

# MUDPACK Documentation

This description consists of the following parts:

1. [Introduction](#)
2. [Special Features](#)
3. [Solver and Test Program Overview](#)
4. [Solver Selection Flowchart](#)
5. [Parallel Performance via OpenMP](#)
6. [Information on Earlier Versions](#)
7. [MUDPACK and Multigrid References](#)
8. [Obtaining MUDPACK Software](#)
9. [Acknowledgements](#)

## 1. Introduction

MUDPACK is a collection of portable, Fortran 77 subprograms, with a few Fortran90 extensions, for efficiently solving linear elliptic Partial Differential Equations (PDEs) using multigrid iteration. The most frequent Fortran90 extension used is the DO-END DO loop. Users may compile with any Fortran90 compiler, but Fortran77 compilers usually accept the code without complaint. In Fortran90 terminology, the source code is fixed-form, not free-form. OpenMP directives are included to enable shared memory parallelism. The package was created to make multigrid iteration available in user friendly form. The software is written in much the same format as the separable elliptic PDE package FISHPACK [5]. It extends the domain of solvable problems to include both separable and nonseparable PDEs. Detailed descriptions of earlier versions of MUDPACK are given [2,9].

Multigrid iteration (see [13,14,16,18,19,21]) combines classical iterative techniques, such as Gauss-Seidel line or point relaxation, with subgrid refinement procedures to yield a method superior to the iterative techniques alone. By iterating and transferring approximations and corrections at subgrid levels, a good initial guess and rapid convergence at the fine grid level can be achieved. Multigrid iteration requires less storage and computation than direct methods for nonseparable elliptic PDEs (e.g., see [7]) and is competitive with direct methods such as cyclic reduction [5,15,25,26] for separable equations. In particular, three-dimensional problems can often be handled at reasonable computational cost. Achieving optimal multigrid performance requires hand-tailored coding for certain problems. The generality of the equations solved by MUDPACK software may sometimes result in loss of efficiency. It is hoped that this is compensated for by the package's ease of use, applicability to a wide range of real problems (including those typically encountered in the atmospheric sciences at NCAR [8,12]), and avoidance of repeated "re-inventions of the wheel." Savings in human code development time can be at least as important as economic use of machine cycles. With careful selection of relaxation and multigrid parameters, optimal performance can often be realized using MUDPACK software. See [1,3,9,10] for a variety of problems where discretization level error (i.e., the same error that a direct method will produce) is reached in only one full multigrid cycle using MUDPACK solvers. Supercomputer performance from a decade ago [23] was measured with the examples in [1,7,9,10].

Reference [1] below  
is <http://nldr.library.ucar.edu/repository/assets/technotes/asset-000-000-000-167.pdf>. Readers may wish to also refer to it in learning about MUDPACK, since it has some information not provided otherwise on this website.

---

## 2. Special Features

\* Solving Linear Elliptic PDEs in a Variety of Forms

These forms include real and complex, two- and three-dimensional, self-adjoint, separable and nonseparable, and PDEs with cross derivative terms (see part 3 of this file).

### \* **Solving PDEs in Curvilinear Coordinate Systems**

The solution regions are rectangular regions in the sense that the domain of each independent variable must be a bounded interval on the real line. This means that curvilinear coordinate systems such as spherical or cylindrical coordinates are acceptable. The codes are not restricted to Cartesian coordinates.

### \* **Generating Second- and Fourth-order Approximations**

Standard second-order finite difference approximations are generated on uniform grids superimposed on the solution region. These can be improved to fourth-order estimates using "deferred corrections" ([22,24]).

### \* **Handling of General Boundary Conditions**

Any combination of periodic, specified (Dirichlet), and mixed derivative boundary conditions is allowed. Some of the solvers allow oblique (non-normal) derivative boundary conditions.

### \* **Ease of Input of the Continuous Problem**

User defined input subroutines are the mechanisms for passing PDE coefficients and boundary conditions.

