;
```

```
end %function internal



function [x,iflag]=elimin(a,neqn)
%
%  SUBROUTINE FOR SOLVING SIMULTANEOUS LINEAR EQUATIONS BY GAUSSIAN
%  ELIMINATION WITH PARTIAL PIVOTING.
%
%  INITIALIZE ILL-CONDITIONING FLAG.
iflag=0;
for i=1:neqn; x(i)=0.; end;
%
%  SCALE EACH EQUATION TO HAVE A MAXIMUM COEFFICIENT MAGNITUDE OF UNITY.
jmax=neqn+1;
for i=1:neqn; % each equation in turn
  amax=0.;
  for j=1:neqn; % search for maximum
    absa=abs(a(i,j));
    if(absa > amax)
      amax=absa;
    end;
  end; % search for maximum
%
  for j=1:jmax; % scale coefficients
    a(i,j)=a(i,j)/amax;
  end; % scale coefficients
%
end; % each equation in turn
%
%  COMMENCE ELIMINATION PROCESS.
for k=1:neqn-1; % eliminate each variable in turn
%
%  SEARCH LEADING COLUMN OF THE COEFFICIENT MATRIX FROM THE DIAGONAL
%  DOWNWARDS FOR THE LARGEST VALUE.
  imax=k;
  for i=k+1:neqn; % search for largest value
    if(abs(a(i,k)) > abs(a(imax,k)))
      imax=i;
    end;
  end; % search for largest value
%
%  IF NECESSARY, INTERCHANGE EQUATIONS TO MAKE THE LARGEST COEFFICIENT
%  BECOME THE PIVOTAL COEFFICIENT.
  if(imax ~= k)
    for j=k:jmax; % interchange coefficients
      atemp=a(k,j);
      a(k,j)=a(imax,j);
      a(imax,j)=atemp;
    end; % interchange coefficients
  end;
%
%  ELIMINATE X(K) FROM EQUATIONS (K+1) TO NEQN, FIRST TESTING FOR
%  EXCESSIVELY SMALL PIVOTAL COEFFICIENT (ASSOCIATED WITH A SINGULAR
%  OR VERY ILL-CONDITIONED MATRIX).
  if(abs(a(k,k)) < 1.0e-5)
    iflag=1;
    return;
  end;
  for i=k+1:neqn; % each of remaining equations
    fact=a(i,k)/a(k,k);
    for j=k:jmax; % modify coefficients
      a(i,j)=a(i,j)-fact*a(k,j);
    end; % modify coefficient
  end; % each of remaining equations
%
end; % eliminate each variable in turn
%
%  SOLVE THE EQUATIONS BY BACK SUBSTITUTION, FIRST TESTING
%  FOR AN EXCESSIVELY SMALL LAST DIAGONAL COEFFICIENT.
if(abs(a(neqn,neqn)) < 1.0e-5)
  iflag=1;
  return;
end;
x(neqn)=a(neqn,jmax)/a(neqn,neqn);
for i=neqn-1:-1:1; % then each unknown in turn backwards
  sum=a(i,jmax);
  for j=i+1:neqn; % sum products
    sum=sum-a(i,j)*x(j);
  end; % sum products
  x(i)=sum/a(i,i);
end; % then each unknown in turn backwards
return;
end %function elimin
```

The equivalent of the Fortran SHAREDDDATA2PQ is shareddata2pq, stored as the file shareddata2pq.m, is as follows.

```
  %%%--- module shareddata2pq;
  %
  % module STORING SHARED DATA.
