Module 13.1: nag_pde_helm
Helmholtz Equations

`nag_pde_helm` provides a procedure for solving the Helmholtz equation in three dimensions.

Contents

Procedures

`nag_pde_helm_3d` 13.1.3

Solves the 3-d Helmholtz equation using a standard seven-point finite difference scheme and a fast Fourier transform method

Examples

Example 1: Solution of the Helmholtz equation with periodic boundary conditions 13.1.7

Example 2: Solution of the Helmholtz equation with mixed boundary conditions 13.1.9

References 13.1.12

Procedure: nag_pde_helm_3d

1 Description

nag_pde_helm_3d solves the three-dimensional Helmholtz equation in Cartesian co-ordinates:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} + \lambda u = f(x, y, z),$$

on the domain $x_{\min} \leq x \leq x_{\max}$, $y_{\min} \leq y \leq y_{\max}$, $z_{\min} \leq z \leq z_{\max}$.

Either the solution or the normal derivative of the solution may be specified on any of the boundaries, or the solution may be specified to be periodic in any of the three dimensions.

2 Usage

```
USE nag_pde_helm

CALL nag_pde_helm_3d(x_min, x_max, y_min, y_max, z_min, z_max, lambda, f &
                                [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**(n)' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$n_x \geq 6$ — the number of mesh points in the x-direction
$n_y \geq 6$ — the number of mesh points in the y-direction
$n_z \geq 6$ — the number of mesh points in the z-direction

3.1 Mandatory Arguments

x_min — real(kind=*wp*), intent(in)
x_max — real(kind=*wp*), intent(in)
Input: the lower and upper bounds of the range of x, i.e., x_min $\leq x \leq$ x_max.
Constraints: x_min $<$ x_max.

y_min — real(kind=*wp*), intent(in)
y_max — real(kind=*wp*), intent(in)
Input: the lower and upper bounds of the range of y, i.e., y_min $\leq y \leq$ y_max.
Constraints: y_min $<$ y_max.

z_min — real(kind=*wp*), intent(in)
z_max — real(kind=*wp*), intent(in)
Input: the lower and upper bounds of the range of z, i.e., z_min $\leq z \leq$ z_max.
Constraints: z_min $<$ z_max.

lambda — real(kind=*wp*), intent(in)

Input: the constant λ in the Helmholtz equation. For certain positive values of λ a solution to the differential equation may not exist, and close to these values the solution of the discretized problem will be extremely ill conditioned. In general the values of λ for which no solution exists cannot be predicted in advance. If $\lambda > 0$, then this procedure will still attempt to find a solution, but a warning will be issued and you are advised to treat any results with caution.

f(n_x, n_y, n_z) — real(kind=*wp*), intent(inout)

Input: the values of the right-hand side of the Helmholtz equation:

$$\texttt{f}(i,j,k) = f(x_i, y_j, z_k) \text{ for } i = 1,2,\ldots,n_x,\ j = 1,2,\ldots,n_y \text{ and } k = 1,2,\ldots,n_z,$$

where $x_i = x_{\min} + (i-1)/n_x \times (x_{\max} - x_{\min})$ for $i = 1,2,\ldots,n_x$ and similarly for y_j and z_k.

Note: if the solution is periodic in the x-direction say (which is assumed if none of the optional arguments `x_min_s`, `x_max_s`, `x_min_d` or `x_max_d` are present) then the procedure will check that $\texttt{f}(1,:,:)$ is equal to $\texttt{f}(n_x,:,:)$ within a small tolerance. If not, a warning will be issued. Similarly for the y- and z-directions.

Output: `f` contains the solution $u(i,j,k)$ of the finite difference approximation for the grid point (x_i, y_j, z_k) for $i = 1,2,\ldots,n_x$, $j = 1,2,\ldots,n_y$ and $k = 1,2,\ldots,n_z$.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

x_min_s(n_y, n_z) — real(kind=*wp*), intent(in), optional
x_min_d(n_y, n_z) — real(kind=*wp*), intent(in), optional
x_max_s(n_y, n_z) — real(kind=*wp*), intent(in), optional
x_max_d(n_y, n_z) — real(kind=*wp*), intent(in), optional

Input: the boundary values of the solution (u) or the derivative of the solution with respect to x (u_x) at `x_min` and `x_max`, i.e.,

`x_min_s`(j,k) contains $u($`x_min`$, y_j, z_k)$,
`x_min_d`(j,k) contains $u_x($`x_min`$, y_j, z_k)$,
`x_max_s`(j,k) contains $u($`x_max`$, y_j, z_k)$,