### \* **Automatic Discretization of the Continuous Problem**

The discretization is transparent to a user who only needs to supply the PDE, boundary conditions, and grid size information. Standard second-order finite difference formula are used to approximate first and second partial derivatives. The result is a linear block tridiagonal system of equations. More complex difference formula (asymmetric near boundaries [4]) are used with the fourth-order solvers. The coefficients multiplying the second partial derivatives in the PDE are adjusted during discretization at coarser grid levels if there are nonzero first-order coefficients which would destroy diagonal dominance. This is necessary to preserve convergence of iterative schemes.

## \* Use of Multigrid Iteration to Approximate the Discretization Equations

This is the essential feature of the MUDPACK software. It makes this complex collection of integrated numerical procedures available in friendly form.

## \* Flexibility in Choosing Grid Size

Second and fourth order approximations are generated on uniform  $I$  by  $m$  by  $n$  grids superimposed on boxes in three dimensions or  $I$  by  $m$  grids superimposed on rectangles in two dimensions. The grid sizes have the form:

$$I = p * 2^{(i-1)} + 1$$

$$m = q * 2^{(j-1)} + 1$$

$$n = r * 2^{(k-1)} + 1$$

where  $p,q,r,i,j,k$  are positive integers.  $i,j,k > 0$  determine the number and size of the subgrid levels employed by multigrid cycling. Values for  $p,q,r$  should be chosen as small as possible (typically 2,3 or 5) and values for  $i,j,k$  as large as possible within grid size requirements for efficient cycling. In particular, larger values for  $p,q$  or  $r$  can cause algorithm deterioration. For 2-d and 3-d nonseparable PDEs this can be bypassed by using one of the "hybrid" solvers described below.

Let  $G$  denote the finest  $I$  by  $m$  by  $n$  grid. In MUDPACK, multigrid cycling is implemented on the ascending chain of grids

$$G(1) < \dots < G(s) < \dots < G(t) = G$$

where  $t = \max(i,j,k)$  and each  $G(s)$  ( $s=1,\dots,t$ ) has  $I(s)$  by  $m(s)$  by  $n(s)$  grid points given by:

$$I(s) = p * 2^{\max(0(i+s-t,1)} + 1$$

$$m(s) = q * 2^{\max(0(j+s-t,1)} + 1$$

$$n(s) = r * 2^{\max(0(k+s-t,1)} + 1$$

When grid size requirements cannot be met with MUDPACK software (even with one of the hybrid solvers described below) then one option is to choose a

grid which does satisfy the constraints which is as close as possible to the required grid and solve the problem there. The approximation can then be transferred to the required grid using multidimensional cubic interpolation.

#### \* Selection of Multigrid Options

MUDPACK has options for implementing variants of multigrid iteration and default options for those preferring black box solvers. The default options (chosen for robustness) set cubic prolongation, fully weighted residual restriction, and W(2,1) cycling. The earlier version of MUDPACK described in [2,3] only allowed V(2,1) cycling with linear prolongation. This is still available as a possibly more efficient choice for certain problems.

#### \* Selection of the Relaxation Method used within Multigrid Iteration

A relaxation menu is provided. It includes vectorized Gauss-Seidel schemes [17] on alternating points (red/black), lines (in any combination of directions) and planes (for three-dimensional anisotropic elliptic PDEs [27]). Choice of the correct relaxation method for a particular problem can be crucial. It depends on the relative grid and PDE coefficient size. Usually this can be pre-determined. Sometimes experimentation is required. Advice on method selection is given in the documentation.

#### \* Availability of "hybrid" Multigrid/Direct Method Solvers

The certainty of direct methods is combined with the efficiency of multigrid iteration by providing "hybrid" solvers for 2-d and 3-d nonseparable PDEs. Gaussian elimination is used whenever the coarsest grid is encountered within multigrid cycling. This eliminates the usual constraint that the coarsest grid must have "few" points thus giving additional flexibility in choosing grid size. It also provides a way to compare approximations from multigrid and direct method solutions. The hybrid codes become full direct method solvers replacing the codes described in [6] if grid size arguments are chosen so that the coarse and fine grids coincide. Large storage and computational requirements make the use of the 3-d hybrid codes **muh3,cuh3** as direct methods possible only for very coarse grids.

#### \* Availability of Subroutines to Compute Residuals

Subroutines to compute fine grid residual after calling any of the second-order solvers are provided. The residual measures how well the current approximation satisfies the linear system of equations coming from the discretization. Residual ratios can be used to estimate the convergence rate of multigrid iteration.