  %
   global xeend; if isempty(xeend), xeend=zeros(1,510); end;
   global yeend; if isempty(yeend), yeend=zeros(1,510); end;
   global xnode; if isempty(xnode), xnode=zeros(1,1000); end;
   global ynode; if isempty(ynode), ynode=zeros(1,1000); end;
   global xsend; if isempty(xsend), xsend=zeros(1,500); end;
   global ysend; if isempty(ysend), ysend=zeros(1,500); end;
   global unx; if isempty(unx), unx=zeros(1,1000); end;
   global uny; if isempty(uny), uny=zeros(1,1000); end;
   global unmx; if isempty(unmx), unmx=zeros(500,3); end;
   global unmy; if isempty(unmy), unmy=zeros(500,3); end;
   global dpsipim; if isempty(dpsipim), dpsipim=zeros(500,3); end;
   global f1; if isempty(f1), f1=0; end;
   global xc; if isempty(xc), xc=0; end;
   global yc; if isempty(yc), yc=0; end;
   global psi; if isempty(psi), psi=zeros(1,1000); end;
   global dpsi; if isempty(dpsi), dpsi=zeros(1,1000); end;
   global alpha; if isempty(alpha), alpha=zeros(1,500); end;
   global beta; if isempty(beta), beta=zeros(1,500); end;
   global alphan; if isempty(alphan), alphan=zeros(1,1000); end;
   global betan; if isempty(betan), betan=zeros(1,1000); end;
   global psiseg; if isempty(psiseg), psiseg=zeros(1,500); end;
   global dpsiseg; if isempty(dpsiseg), dpsiseg=zeros(1,500); end;
   global alphaseg; if isempty(alphaseg), alphaseg=zeros(1,500); end;
   global betaseg; if isempty(betaseg), betaseg=zeros(1,500); end;
   global store; if isempty(store), store=zeros(1,500); end;
   global psipi; if isempty(psipi), psipi=zeros(1,1000); end;
   global dpsipi; if isempty(dpsipi), dpsipi=zeros(1,1000); end;
```

```
global psit; if isempty(psit), psit=zeros(1,1000); end;
global dpsit; if isempty(dpsit), dpsit=zeros(1,1000); end;
global flowseg; if isempty(flowseg), flowseg=zeros(1,500); end;
global angstore; if isempty(angstore), angstore=zeros(1,510); end;
global a; if isempty(a), a=zeros(1000,1001); end;
global arow; if isempty(arow), arow=zeros(500,3); end;
global brow; if isempty(brow), brow=zeros(500,3); end;
global maxl; if isempty(maxl), maxl=0; end;
global zg; if isempty(zg), zg=zeros(1,8); end;
global wg; if isempty(wg), wg=zeros(1,8); end;
global egl; if isempty(egl), egl=zeros(1,8); end;
global wgl; if isempty(wgl), wgl=zeros(1,8); end;
global jacob; if isempty(jacob), jacob=0; end;
global ungx; if isempty(ungx), ungx=0; end;
global ungy; if isempty(ungy), ungy=0; end;
global sf; if isempty(sf), sf=zeros(3,8); end;
global sd; if isempty(sd), sd=zeros(3,13); end;
global sfl; if isempty(sfl), sfl=zeros(4,3,8); end;
global sdl; if isempty(sdl), sdl=zeros(4,3,8); end;
global nel; if isempty(nel), nel=0; end;
global nnp; if isempty(nnp), nnp=0; end;
global maxnel; if isempty(maxnel), maxnel=0; end;
global maxnnp; if isempty(maxnnp), maxnnp=0; end;
global maxnb; if isempty(maxnb), maxnb=0; end;
global neend; if isempty(neend), neend=0; end;
global ngauss; if isempty(ngauss), ngauss=0; end;
global node; if isempty(node), node=zeros(500,3); end;
global m1; if isempty(m1), m1=zeros(1,500); end;
global m3; if isempty(m3), m3=zeros(1,500); end;
global nbound; if isempty(nbound), nbound=0; end;
global nsegtot; if isempty(nsegtot), nsegtot=0; end;
global nelb; if isempty(nelb), nelb=zeros(1,10); end;
global nsegb; if isempty(nsegb), nsegb=zeros(1,10); end;
global nbcp; if isempty(nbcp), nbcp=0; end;
global nbcd; if isempty(nbcd), nbcd=0; end;
global nbcm; if isempty(nbcm), nbcm=0; end;
global nbct; if isempty(nbct), nbct=0; end;
global ibce; if isempty(ibce), ibce=zeros(1,500); end;
global ibcn; if isempty(ibcn), ibcn=zeros(1,1000); end;
global isegbc; if isempty(isegbc), isegbc=zeros(1,500); end;
global isegend; if isempty(isegend), isegend=zeros(1,500); end;
```

```
 global isegelem; if isempty(isegelem), isegelem=zeros(1,500); end;
 global mfirst; if isempty(mfirst), mfirst=zeros(1,500); end;
 global mlast; if isempty(mlast), mlast=zeros(1,500); end;
```

Finally, the alternative form of function "internal", which is required to carry out area integration over a rectangular solution domain (described in Section 4.3), is as follows.

