NAG Toolbox

nag_pde_2d_laplace (d03ea)

1 Purpose

nag_pde_2d_laplace (d03ea) solves Laplace's equation in two dimensions for an arbitrary domain bounded internally or externally by one or more closed contours, given the value of either the unknown function or its normal derivative (into the domain) at each point of the boundary.

2 Syntax

```
[phi, phid, alpha, ifail] = nag_pde_2d_laplace(stage1, ext, dorm, p, q, x, y,
phi, phid, alpha, 'n', n, 'n1p1', n1p1)
```

```
[phi, phid, alpha, ifail] = d03ea(stage1, ext, dorm, p, q, x, y, phi, phid,
alpha, 'n', n, 'n1p1', n1p1)
```

3 Description

nag_pde_2d_laplace (d03ea) uses an integral equation method, based upon Green's formula, which yields the solution, ϕ, within the domain, given its value or that of its normal derivative at each point of the boundary (except possibly at a finite number of discrete points). The solution is obtained in two stages. The first, which is executed once only, determines the complementary boundary values, i.e., ϕ, where its normal derivative is known and vice versa. The second stage is entered once for each point at which the solution is required.

The boundary is divided into a number of intervals in each of which ϕ and its normal derivative are approximated by constants. Of these half are evaluated by applying the given boundary conditions at one 'nodal' point within each interval while the remainder are determined (in stage 1) by solving a set of simultaneous linear equations. Here this is achieved by means of auxiliary function nag_lapack_dgels (f08aa), which will yield the least squares solution of an overdetermined system of equations as well as the unique solution of a square nonsingular system.

In exterior domains the solution behaves as $c + s\log(r) + O(1/r)$ as r tends to infinity, where c is a constant, s is the total integral of the normal derivative around the boundary and r is the radial distance from the origin of coordinates. For the Neumann problem (when the normal derivative is given along the whole boundary) s is fixed by the boundary conditions whilst c is chosen by you. However, for a Dirichlet problem (when ϕ is given along the whole boundary) or for a mixed problem, stage 1 produces a value of c for which $s = 0$; then as r tends to infinity the solution tends to the constant c.

4 References

Symm G T and Pitfield R A (1974) Solution of Laplace's equation in two dimensions *NPL Report NAC 44* National Physical Laboratory

5 Parameters

5.1 Compulsory Input Parameters

1: **stage1** – LOGICAL

Indicates whether the function call is for stage 1 of the computation as defined in Section 3.

stage1 $=$ *true*
The call is for stage 1.

stage1 $=$ *false*
The call is for stage 2.

2: **ext** – LOGICAL

The form of the domain. If **ext** $= true$, the domain is unbounded. Otherwise the domain is an interior one.

3: **dorm** – LOGICAL

The form of the boundary conditions. If **dorm** $= true$, then the problem is a Dirichlet or mixed boundary value problem. Otherwise it is a Neumann problem.

4: **p** – REAL (KIND=nag_wp)
5: **q** – REAL (KIND=nag_wp)

To stage 2, **p** and **q** must specify the x and y coordinates respectively of a point at which the solution is required.

When **stage1** is *true*, **p** and **q** are ignored.

6: **x**(**n1p1**) – REAL (KIND=nag_wp) array
7: **y**(**n1p1**) – REAL (KIND=nag_wp) array

The x and y coordinates respectively of points on the one or more closed contours which define the domain of the problem.

Note: each contour is described in such a manner that the subscripts of the coordinates increase when the domain is kept on the left. The final point on each contour coincides with the first and, if a further contour is to be described, the coordinates of this point are repeated in the arrays. In this way each interval is defined by three points, the second of which (the nodal point) always has an even subscript. In the case of the interior Neumann problem, the outermost boundary contour must be given first, if there is more than one.

8: **phi**(**n**) – REAL (KIND=nag_wp) array

For stage 1, **phi** must contain the nodal values of ϕ or its normal derivative (into the domain) as prescribed in each interval. For stage 2 it must retain its output values from stage 1.

9: **phid**(**n**) – REAL (KIND=nag_wp) array

For stage 1, **phid**(i) must hold the value 0.0 or 1.0 accordingly as **phi**(i) contains a value of ϕ or its normal derivative, for $i = 1, 2, \ldots,$ **n**. For stage 2 it must retain its output values from stage 1.

10: **alpha** – REAL (KIND=nag_wp)

For stage 1, the use of **alpha** depends on the nature of the problem:

if **dorm** $= true$, **alpha** need not be set;

if **dorm** $= false$ and **ext** $= true$, **alpha** must contain the prescribed constant c (see Section 3);

if **dorm** $= false$ and **ext** $= false$, **alpha** must contain an appropriate value (often zero) for the integral of ϕ around the outermost boundary.

For stage 2, on every call **alpha** must contain the value returned at stage 1.

5.2 Optional Input Parameters

1: **n** – INTEGER

Default: the dimension of the arrays **phi**, **phid**. (An error is raised if these dimensions are not equal.)

The number of intervals into which the boundary is divided (see Section 7 and Section 9).