`x_max_d`(j,k) contains $u_x($`x_max`$, y_j, z_k)$,

for $j = 1,2,\ldots,n_y$, $k = 1,2,\ldots,n_z$.

Constraints: either none or two of these arguments may be supplied, and the only valid combinations are:

`x_min_s` and `x_max_s`,
`x_min_s` and `x_max_d`,
`x_min_d` and `x_max_s`,
`x_min_d` and `x_max_d`.

Default: if none of these arguments are present, the solution is assumed to be periodic in x.

y_min_s(n_x, n_z) — real(kind=*wp*), intent(in), optional
y_min_d(n_x, n_z) — real(kind=*wp*), intent(in), optional
y_max_s(n_x, n_z) — real(kind=*wp*), intent(in), optional
y_max_d(n_x, n_z) — real(kind=*wp*), intent(in), optional

Input: the boundary values of the solution (u) or the derivative of the solution with respect to y (u_y) at `y_min` and `y_max`, i.e.,

`y_min_s`(i,k) contains $u(x_i,$`y_min`$, z_k)$,
`y_min_d`(i,k) contains $u_y(x_i,$`y_min`$, z_k)$,

y_max_s(i,k) contains $u(x_i,$y_max$,z_k)$,

y_max_d(i,k) contains $u_y(x_i,$y_max$,z_k)$,

for $i = 1,2,\ldots,n_x$, $k = 1,2,\ldots,n_z$.

Constraints: either none or two of these arguments may be supplied, and the only valid combinations are:

y_min_s and y_max_s,

y_min_s and y_max_d,

y_min_d and y_max_s,

y_min_d and y_max_d.

Default: if none of these arguments are present, the solution is assumed to be periodic in y.

z_min_s(n_x,n_y) — real(kind=*wp*), intent(in), optional

z_min_d(n_x,n_y) — real(kind=*wp*), intent(in), optional

z_max_s(n_x,n_y) — real(kind=*wp*), intent(in), optional

z_max_d(n_x,n_y) — real(kind=*wp*), intent(in), optional

Input: the boundary values of the solution (u) or the derivative of the solution with respect to z (u_z) at z_min and z_max, i.e.,

z_min_s(i,j) contains $u(x_i,y_j,$z_min$)$,

z_min_d(i,j) contains $u_z(x_i,y_j,$z_min$)$,

z_max_s(i,j) contains $u(x_i,y_j,$z_max$)$,

z_max_d(i,j) contains $u_z(x_i,y_j,$z_max$)$,

for $i = 1,2,\ldots,n_x$, $j = 1,2,\ldots,n_y$.

Constraints: either none or two of these arguments may be supplied, and the only valid combinations are:

z_min_s and z_max_s,

z_min_s and z_max_d,

z_min_d and z_max_s,

z_min_d and z_max_d.

Default: if none of these arguments are present, the solution is assumed to be periodic in z.

pertrb — real(kind=*wp*), intent(out), optional

Output: pertrb = 0.0, unless a solution to Poisson's equation was requested (lambda = 0.0) and no solution values were supplied on any boundary (i.e., none of x_min_s, y_min_s, z_min_s, x_max_s, y_max_s and z_max_s were supplied). In this case a solution may not exist. pertrb is a constant which is calculated and subtracted from the array f in order to ensure that a solution exists. This procedure then computes this solution, which is a least-squares solution to the unperturbed problem. This solution is not unique and is unnormalised. The value of pertrb should be small compared to the right-hand side f, otherwise a solution has been obtained to an essentially different problem. This comparison should always be made to ensure that a meaningful solution has been obtained.

Constraints: pertrb must be supplied if lambda = 0.0 and none of x_min_s, y_min_s, z_min_s, x_max_s, y_max_s or z_max_s are supplied.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document nag_error_handling (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to nag_set_error before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.
304	Invalid presence of an optional argument.
305	Invalid absence of an optional argument.
320	The procedure was unable to allocate enough memory.

Warnings (error%level = 1):

error%code	Description
101	`lambda` > 0.0. A solution may not exist. For certain positive values of λ a solution to the Helmholtz equation may not exist, and close to these values the solution will be extremely ill conditioned. In general these values of λ cannot be predicted in advance. This procedure will attempt to find a solution, but you should treat any results with caution.