```
function internal(varargin)
%
%  SUBPROGRAM TO FIND AND OUTPUT THE INTEGRAL OF POTENTIAL OVER THE
%  AREA OF THE SOLUTION DOMAIN.
%  THIS VERSION IS DESIGNED FOR A RECTANGULAR DOMAIN.
%
global nbcp isegbc psiseg xsend ysend maxl psi dpsi xc yc f1 nel ...
    node ibce store dpsipim alpha beta ngauss sf wg jacob fid5 fid6
global psival; if isempty(psival), psival=zeros(1,300); end;
global yintgl; if isempty(yintgl), yintgl=zeros(1,300); end;
%
%  INPUT THE NUMBERS OF SAMPLING POINTS INSIDE THE DOMAIN IN THE X
%  AND Y DIRECTIONS, WHICH SHOULD BOTH BE ODD.
line=fgetl(fid5);
vec=str2num(line);
nx=vec(1); ny=vec(2);
if(rem(nx,2) == 0)
  fprintf(fid6,['\n','WARNING - NX =','%4i', ...
      ' NOT ODD - INCREASE BY 1','\n'],nx);
  nx=nx+1;
end;
if(rem(ny,2) == 0)
  fprintf(fid6,['\n','WARNING - NY =','%4i', ...
      ' NOT ODD - INCREASE BY 1','\n'],ny);
  ny=ny+1;
end;
if(nx+2 > 300 || ny+2 > 300)
  fprintf(fid6,['\n','NX =','%6i','     ','NY =','%6i','     ', ...
      'ARRAY DIMENSIONS EXCEEDED',' IN INTEGRATION OVER DOMAIN', ...
      '\n'],nx,ny);
  error('array dimension error in internal');
end;
```

```
%
%  FIND PRESCRIBED BOUNDARY VALUES OF POTENTIAL.
for ibcp=1:nbcp; % each segment in turn
  if(isegbc(ibcp) ==  1)
    psibot=psiseg(ibcp);
  end;
  if(isegbc(ibcp) ==  2)
    psiright=psiseg(ibcp);
  end;
  if(isegbc(ibcp) ==  3)
    psitop=psiseg(ibcp);
  end;
  if(isegbc(ibcp) ==  4)
    psileft=psiseg(ibcp);
  end;
end % each segment in turn
%
%  FIND COORDINATES OF DOMAIN SIDES.
ybot=ysend(1);
xright=xsend(2);
ytop=ysend(3);
xleft=xsend(4);
%
%  OUTER LOOP FOR INTEGRATION IN X DIRECTION.
hx=(xright-xleft)/(nx+1);
hy=(ytop-ybot)/(ny+1);
ixmax=nx+2;
iymax=ny+2;
yintgl(1)=psileft*(ytop-ybot);
yintgl(ixmax)=psiright*(ytop-ybot);
for ix=2:ixmax-1; % each x position in turn
  psival(1)=psibot;
  psival(iymax)=psitop;
  xpoint=xleft+hx*(ix-1);
%
%  INNER LOOP FOR INTEGRATION IN Y DIRECTION.
%
%  FIRST FIND POTENTIALS AT THE INTERNAL POINTS.
  for iy=2:iymax-1; % each y position in turn
```

```
      ypoint=ybot+hy*(iy-1);
      xp=xpoint/maxl;
      yp=ypoint/maxl;
%
%  INTEGRATE ROUND THE BOUNDARY.
      sum=0.;
      for m=1:nel; % each element in turn
        for in=1:3; % each element node in turn
          j=node(m,in);
%
%  FIND THE POTENTIAL GRADIENT AT THE NODE, EITHER AS COMPUTED
%  IF POTENTIAL WAS PRESCRIBED, OR FROM THE PRESCRIBED GRADIENT
%  OR MIXED BOUNDARY CONDITIONS.
          if(ibce(m) == 1)
            dpsistore=dpsi(j);
          end;
          if(ibce(m) == 2)
            dpsistore=store(m)*maxl-dpsipim(m,in);
          end;
          if(ibce(m) == 3)
            dpsistore=(alpha(m)*psit(j)+beta(m))*maxl-dpsipim(m,in);
          end;
%
%  APPLY GAUSSIAN QUADRATURE.
