

NAG Toolbox

nag_pde_2d_gen_order2_rectilinear_extractgrid (d03rz)

1 Purpose

nag_pde_2d_gen_order2_rectilinear_extractgrid (d03rz) is designed to be used in conjunction with nag_pde_2d_gen_order2_rectilinear (d03rb). It can be called from the **monitr** to obtain the number of grid points and their (x, y) coordinates on a solution grid.

2 Syntax

```
[npts, x, y, ifail] = nag_pde_2d_gen_order2_rectilinear_extractgrid(level, nlev,
xmin, ymin, dxb, dyb, lgrid, istruc, lenxy)
[npts, x, y, ifail] = d03rz(level, nlev, xmin, ymin, dxb, dyb, lgrid, istruc,
lenxy)
```

Note: the interface to this routine has changed since earlier releases of the toolbox:

At Mark 25: **nlev** was made optional.

3 Description

nag_pde_2d_gen_order2_rectilinear_extractgrid (d03rz) extracts the number of grid points and their (x, y) coordinates on a specific solution grid produced by nag_pde_2d_gen_order2_rectilinear (d03rb). It must be called only from within the **monitr**. The arguments **nlev**, **xmin**, **ymin**, **dxb**, **dyb**, **lgrid** and **istruc** to **monitr** must be passed unchanged to nag_pde_2d_gen_order2_rectilinear_extractgrid (d03rz).

4 References

None.

5 Parameters

5.1 Compulsory Input Parameters

1: **level** – INTEGER

The grid level at which the coordinates are required.

Constraint: $1 \leq \text{level} \leq \text{nlev}$.

2: **nlev** – INTEGER

3: **xmin** – REAL (KIND=nag_wp)

4: **ymin** – REAL (KIND=nag_wp)

5: **dxb** – REAL (KIND=nag_wp)

6: **dyb** – REAL (KIND=nag_wp)

nlev, **xmin**, **ymin**, **dxb** and **dyb** as supplied to **monitr** must be passed unchanged to nag_pde_2d_gen_order2_rectilinear_extractgrid (d03rz).

7: **lgrid(:)** – INTEGER array

The dimension of the array **lgrid** must be at least **nlev**

lgrid as supplied to **monitr** must be passed unchanged to nag_pde_2d_gen_order2_rectilinear_extractgrid (d03rz).

8: **istruc**(:) – INTEGER array

The dimension of the array **istruc** must be at least **lgrid(nlev)** + 2 × **nrows** + **npts** + 1 where **nrows** is stored in **istruc(lgrid(nlev))** and is the number of rows in the grid at level **nlev**.

istruc as supplied to **monitr** must be passed unchanged to nag_pde_2d_gen_order2_rectilinear_extractgrid (d03rz).

9: **lenxy** – INTEGER

The dimension of the arrays **x** and **y**.

Constraint: **lenxy** ≥ **npts**.

5.2 Optional Input Parameters

None.

5.3 Output Parameters

1: **npts** – INTEGER

The number of grid points in the grid level **level**.

2: **x(lenxy)** – REAL (KIND=nag_wp) array

3: **y(lenxy)** – REAL (KIND=nag_wp) array

x(i) and **y(i)** contain the (x, y) coordinates respectively of the i th grid point, for $i = 1, 2, \dots, \text{npts}$.

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

On entry, **level** < 1,
or **level** > **nlev**.

ifail = 2

The dimension of the arrays **x** and **y** is too small for the requested grid level, i.e., **lenxy** < **npts**.

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

Not applicable.

8 Further Comments

None.

9 Example

See Section 10 in nag_pde_2d_gen_order2_rectilinear (d03rb).