2: **n1p1** – INTEGER

Default: the dimension of the arrays **x**, **y**. (An error is raised if these dimensions are not equal.)

The value $2(\mathbf{n} + M) - 1$, where M denotes the number of closed contours which make up the boundary.

5.3 Output Parameters

1: **phi**(**n**) – REAL (KIND=nag_wp) array

From stage 1, it contains the constants which approximate ϕ in each interval. It remains unchanged on exit from stage 2.

2: **phid**(**n**) – REAL (KIND=nag_wp) array

From stage 1, **phid** contains the constants which approximate the normal derivative of ϕ in each interval. It remains unchanged on exit from stage 2.

3: **alpha** – REAL (KIND=nag_wp)

From stage 1:

if **ext** $= \mathit{false}$, **alpha** contains 0.0;

if **ext** $= \mathit{true}$ and **dorm** $= \mathit{false}$ **alpha** is unchanged;

if **ext** $= \mathit{true}$ and **dorm** $= \mathit{true}$ **alpha** contains a computed estimate for c.

From stage 2:

alpha contains the computed value of ϕ at the point (**p**,**q**).

4: **ifail** – INTEGER

ifail $= 0$ unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail $= 1$

Invalid tolerance used in an internal call to an auxiliary function:

$icint(1) = 0$
Indicates too large a tolerance.

$icint(1) > 0$
Indicates too small a tolerance.

Note: this error is only possible in stage 1, and the circumstances under which it may occur cannot be foreseen. In the event of a failure, it is suggested that you change the scale of the domain of the problem and apply the function again.

ifail $= 2$

Incorrect rank obtained by an auxiliary function; $icint(1)$ contains the computed rank.

ifail $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

ifail $= -399$

Your licence key may have expired or may not have been installed correctly.

ifail $= -999$

Dynamic memory allocation failed.

7 Accuracy

The accuracy of the computed solution depends upon how closely ϕ and its normal derivative may be approximated by constants in each interval of the boundary and upon how well the boundary contours are represented by polygons with vertices at the selected points $(\mathbf{x}(i), \mathbf{y}(i))$, for $i = 1, 2, \ldots, 2(\mathbf{n} + \mathrm{M}) - 1$.

Consequently, in general, the accuracy increases as the boundary is subdivided into smaller and smaller intervals and by comparing solutions for successive subdivisions one may obtain an indication of the error in these solutions.

Alternatively, since the point of maximum error always lies on the boundary of the domain, an estimate of the error may be obtained by computing ϕ at a sufficient number of points on the boundary where the true solution is known. The latter method (not applicable to the Neumann problem) is most useful in the case where ϕ alone is prescribed on the boundary (the Dirichlet problem).

8 Further Comments

The time taken for stage 1, which is executed once only, is roughly proportional to $\mathbf{n}^2$, being dominated by the time taken to compute the coefficients. The time for each stage 2 application is proportional to $\mathbf{n}$.

The intervals into which the boundary is divided need not be of equal lengths.

For many practical problems useful results may be obtained with 20 to 40 intervals per boundary contour.

9 Example

An interior Neumann problem to solve Laplace's equation in the domain bounded externally by the triangle with vertices $(3, 0)$, $(-3, 0)$ and $(0, 4)$, and internally by the triangle with vertices $(2, 1)$, $(-2, 1)$ and $(0, 3)$, given that the normal derivative of the solution ϕ is zero on each side of each triangle and, for uniqueness that the total integral of ϕ around the outer triangle is 16 (the length of the contour).

This trivial example has the obvious solution $\phi = 1$ throughout the domain. However it provides a useful illustration of data input for a doubly-connected domain. The solution is computed at one corner of each triangle and at one point inside the domain.