102	The solution is assumed to be periodic in at least one of the directions, but the input values of `f` are not equal on corresponding boundaries. Check the input values of `f`. This warning can be avoided by explicitly setting the values to be equal at corresponding points on each boundary, but note that there is no other internal check on periodicity.

5 Examples of Usage

Complete examples of the use of this procedure appear in Examples 1 and 2 of this module document.

6 Further Comments

6.1 Algorithmic Detail

This procedure forms the system of linear equations resulting from the standard seven-point finite difference discretisation on a uniform mesh, and then solves the system using a method based on the fast Fourier transform (FFT) described by Swarztrauber [1]. This procedure is based on the routine HW3CRT from FISHPACK (see Swarztrauber and Sweet [2]).

The procedure replaces all the second derivatives by second-order central difference approximations, resulting in a block tridiagonal system of linear equations. The equations are modified to allow for the prescribed boundary conditions. By taking the discrete Fourier transform in the x- and y-directions, the equations are reduced to sets of tridiagonal systems of equations.

Example 1: Solution of the Helmholtz equation with periodic boundary conditions

This example program calls `nag_pde_helm_3d` to solve the Helmholtz equation with $\lambda = -1$ and

$$f(x, y, z) = -4(\cos x \cos y \cos z + \sin x \sin y \sin z),$$

on the domain $0 \leq x \leq 2\pi$, $\pi \leq y \leq 3\pi$, $-\pi \leq z \leq \pi$.

The problem is periodic in all directions and so boundary conditions need not be specified.

The global error is obtained from the difference between the approximate solution calculated by the procedure and the exact solution given by

$$u(x, y, z) = \cos x \cos y \cos z + \sin x \sin y \sin z.$$

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_pde_helm_ex01

  ! Example Program Text for nag_pde_helm
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_math_constants, ONLY : nag_pi
  USE nag_pde_helm, ONLY : nag_pde_helm_3d
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC ABS, COS, KIND, MAX, MAXVAL, REAL, SIN
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, j, k, nx, ny, nz
  REAL (wp) :: dx, dy, dz, err, global_err, lambda, pi, x_max, x_min, &
   y_max, y_min, z_max, z_min
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: f(:,:,:), x(:), y(:), z(:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_pde_helm_ex01'

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) nx, ny, nz

  ALLOCATE (f(nx,ny,nz),x(nx),y(ny),z(nz)) ! Allocate storage

  pi = nag_pi(0.0_wp)
  x_min = 0.0_wp
  x_max = 2.0_wp*pi
  y_min = pi
  y_max = 3.0_wp*pi
  z_min = -pi
  z_max = pi
  lambda = -1.0_wp

  ! Define the grid points for later use
  dx = (x_max-x_min)/REAL(nx-1,kind=wp)
  dy = (y_max-y_min)/REAL(ny-1,kind=wp)
```

```
      dz = (z_max-z_min)/REAL(nz-1,kind=wp)
      x = (/ (x_min+dx*REAL(i-1,kind=wp),i=1,nx-1), x_max/)
      y = (/ (y_min+dy*REAL(j-1,kind=wp),j=1,ny-1), y_max/)
      z = (/ (z_min+dz*REAL(k-1,kind=wp),k=1,nz-1), z_max/)

      ! Define the values of the right-hand side of the Helmholtz eqn.
      DO k = 1, nz
        DO j = 1, ny
          f(:,j,k) = -4.0_wp*(COS(x)*COS(y(j))*COS(z(k))+SIN(x)*SIN(y(j))*SIN( &
           z(k)))
        END DO
      END DO

      ! Problem is periodic in all directions, so set boundary values
      ! equal (to avoid possible warning message 102)
      f(nx,:,:) = f(1,:,:)
      f(:,ny,:) = f(:,1,:)
      f(:,:,nz) = f(:,:,1)

      CALL nag_pde_helm_3d(x_min,x_max,y_min,y_max,z_min,z_max,lambda,f)

      ! Compute global error.  The exact solution to the problem
      ! is u(x,y,z) = cos(x)*cos(y)*cos(z) + sin(x)*sin(y)*sin(z)
      global_err = 0.0_wp
      DO k = 1, nz
        DO j = 1, ny
          err = MAXVAL(ABS(f(:,j,k)-(COS(x)*COS(y(j))*COS(z(k))+SIN(x)*SIN(y(j &
           ))*SIN(z(k)))))
          global_err = MAX(global_err,err)
        END DO
      END DO

      WRITE (nag_std_out,*)
      WRITE (nag_std_out,'(1X,A,ES11.2)') 'Global error = ', global_err

      DEALLOCATE (f,x,y,z)         ! Deallocate storage

    END PROGRAM nag_pde_helm_ex01
```