          for igauss=1:ngauss; % each gauss point in turn
            it=1;
            ic=1;
            jacobi(m,igauss,it,ic);
            [ak,bk]=kernel(xp,yp,m,igauss);
            sfn=sf(in,igauss);
            sum=sum+wg(igauss)*sfn*jacob*(-ak*psi(j)+bk*dpsistore);
          end % each gauss point in turn
        end % each element node in turn
      end % each element in turn
      psiip=sum*0.5/pi;
      psiipt=psiip;
%
%  ADD THE PARTICULAR INTEGRAL FOR A POISSON TYPE PROBLEM.
      if(f1 ~= 0.)
```

```
        xx=xp-xc;
        yy=yp-yc;
        psiipt=psiipt+0.25*f1*(xx^2+yy^2);
      end;
      psival(iy)=psiipt;
    end; % each y position in turn
  %
  %  INTEGRATE POTENTIAL IN Y DIRECTION.
    sum=0.;
    for iy=2:2:iymax-1; % alternate points in the y direction
      sum=sum+psival(iy-1)+4.*psival(iy)+psival(iy+1);
    end; % alternate points in the y direction
    yintgl(ix)=sum*hy/3.;
  end; % each x position in turn
  %
  %  NOW INTEGRATE IN X DIRECTION.
  sum=0.;
  for ix=2:2:ixmax-1; % alternate points in the x direction
    sum=sum+yintgl(ix-1)+4.*yintgl(ix)+yintgl(ix+1);
  end; % alternate points in the x direction
  psiintg=sum*hx/3.;
  area=(xright-xleft)*(ytop-ybot);
  apsiintg=psiintg/area;
  %
  %  OUTPUT RESULT.
  fprintf(fid6,['\n','INTEGRAL OF POTENTIAL OVER SOLUTION DOMAIN =', ...
      '%15.6e','\n','AREA AVERAGE OVER SOLUTION DOMAIN =','%15.6e', ...
      '\n','NUMBERS OF INTERNAL POINTS IN X AND Y DIRECTIONS =', ...
      '%5i','     ','AND','%5i','\n'],psiintg,apsiintg,nx,ny);
  %
  return;
  end %function internal
```

Solutions to Problems – Part 1

Numerical answers are given, followed where appropriate by detailed notes on the solution procedure and comments on the results obtained.

Chapter 1

1.1 Considering $0 \le x \le W/2$: $w = 0$ on $x = 0$ and $y = 0, w = V_z$ on $y = H$. Symmetry condition $\partial w/\partial x = 0$ on $x = W/2$.

1.2 A membrane such as a rubber sheet or soap film can be stretched over a frame whose shape follows that of the cross section of the twisted bar. Under pressure it deflects according to the given equation, which is analogous to Equation 1.41. Torque can be found from the measured volume of membrane displacement, and shear stresses from the slopes of the membrane.

1.3 Velocities defined by Equation 1.58 satisfy continuity Equation 1.46 to give:

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = 0$$

Porous medium flow under a dam can be analysed by choosing a region of rock extending some distance to either side of the dam and some distance below the dam. Boundary conditions for this domain are hydrostatic water pressure on the upper surface on one side of the dam, zero on the upper surface on the other side, and zero normal derivative of pressure (no flow into the boundary) on all other parts of the boundary, including the base of the dam. Some trial and error is required to find the extent of the domain in both the vertical and horizontal directions to give results which do not depend on the domain size.

1.4 (a) Plane stress. (b) Plane strain. (c) Plane strain. (d) Plane strain. (e) Depends on thickness of specimen: if thin then plane stress, if thicker then plane stress near the surfaces, plane strain in the middle.

1.5 (a) Simple uniform direct stress: $\sigma_{yy} = 2A$. (b) Pure bending: $\sigma_{xx} = 6By \equiv yM/I$, where M is bending moment and is second moment of area of a beam cross section.