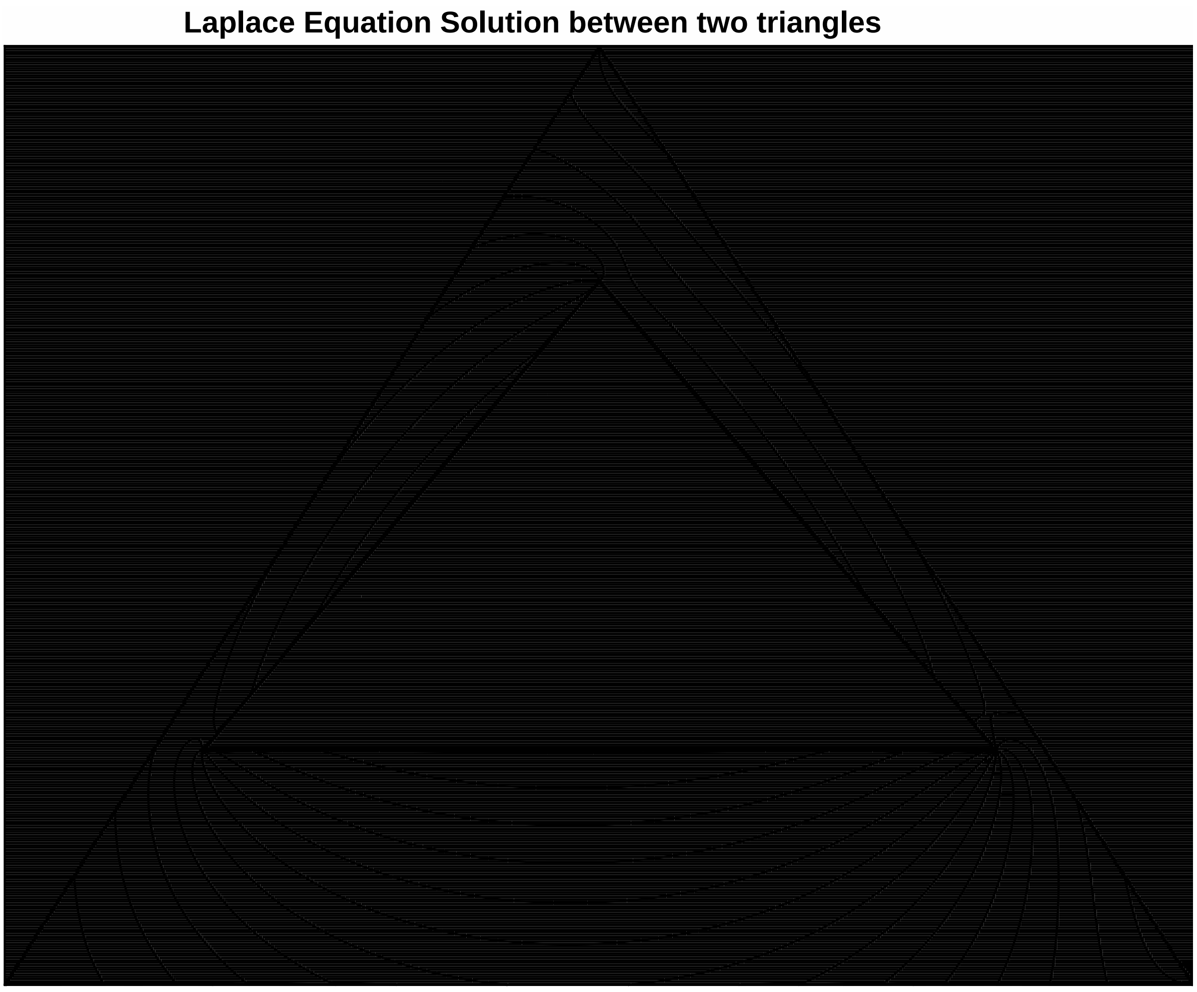
The program is written to handle any of the different types of problem that the function can solve. The array dimensions must be increased for larger problems.

9.1 Program Text

```
     function d03ea_example

fprintf('d03ea example results\n\n');

% This example illustrates using an interior Neumann problem to
% solve the Laplace equation in the domain bounded by two triangles.

% Stage 1: Set up phi and phid
stage1 = true;
ext = false;
dorm = false;
p = 0;
q = 0;
% Outer and inner triangles with mid-points
x = [ 3.0;  1.5;  0.0; -1.5; -3.0;  0.0;  3.0;
      3.0;
      2.0;  0.0; -2.0; -1.0;  0.0;  1.0;  2.0];
y = [ 0.0;  2.0;  4.0;  2.0;  0.0;  0.0;  0.0;
      0.0;
      1.0;  1.0;  1.0;  2.0;  3.0;  2.0;  1.0];
% Some interesting Neumann conditions
phi(1:6) = [3, 4, 4, 5, 4, 3];
phid(1:6) = [0, 0, 0, 0, 0, 0];
% Integral of solution around outer triangle
alpha = 12;
[phi, phid, alpha, ifail] = ...
    d03ea( ...
           stage1, ext, dorm, p, q, x, y, phi, phid, alpha);

% Stage 2 - solution at a set of points
stage1 = false;
phisol = zeros(401,601);
solmin = 1.e10;
solmax = -1.0e10;
i = 0;
for xx = -3:0.01:3
  i = i + 1;
  j = 0;
  xl = 3-abs(xx);
  y1 = (4/3)*xl;
  for yy = 0:0.01:4
    j = j + 1;
    if ( (yy<=y1) && ((yy<=1) || (yy>=xl)));
      % In Outer triangle and outside inner triangle
      [phi, phid, sol, ifail] = ...
      d03ea( ...
             stage1, ext, dorm, xx, yy, x, y, phi, phid, alpha);
      phisol(j,i) = sol;
      solmax = max(solmax,sol);
      solmin = min(solmin,sol);
    end
  end
end

fprintf('The maximum value of solution = %8.3f\n',solmax);
fprintf('The minimum value of solution = %8.3f\n',solmin);

% Plot Solution
fig1 = figure;
contourf(phisol,[solmin:-1,1,2:0.2:solmax]);
title('Laplace Equation Solution between two triangles');
set(gca,'XTick',[],'XTickLabel',[],'YTick',[],'YTickLabel',[]);
```

9.2 Program Results

```
     d03ea example results

The maximum value of solution =    5.430
The minimum value of solution =   -2.979
```


Laplace Equation Solution between two triangles