2 Program Data

```
Example Program Data for nag_pde_helm_ex01
15 15 15       : Values of nx, ny, nz
```

3 Program Results

```
 Example Program Results for nag_pde_helm_ex01

 Global error =    1.27E-02
```

Example 2: Solution of the Helmholtz equation with mixed boundary conditions

This example program calls `nag_pde_helm_3d` to solve the Helmholtz equation with $\lambda = -2$ and

$$f(x, y, z) = 4x^2(3 - x^2)\sin y \cos z,$$

on the domain $0 \leq x \leq 1$, $0 \leq y \leq 2\pi$, $0 \leq z \leq \pi/2$.

The solution is specified on the $x_{\min}$, $x_{\max}$ and $z_{\min}$ boundaries, and the derivative is specified on the $z_{\max}$ boundary. The required values are obtained from the exact solution

$$u(x, y, z) = x^4 \sin y \cos z.$$

The problem is periodic in the y-direction, and so no boundary conditions are required at the $y_{\min}$ and $y_{\max}$ boundaries.

The global error is obtained from the difference between the approximate solution calculated by the procedure and the exact solution.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_pde_helm_ex02

  ! Example Program Text for nag_pde_helm
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_math_constants, ONLY : nag_pi
  USE nag_pde_helm, ONLY : nag_pde_helm_3d
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC ABS, COS, KIND, MAX, MAXVAL, REAL, SIN
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, j, k, nx, ny, nz
  REAL (wp) :: cos_z, dx, dy, dz, err, global_err, lambda, pi, x_max, &
   x_min, y_max, y_min, z_max, z_min
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: f(:,:,:), sin_y(:), x(:), x_4(:), &
   x_max_s(:,:), x_min_s(:,:), y(:), z(:), z_max_d(:,:), z_min_s(:,:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) 'Example Program Results for nag_pde_helm_ex02'

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) nx, ny, nz

  ALLOCATE (f(nx,ny,nz),sin_y(ny),x_4(nx),x_min_s(ny,nz),x_max_s(ny,nz), &
   z_min_s(nx,ny),z_max_d(nx,ny),x(nx),y(ny),z(nz)) ! Allocate storage

  pi = nag_pi(0.0_wp)
  x_min = 0.0_wp
  x_max = 1.0_wp
  y_min = 0.0_wp
  y_max = 2.0_wp*pi
```

```
  z_min = 0.0_wp
  z_max = pi/2.0_wp
  lambda = -2.0_wp

  ! Define the grid points for later use
  dx = (x_max-x_min)/REAL(nx-1,kind=wp)
  dy = (y_max-y_min)/REAL(ny-1,kind=wp)
  dz = (z_max-z_min)/REAL(nz-1,kind=wp)
  x = (/ (x_min+dx*REAL(i-1,kind=wp),i=1,nx-1), x_max/)
  y = (/ (y_min+dy*REAL(j-1,kind=wp),j=1,ny-1), y_max/)
  z = (/ (z_min+dz*REAL(k-1,kind=wp),k=1,nz-1), z_max/)

  ! Define the values of the right-hand side of the Helmholtz eqn.
  x_4 = x*x
  x_4 = 4.0_wp*x_4*(3.0_wp-x_4)
  DO k = 1, nz
    cos_z = COS(z(k))
    DO j = 1, ny
      f(:,j,k) = (SIN(y(j))*cos_z)*x_4
    END DO
  END DO

  ! Problem is periodic in y direction, so set y boundary values
  ! equal (to avoid possible warning message 102)
  f(:,ny,:) = f(:,1,:)

  ! Define the arrays of x-boundary values
  x_min_s(:,:) = 0.0_wp
  sin_y = SIN(y)
  DO k = 1, nz
    x_max_s(:,k) = COS(z(k))*sin_y
  END DO

  ! Define the arrays of z-boundary values
  x_4 = x**4
  DO j = 1, ny
    z_min_s(:,j) = x_4*SIN(y(j))
  END DO
  z_max_d = -z_min_s

  CALL nag_pde_helm_3d(x_min,x_max,y_min,y_max,z_min,z_max,lambda,f, &
   x_min_s=x_min_s,x_max_s=x_max_s,z_min_s=z_min_s,z_max_d=z_max_d)

  ! Compute discretization error.  The exact solution to the
  ! problem is u(x,y,z) = x**4*sin(y)*cos(z)
  global_err = 0.0_wp
  DO k = 1, nz
    cos_z = COS(z(k))
    DO j = 1, ny
      err = MAXVAL(ABS(f(:,j,k)-(SIN(y(j))*cos_z)*x_4))
      global_err = MAX(global_err,err)
    END DO
  END DO

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,'(1X,A,ES11.2)') 'Global error = ', global_err

  DEALLOCATE (f,sin_y,x_4,x_min_s,x_max_s,z_min_s,z_max_d,x,y, &
   z)                          ! Deallocate storage

END PROGRAM nag_pde_helm_ex02
```

2 Program Data

```
Example Program Data for nag_pde_helm_ex02
17 33 21       : Values of nx, ny, nz
```

3 Program Results

```
 Example Program Results for nag_pde_helm_ex02

 Global error =    5.18E-04
```

References

[1] Swarztrauber P N (1984) Fast Poisson solvers *Studies in Numerical Analysis* (ed G H Golub) Mathematical Association of America

[2] Swarztrauber P N and Sweet R A (1979) Efficient Fortran subprograms for the solution of separable elliptic partial differential equations *ACM Trans. Math. Software* **5** 352–364