#### \* **No Initial Guess Requirement**

Unlike the case with classical iterative schemes, initial guesses are not necessary and should not be supplied unless they are very good (as, for example, when restarting multigrid iteration using an approximation generated earlier). Full multigrid cycling [13], beginning at the coarsest grid level, is used when there is no initial guess. Advice on how to use initial guesses within a time marching problem is given in the documentation.

#### \* **Non-initialization Calls**

Redundant discretization and matrix factorization processes can (and should) be bypassed on recalls to the software. For example, this happens when only the right-hand side array has changed from a previous call or when more multigrid cycles are needed for additional accuracy.

#### \* **Error Control**

Maximum relative error can be used to monitor convergence. Use of error control is optional and requires additional storage and computation.

#### \* **Flagging of Errors involving Input Parameters**

This includes detection of singular and/or nonelliptic PDEs. Fatal and nonfatal errors are flagged.

#### \* **Output of Exact Minimal Work Space Requirements**

This is especially important with three-dimensional problems where central memory is easily exhausted.

#### \* **Extensive Documentation and Test Programs**

Users are encouraged to carefully read the documentation and execute the test program for the solver to be used. The next section provides links to documentation and fortran test program files.

---

### 3. Solver and Test Program Overview

Table 1 below lists all mudpack two- and three-dimensional, second and fourth order solvers for real and complex elliptic partial differential equations with and without cross derivative terms. Clicking on a solver will bring up its documentation file.

Table 2 provides a list of the test and residual codes for each solver. These codes are meant as tests but also as example codes in guiding users through the setup and calling of MUDPACK routines in their own applications.

<b>Table 1</b> <b>An overview of MUDPACK solvers</b>	
<b>computation</b>	<b>subprograms</b>
2nd order/real 2D self-adjoint nonseparable	<a href="#">mud2sa</a>

2nd order/real 2D separable	<a href="#">mud2sp</a>
2nd order/real 2D nonseparable	<a href="#">muh2</a> , <a href="#">mud2</a>
2nd order/real 2D with cross term	<a href="#">muh2cr</a> , <a href="#">mud2cr</a>
4th order/real 2D separable	<a href="#">mud24sp</a>
4th order/real 2D nonseparable	<a href="#">muh24</a> , <a href="#">mud24</a>
4th order/real 2D with cross term	<a href="#">muh24cr</a> , <a href="#">mud24cr</a>
2nd order/real 3D self-adjoint nonseparable	<a href="#">mud3sa</a>
2nd order/real 3D separable	<a href="#">mud3sp</a>
2nd order/real 3D nonseparable	<a href="#">muh3</a> , <a href="#">mud3</a>
2nd order/real 3D with cross term	<a href="#">mud3cr</a>
4th order/real 3D separable	<a href="#">mud34sp</a>

4th order/real 3D nonseparable	<a href="#">mud34</a> , <a href="#">muh34</a>
2nd order/complex 2D separable	<a href="#">cud2sp</a>
2nd order/complex 2D nonseparable	<a href="#">cuh2</a> , <a href="#">cud2</a>
2nd order/complex 2D with cross term	<a href="#">cuh2cr</a> , <a href="#">cud2cr</a>
4th order/complex 2D separable	<a href="#">cud24sp</a>
4th order/complex 2D nonseparable	<a href="#">cud24</a> , <a href="#">cuh24</a>
4th order/complex 2D with cross term	<a href="#">cud24cr</a> , <a href="#">cuh24cr</a>
2nd order/complex 3D separable	<a href="#">cud3sp</a>
2nd order/complex 3D nonseparable	<a href="#">cuh3</a> , <a href="#">cud3</a>
2nd order/complex 3D with cross term	<a href="#">cud3cr</a>
4th order/complex 3D separable	<a href="#">cud34sp</a>

4th order/complex 3D nonseparable	<a href="#">cud34</a>	
-----------------------------------	-----------------------	--