1.6 Under plane strain with $e_{zz} = 0$

$$\sigma_{xx} = \frac{E}{(1+\nu)(1-2\nu)}\left[(1-\nu)e_{xx} + \nu e_{yy}\right]$$

$$\sigma_{xx} = \frac{E}{(1+\nu)(1-2\nu)}\left[(1-\nu)e_{xx} + \nu e_{yy}\right]$$

If Poisson's ratio is ½ then the material is incompressible: from Equations 1.20 to 1.22 the volumetric strain $(e_{xx} + e_{yy} + e_{zz})$ is zero, irrespective of the magnitude of the direct stresses. In the above expressions, the $(1-2\nu)$ denominator factor means that stresses cannot be uniquely defined by specifying the strains.

Chapter 2

2.2 $a_1 = k\,,\quad a_2 = h\,,\quad a_3 = -hT_\infty.$

2.3 $N_1(\xi) = ½(1-\xi),\quad N_2(\xi) = ½(1+\xi)$

2.5 If a quadratic element is straight, $x_2 = ½(x_1 + x_3)$, which leads to

$$x(\xi) = ½(1-\xi)x_1 + ½(1+\xi)x_3$$

which is the linear form (see the solution to Problem 2.3 above). Similarly for $y(\xi)$.

2.6 $J(\xi) = ½\sqrt{(x_3 - x_1)^2 + (y_3 - y_1)^2}$ = constant (half the length of the element).

2.7 (a) 0.81% (b) 0.05%.

2.9 $\psi_{PI} = \frac{Ex^3}{6} + \frac{Fy^3}{6} + \frac{G}{4}(x^2 + y^2)$ is one possibility. At any point on the boundary at which potential is prescribed, the value of this expression must be subtracted from the prescribed potential. At any point on the boundary at which potential gradient is prescribed, from this prescribed value must be subtracted the value of

$$\frac{\mathrm{d}\psi_{PI}}{\mathrm{d}n} = \frac{\partial\psi_{PI}}{\partial x}n_x + \frac{\partial\psi_{PI}}{\partial y}n_y = \frac{f_1}{2}\left(xn_x + yn_y\right)$$

where, $\frac{\partial\psi_{PI}}{\partial x} = \frac{Ex^2}{2} + \frac{Gx}{2}\,, \qquad \frac{\partial\psi_{PI}}{\partial y} = \frac{Fy^2}{2} + \frac{Gy}{2}$

Chapter 3

3.1 Answer: 17.8 m/s at the centre of the pipe. About 240 nodes/elements around the circumference are required to get three significant figure accuracy. The problem is Poisson type, with

$$f_1 = -\frac{P_z}{\mu} = -\frac{120\times10^3}{9.5} = -12631.6 \quad 1/\text{ms}$$

The analytical solution (D is pipe diameter) for maximum velocity is

$$\frac{P_z D^2}{16\mu} = 17.76\ \text{m/s}.$$

Using symmetry to analyse a 30° slice, the same answer is obtained, but about 120 nodes/elements on each of the three boundary segments (the 30° arc and the two radii) are needed for three significant figure accuracy, and the maximum velocity is found not at the centre of the pipe, but at the nodes closest to it – fortunately the velocity profile is quite flat there so the error in the location is not significant. There is no benefit to be gained by using symmetry in this case.

3.2 Answer: 28.4 m/s at the centre of the pipe. Rather than create the mesh data for an elliptical boundary, the same data file used for Problem 3.1 can be employed, but in subprogram MESHC the ellipse can be generated by scaling all XNODE and XEEND values by a factor of 2 immediately before finding and storing the components of the unit outward normals. Convergence to three significant figure accuracy occurs at about 240 nodes/elements around the boundary.

3.3 Answer: 9.05 m/s at the centroid of the cross section (on the vertical centre line through the vertex of the triangle, and 50 mm above the base). Three significant accuracy is achieved with about 60 nodes/element per side, 180 in total.

3.4 Answer: 2.95×10^4 Nm^2/rad. The problem is Poisson type, with

$$f_1 = -2G\theta = -2\times79.6\times10^9\times1 = -159.2\times10^9$$

Three significant accuracy is achieved with about 60 nodes/elements per side of the domain, and 61 internal points in each direction for the domain integration.