9.1 Program Text

```

function d03rz_example

fprintf('d03rz example results\n\n');

global iout xsol ysol usol vsol inds;

ts      = 0;
twant  = [0.25; 1];
dt      = [0.001; 1e-07; 0];
tol_s  = 0.1;
tol_t  = 0.05;
opti   = nag_int([5;0;0;0]);
optr   = [1, 1; 1, 1; 1, 1];
rwk    = zeros(426000, 1);
iwk    = zeros(117037, 1, nag_int_name);
lenlwk = nag_int(6000);
itrace = nag_int(0);
ind    = nag_int(0);
inds   = nag_int(0);

for iout = 1:2
    tout = twant(iout);
    [ts, dt, rwk, iwk, ind, ifail] = ...
        d03rb(...,
            ts, tout, dt, tol_s, tol_t, @inidom, @pdedef, ...
            @bndary, @pdeiv, @monitr, opti, optr, ...
            rwk, iwk, lenlwk, itrace, ind);
    fprintf('\nStatistics\n');
    fprintf('Time = %8.4f\n', ts);
    fprintf('Total number of accepted timesteps = %d\n', iwk(1));
    fprintf('Total number of rejected timesteps = %d\n', iwk(2));
    fprintf('\n          T o t a l n u m b e r o f      \n');
    fprintf('          Residual   Jacobian   Newton Lin sys\n');
    fprintf('          evals     evals     iters   iters\n');
    fprintf(' At level \n');
    maxlev = opti(1);
    for j = 1:maxlev
        if (iwk(j+2) ~= 0)
            fprintf('%6d %10d %10d %10d %10d\n', j, iwk(j+2), iwk(j+2+maxlev), ...
                iwk(j+2+2*maxlev), iwk(j+2+3*maxlev));
        end
    end
    fprintf('\n          M a x i m u m n u m b e r o f\n');
    fprintf('          Newton iters   Lin sys iters \n');
    fprintf(' At level \n');
    for j = 1:maxlev
        if (iwk(j+2) ~= 0)
            fprintf('%6d%14d%14d\n', j, iwk(j+2+4*maxlev), iwk(j+2+5*maxlev));
        end
    end
end

% Plot solutions u and v at t=1.0
fig1 = figure;
scatter3(xsol,ysol,usol,2,'fill');
title('2D Burgers'' equation at t=1 on disjoint domain: u');
xlabel('x');
ylabel('y');
zlabel('u(x,y;t=1)');

```

```

fig2 = figure;
scatter3(xsol,ysol,vsol,2,'fill');
view(-54,32);
xlabel('x');
ylabel('y');
zlabel('v(x,y;t=1)');
title('2D Burgers'' equation at t=1 on disjoint domain: v');

function [res] = bndary(npts, npde, t, x, y, u, ut, ux, uy, nbnds, ...
    nbpts, llbnd, ilbnd, lbnd, res)

    epsilon = 1e-3;

    for k = llbnd(1):nbpts
        i = lbnd(k);
        a = (-4.0d0*x(i)+4.0d0*y(i)-t)/(32.0d0*epsilon);
        if (a <= 0.0d0)
            res(i,1) = u(i,1) - (0.75d0-0.25d0/(1.0d0+exp(a)));
            res(i,2) = u(i,2) - (0.75d0+0.25d0/(1.0d0+exp(a)));
        else
            res(i,1) = u(i,1) - (0.75d0-0.25d0*exp(-a)/(exp(-a)+1.0d0));
            res(i,2) = u(i,2) - (0.75d0+0.25d0*exp(-a)/(exp(-a)+1.0d0));
        end
    end

function [xmin, xmax, ymin, ymax, nx, ny, npts, nrows, nbnds, nbpts, ...
    lrow, irow, icol, llbnd, ilbnd, ierr] = inidom(maxpts, ierr)

    nrows = nag_int(11);
    npts = nag_int(105);
    nbnds = nag_int(28);
    nbpts = nag_int(72);

    lrow = zeros(nrows, 1, nag_int_name);
    irow = zeros(nrows, 1, nag_int_name);
    icol = zeros(npts, 1, nag_int_name);
    llbnd = zeros(nbnds, 1, nag_int_name);
    ilbnd = zeros(nbnds, 1, nag_int_name);
    lbnd = zeros(nbpts, 1, nag_int_name);

    icold = nag_int([0,1,2,0,1,2,3,4,5,6,7,8,9,10,0,1,2,3,4,5,6,7,8,9, ...
        10,0,1,2,3,4,5,6,7,8,9,10,0,1,2,3,4,5,8,9,10,0,1,2,3,4,5, ...
        6,7,8,9,10,0,1,2,3,4,5,6,7,8,9,10,0,1,2,3,4,5,6,7,8,9,10, ...
        0,1,2,3,4,5,6,7,8,0,1,2,3,4,5,6,7,8,0,1,2,3,4,5,6,7,8]);
    ilbndd = nag_int([1,2,3,4,1,4,1,2,3,4,3,4,1,2,12,23,34,41,14,41,12, ...
        23,34,41,43,14,21,32]);
    irowd = nag_int([0,1,2,3,4,5,6,7,8,9,10]);
    lbndd = nag_int([2,4,15,26,37,46,57,68,79,88,98,99,100,101,102,103,104,96, ...
        86,85,84,83,82,70,59,48,39,28,17,6,8,9,10,11,12,13,18, ...
        29,40,49,60,72,73,74,75,76,77,67,56,45,36,25,33,32,42, ...
        52,53,43,1,97,105,87,81,3,7,71,78,14,31,51,54,34]);
    llbndd = nag_int([1,2,11,18,19,24,31,37,42,48,53,55,56,58,59,60,61,62, ...
        63,64,65,66,67,68,69,70,71,72]);
    lrowd = nag_int([1,4,15,26,37,46,57,68,79,88,97]);
    nx = nag_int(11);
    ny = nag_int(11);

    % check maxpts against rough estimate of npts
    if (maxpts < nx*ny)
        ierr = -1;
        return;
    end

    xmin = 0;

```

```

ymin = 0;
xmax = 1;
ymax = 1;

lrowd(1:nrows) = lrowd(1:nrows);
irowd(1:nrows) = irowd(1:nrows);
llbnd(1:nbnds) = llbndd(1:nbnds);
ilbnd(1:nbnds) = ilbndd(1:nbnds);
lbnnd(1:nbpts) = lbnnd(1:nbpts);
icold(1:npts) = icold(1:npts);

function [res] = pdedef(npts, npde, t, x, y, u, ut, ux, uy, uxx, uxy, uyy)
    res = zeros(npts, npde);

    epsilon = 1e-3;
    uxxyy = epsilon*(uxx(1:npts,1:2)+uyy(1:npts,1:2));
    uuxuy(1:npts,1) = -u(1:npts,1).*ux(1:npts,1)-u(1:npts,2).*uy(1:npts,1);
    uuxuy(1:npts,2) = -u(1:npts,1).*ux(1:npts,2)-u(1:npts,2).*uy(1:npts,2);
    res(1:npts,1:2) = ut(1:npts,1:2)-(uuxuy+uxxy);

function [u] = pdeiv(npts, npde, t, x, y)
    u = zeros(npts, npde);

    epsilon = 1e-3;

    for i = 1:npts
        a = (-4.0d0*x(i)+4.0d0*y(i)-t)/(32.0d0*epsilon);
        if (a <= 0.0d0)
            u(i,1) = 0.75d0 - 0.25d0/(1.0d0+exp(a));
            u(i,2) = 0.75d0 + 0.25d0/(1.0d0+exp(a));
        else
            u(i,1) = 0.75d0 - 0.25d0*exp(-a)/(exp(-a)+1.0d0);
            u(i,2) = 0.75d0 + 0.25d0*exp(-a)/(exp(-a)+1.0d0);
        end
    end

function [ierr] = ...
    monitr(npde, t, dt, dtnew, tlast, nlev, xmin, ymin, dxb, dyb, lgrid, ...
        istruc, lsol, sol, ierr)

lenxy=nag_int(2500);

global iout xsol ysol usol vsol inds;

if (tlast && iout == 2)
    for level = nag_int(1:nlev)
        ipsol = lsol(level);

        % Get grid information
        [npts, x, y, ifail] = ...
            d03rz(..., ...
                level, xmin, ymin, dxb, dyb, lgrid, istruc, lenxy);
        % Get exact solution
        [uex] = pdeiv(npts, npde, t, x, y);

        xsol(inds+1:inds+npts) = x(1:npts);
        ysol(inds+1:inds+npts) = y(1:npts);
        usol(inds+1:inds+npts) = sol(ipsol+1:ipsol+npts);
        vsol(inds+1:inds+npts) = sol(ipsol+npts+1:ipsol+npts+npts);
        inds = inds + npts;
    end
end

```

9.2 Program Results

d03rz example results

Statistics

Time = 0.2500

Total number of accepted timesteps = 14

Total number of rejected timesteps = 0

Total number of				
Residual evals	Jacobian evals	Newton iters	Lin sys iters	
At level				
1	196	14	28	14
2	196	14	28	22
3	196	14	28	25
4	196	14	28	31
5	141	10	21	29