3.5 Answers: (a) 41.9°C, 10.0 kW/m (b) 35.0°C, 10.0 kW/m. The problem is Poisson type, with

$$f_1 = -\frac{g}{k} = -\frac{16\times10^6}{295} = -54237.3 \quad ^\circ\text{C/m}^2$$

In (a) the problem is one-dimensional and can be solved analytically to give: maximum $T = \frac{gL^2}{2k} + 25°\text{C} = \frac{16\times 10^6 \times 0.025^2}{2\times 295} + 25°\text{C} = 41.95°\text{C}$ (at the surface opposite the cooled one, length of conductor side L). In (b) the maximum temperature is detected at the nodes adjacent to the corner between the two insulated surfaces. In both cases three significant figure accuracy is achieved at about 60 nodes/elements per side of the domain. The heat flows of 10.0 kW/m are to be expected because the uniform rate of heat generation is 16.0 MW/m^3, over an area of $0.025 \times 0.025 = 625 \times 10^{-6}\text{m}^2$, and $16.0 \times 10^3 \times 625 \times 10^{-6} = 10.0\ \text{kW/m}$.

3.6 Answers: 27.1°C (at the centre of the conductor), 7.85 kW/m. The analytical solution gives $T = \frac{gR^2}{4k} + 25°\text{C} = 27.12°\text{C}$ (at the centre of the conductor, radius R). The uniform rate of heat generation multiplied by the cross sectional area of the conductor is 7.854 kW/m. Three significant figure accuracy is achieved at about 120 nodes/elements around the circumference.

3.7 Answers: (a) 55.7 (at the outer surface), 159 W/m (b) 101°C (at the outer surface), 159 W/m. Three significant figure accuracy is achieved at about 120 nodes/elements per boundary.

3.8 Answers: 42.8°C (at the nodes adjacent to the origin), 4.89 kW/m (hole centred at (30, 40)), 3.14 kW/m (hole centred at (80, 20)), and 2.91 kW/m (hole centred at (70, 60)). Three significant figure accuracy is achieved for the heat flow rates at about 90 nodes/elements on each of the straight edges, and 180 around each hole. The maximum temperature may not have converged to the same accuracy, but at 900 nodes the maximum problem size permitted by the program has almost been reached. The reason for this lack of convergence is that the nodes where the maximum temperature is detected are not exactly at the origin.

3.9 Answer: 50.5 mm at (0.52, 0.45) approximately. The problem is Poisson type, with

$$f_1 = -\frac{p}{S} = -\frac{50}{72} = -0.694444\ \ 1/\text{m}$$

Three significant figure accuracy is achieved at about 60 nodes/elements on each side of the domain. Trial and error is used to find the position of the internal point at which the deflection is a maximum (it is a relatively flat maximum).

3.10 The only dimension which gives any length scale to the problem is the base thickness of the dam, 10 m. The depth of water is only relevant in defining the pressure on the rock on the wet side of the dam: density $\times\, g \times$ depth $= 1000 \times 9.81 \times 15 = 147150$ N/m^2. A possible, but somewhat arbitrary, domain to choose is a rectangular one that extends 1000 m horizontally on the wet side of the dam, 1000 m horizontally on the dry side, and goes to a vertical depth of 1000 m, each of these dimensions being a hundred times the thickness of the dam. This gives a total of 5 boundary segments, including one for the base of the dam, and the overall domain size is 2010 m horizontally and 1000 m vertically.

The boundary conditions are a pressure of 147150 on the top surface of the domain on the wet side of the dam, zero on the dry side, and zero normal derivative (of pressure) on all other parts of the boundary. The program computes the potential flow rates (integrated pressure gradients) into the domain (wet side) and out of the domain (dry side). Using 60 elements of uniform size on each of the segments, this model of the problem gives potential flows in and out of about 260000 N/m. Multiplying by the permeability of 2×10^{-10} m^4/Ns gives the volumetric flow rate of water as 5.2×10^{-5} m^3/s per meter length of dam (normal to its cross section). To improve accuracy it is necessary to refine the sizes of the elements modelling the rock surface close to the base of the dam, where using elements of uniform size gives very large changes in sizes of elements. Applying an element size ratio of 1.01 to elements moving away from the dam on the two segments representing the rock surface gives a potential flow result of about 290000 N/m, and a volumetric flow rate of water of 5.8×10^{-5} m^3/s per meter length of dam (normal to its cross section).

Chapter 4

4.1 Answer: 17.8 m/s at the centre of the pipe. Three significant figure accuracy is obtained with one element per quadrant (4 elements with 8 nodes in total). The problem is Poisson type, with

$$f_1 = -\frac{P_z}{\mu} = -\frac{120 \times 10^3}{9.5} = -12631.6 \quad 1/\text{ms}$$

The analytical solution (D is pipe diameter) for maximum velocity is

$$\frac{P_z D^2}{16\mu} = 17.76 \text{ m/s}.$$

Using symmetry to analyse a 30° slice, the same answer is obtained, with one element on each of the three edges of the slice (6 nodes in total). There is little benefit to be gained by using symmetry in this case. The reason that such good accuracy is obtained with only one element on each of the straight edges of the slice is that the velocity profile is parabolic (quadratic), which can be represented exactly by the element shape functions. For comparison, over 200 nodes are required for either version of the problem when using constant boundary elements (see Problem 3.1).

4.2 Answer: 28.4 m/s at the centre of the pipe. Rather than create the mesh data for an elliptical boundary, the same data file used for Problem 4.1 can be employed, but in subprogram MESHC the ellipse can be generated by scaling all XNODE values by a factor of 2 immediately before finding the maximum dimension of the domain. Convergence to three significant figure accuracy occurs at about 16 elements (32 nodes) around the boundary. This compares with 240 nodes to obtain similar accuracy using constant boundary elements (Problem 3.2).

4.3 Answer: 9.05 m/s at the centroid of the cross section (on the vertical centre line through the vertex of the triangle, and 50 mm above the base). Three significant accuracy is achieved with about 4 elements per side, 24 nodes in total. This compares with 180 nodes to obtain similar accuracy using constant boundary elements (Problem 3.3).

4.4 Answer: 2.95×10^4 Nm²/rad. The problem is Poisson type, with

$$f_1 = -2G\theta = -2 \times 79.6 \times 10^9 \times 1 = -159.2 \times 10^9$$

Three significant accuracy is achieved with about 4 elements (9 nodes) per side of the domain, and 9 internal points in each direction for the domain integration. This compares with about 60 nodes per side, and 61 internal points in each direction using constant boundary elements (Problem 3.4). In section 4.3 an analogous channel flow problem was considered where using quadratic rather than constant boundary elements offered no significant benefits, which contrasts with the present problem. The reason for the difference is that here the aspect ratio of the domain of 2 is much less than the figure of 5 in the channel flow problem: quadratic elements are less good at analysing slender domains.

4.5 Answers: (a) 41.9°C, 10.0 kW/m (b) 35.0°C, 10.0 kW/m. The problem is Poisson type, with

$$f_1 = -\frac{g}{k} = -\frac{16 \times 10^6}{295} = -54237.3 \ \text{°C/m}^2$$

In (a) the problem is one-dimensional and can be solved analytically to give: maximum $T = \frac{gL^2}{2k} + 25°\text{C} = \frac{16\times10^6\times0.025^2}{2\times295} + 25°\text{C} = 41.95°\text{C}$ (at the surface opposite the cooled one, length of conductor side L). In (b) the maximum temperature occurs the corner between the two insulated surfaces. In both cases (a) and (b) three significant figure accuracy is achieved with one element per side of the domain, 8 nodes in total. This compares with 240 nodes to obtain similar accuracy using constant boundary elements (Problem 3.5). As in Problem 3.5 the heat flows of exactly 10.0 kW/m are to be expected because of the known rate of heat generation. In case (b) the maximum temperature occurs at a corner point where there is a quadratic element node, but not a constant element node. Although this means that the maximum from the constant element analysis is inherently less accurate, the error in this case is undetectable at three significant figure accuracy, and indeed was removed by refining the mesh.

4.6 Answers: 27.1°C (at the centre of the conductor), 7.85 kW/m. The analytical solution gives $T = \frac{gR^2}{4k} + 25°\text{C} = 27.12°\text{C}$ (at the centre of the conductor, radius *R*). The uniform rate of heat generation multiplied by the cross sectional area of the conductor is 7.854 kW/m. Three significant figure accuracy is achieved at 2 elements per quadrant, 16 nodes in total. This compares with 120 nodes to obtain similar accuracy using constant boundary elements (Problem 3.6).