Maximum number of		
Newton iters	Lin sys	iters

At level			
1	2	1	
2	2	1	
3	2	1	
4	2	2	
5	3	2	

Statistics

Time = 1.0000

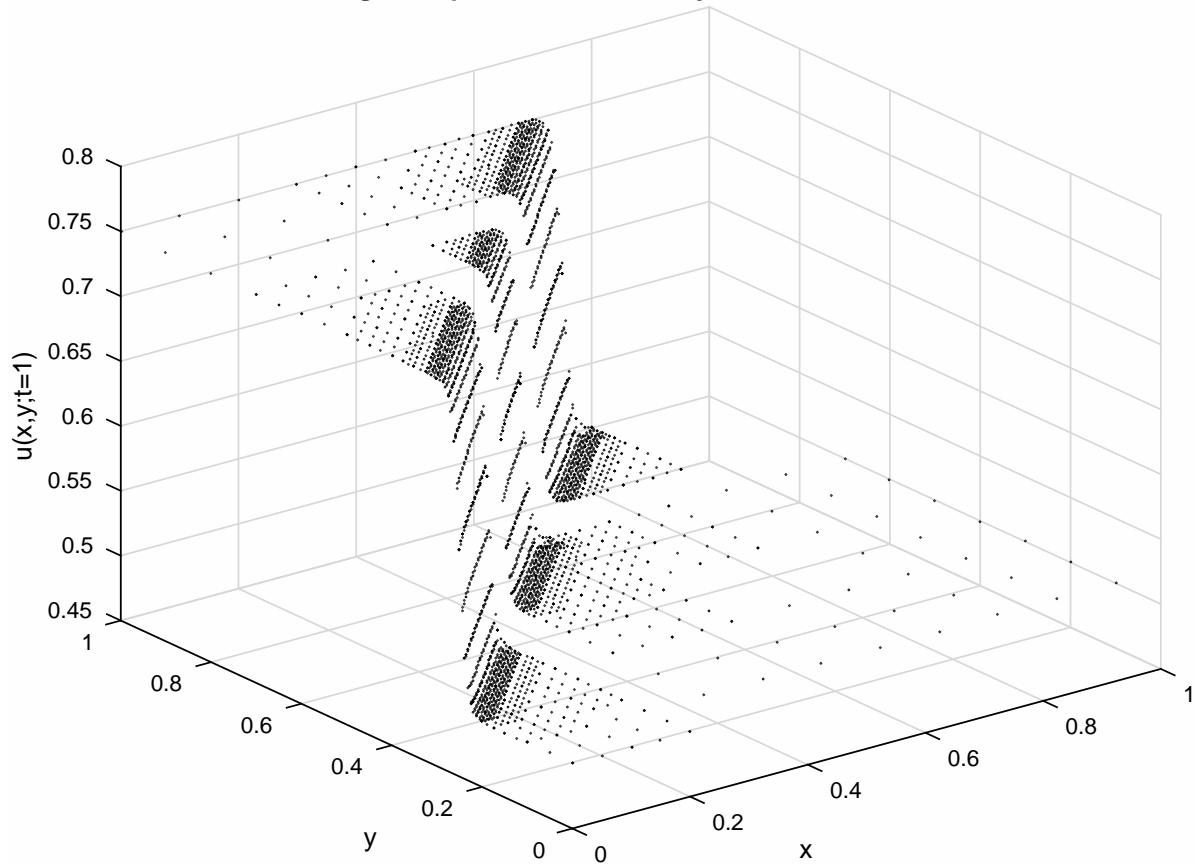
Total number of accepted timesteps = 45

Total number of rejected timesteps = 0

Total number of				
Residual evals	Jacobian evals	Newton iters	Lin sys iters	
At level				
1	630	45	90	45
2	630	45	90	78
3	630	45	90	87
4	630	45	90	124
5	575	41	83	122

Maximum number of		
Newton iters	Lin sys	iters

At level			
1	2	1	
2	2	1	
3	2	1	
4	2	2	
5	3	2	

2D Burgers' equation at t=1 on disjoint domain: u

2D Burgers' equation at t=1 on disjoint domain: v