4.7 Answers: (a) 55.7°C (at the outer surface), 159 W/m (b) 101°C (at the outer surface), 159 W/m. Three significant figure accuracy is achieved at 2 elements per quadrant, 16 nodes per boundary. This compares with 120 nodes per boundary to obtain similar accuracy using constant boundary elements (Problem 3.7).

4.8 Answers: 43.0°C (at the nodes adjacent to the origin), 4.89 kW/m (hole centred at (30, 40)), 3.14 kW/m (hole centred at (80, 20)), and 2.91 kW/m (hole centred at (70, 60)). Three significant figure accuracy is achieved with 2 elements per quadrant on the holes and 4 elements on each of the straight domain edges, 80 nodes in total. This compares with the 900 nodes used in the constant boundary element analysis (Problem 3.8) where the maximum temperature had not converged (42.8°C compared with 43.0°C) because there was no node at the point of maximum temperature.

4.9 Answer: 50.5 mm at (0.52, 0.45) approximately. The problem is Poisson type, with

$$f_1 = -\frac{p}{S} = -\frac{50}{72} = -0.694444 \text{ 1/m}$$

Three significant figure accuracy is achieved at 2 element on each side of the domain, 16 nodes in total. This compares with about 240 nodes to obtain similar accuracy using constant boundary elements (Problem 3.9). Trial and error is used to find the position of the internal point at which the deflection is a maximum (it is a relatively flat maximum).

4.10 The only dimension which gives any length scale to the problem is the base thickness of the dam, 10 m. The depth of water is only relevant in defining the pressure on the rock on the wet side of the dam: density $\times\, g \times$ depth $= 1000 \times 9.81 \times 15 = 147150$ N/m^2. A possible, but somewhat arbitrary, domain to choose is a rectangular one that extends 1000 m horizontally on the wet side of the dam, 1000 m horizontally on the dry side, and goes to a vertical depth of 1000 m, each of these dimensions being a hundred times the thickness of the dam. This gives a total of 5 boundary segments, including one for the base of the dam, and the overall domain size is 2010 m horizontally and 1000 m vertically.

The boundary conditions are a pressure of 147150 on the top surface of the domain on the wet side of the dam, zero on the dry side, and zero normal derivative (of pressure) on all other parts of the boundary. The program computes the potential flow rates (integrated pressure gradients) into the domain (wet side) and out of the domain (dry side). Using 60 elements of uniform size on each of the segments, this model of the problem gives potential flows in and out of about 305000 N/m. Multiplying by the permeability of 2×10^{-10} m^4/Ns gives the volumetric flow rate of water as 6.1×10^{-5} m^3/s per meter length of dam (normal to its cross section). To improve accuracy it is necessary to refine the sizes of the elements modelling the rock surface close to the base of the dam, where using elements of uniform size gives very large changes in sizes of elements. Applying an element size ratio of 1.1 to elements moving away from the dam on the two segments representing the rock surface gives a potential flow result of about 292000 N/m, and a volumetric flow rate of water of 5.8×10^{-5} m^3/s per meter length of dam (normal to its cross section).

4.11 Answers: (a) 120%, 266°C (b) 218%, 209°C (c) 270%, 153°C. In the absence of a fin, the potential (temperature) flow rate to atmosphere is 8.152, from either the analytical solution or a boundary element analysis. The domain boundary can be divided into 10 segments: 7 straight, 2 quadrants and 1 semicircle. Using 8 elements per segment is more than sufficient to give convergence to three significant figures. The three fin lengths give potential flow rates to atmosphere of 17.92, 25.90 and 30.15.

4.12 Answers: 109 W/m, 21.4°C. With a number of identical pipes running parallel to each other, there will be planes of symmetry half way between successive pipes, and a domain of width 2 m with a pipe at the middle is appropriate for analysis. Tests show that a domain depth of 10 m with a zero temperature gradient boundary condition over the bottom edge is a reasonable choice (further increase in the depth has no significant effect on the answers). Using 8 elements per straight edge of the domain, and 16 elements around the pipe circumference gives three significant figure accuracy.

To see Part 2 download
Boundary Element Methods for Engineers – Part 2: Plane Elastic Problems