<b>Table 2</b>	
<b>solver</b>	<b>test &amp; residual codes</b>
<b>mud2sa</b>	<a href="#">tmud2sa</a>
<b>mud2sp</b>	<a href="#">tmud2sp</a> , <a href="#">resm2sp</a>
<b>mud2,muh2</b>	<a href="#">tmud2</a> , <a href="#">tmuh2</a> , <a href="#">resm2</a>
<b>mud2cr,muh2cr</b>	<a href="#">tmud2cr</a> , <a href="#">tmuh2cr</a> , <a href="#">resm2cr</a>
<b>mud24sp</b>	<a href="#">tmud24sp</a>
<b>mud24,muh24</b>	<a href="#">tmud24</a> , <a href="#">tmuh24</a>
<b>mud24cr,muh24cr</b>	<a href="#">tmud24cr</a> , <a href="#">tmuh24cr</a>
<b>mud3sa</b>	<a href="#">tmud3sa</a>

<b>mud3sp</b>	<a href="#">tmud3sp</a> , <a href="#">resm3sp</a>
<b>mud3,muh3</b>	<a href="#">tmud3</a> , <a href="#">tmuh3</a> , <a href="#">resm3</a>
<b>mud3cr</b>	<a href="#">tmud3cr</a>
<b>mud34sp</b>	<a href="#">tmud34sp</a>
<b>mud34,muh34</b>	<a href="#">tmud34</a> , <a href="#">tmuh34</a>
<b>cud2sp</b>	<a href="#">tcud2sp</a> , <a href="#">resc2sp</a>
<b>cud2,cuh2</b>	<a href="#">tcud2</a> , <a href="#">tcuh2</a> , <a href="#">resc2</a>
<b>cud2cr,cuh2cr</b>	<a href="#">tcud2cr</a> , <a href="#">tcuh2cr</a> , <a href="#">resc2cr</a>
<b>cud24sp</b>	<a href="#">tcud24sp</a>
<b>cud24,cuh24</b>	<a href="#">tcud24</a> , <a href="#">tcuh24</a>
<b>cud24cr,cuh24cr</b>	<a href="#">tcud24cr</a> , <a href="#">tcuh24cr</a>

<b>cud3sp</b>	<a href="#">tcud3sp</a> , <a href="#">resc3sp</a>	
<b>cud3,cuh3</b>	<a href="#">tcud3</a> , <a href="#">tcuh3</a> , <a href="#">resc3</a>	
<b>cud34sp</b>	<a href="#">tcud34sp</a>	
<b>cud34</b>	<a href="#">tcud34</a>	

---

## 4. Solver Selection

The following "flow chart" can be used in selecting the appropriate second-order software for the elliptic PDE to be solved:

- (1) If the PDE is complex go to (9) else go to (2)
- (2) If the PDE is three-dimensional go to (6) else go to (3)
- (3) If the PDE is separable use **mud2sp** else go to (4)
- (4) If the PDE has a cross derivative use **muh2cr** or **mud2cr** else go to (5)
- (5) If the PDE is self-adjoint use **mud2sa** else use **muh2** or **mud2**.
- (6) If the PDE is separable use **mud3sp** else go to (7)
- (7) If the PDE is self-adjoint use **mud3sa** else go to (8)

- (8) If the PDE has cross derivatives use **mud3cr** else use **muh3** or **mud3**.
- (9) If the PDE is three dimensional go to (13) else go to (10)
- (10) If the PDE is separable use **cud2sp** else go to (11)
- (11) If the PDE has a cross derivative use **cuh2cr** or **cud2cr** else go to (12)
- (12) Use **cuh2** or **cud2**
- (13) If the PDE is separable use **cud3sp** else use **cuh3** or **cud3**.

Fourth-order solvers can improve the approximation if the corresponding second-order solver has reached discretization level error (i.e., the same error level that a direct method will reach) [1,3,10].

---

## 5. Parallel Performance via OpenMP

In single processor mode, the openMP statements in version 5.0.1 of MUDPACK are interpreted as comment cards not affecting execution. To ensure this is the case, users should check that their compilers do not recognize OpenMP directives by default. If this is the case, the directives can be turned off with compiler flags or removed by passing MUDPACK source code through an appropriate sed or awk script which removes lines beginning with C\$OMP.

Parallel performance was measured on on a Cray J9, a SGI Origin, and a two processor IBM SP computer, using the three MUDPACK solvers **mud2**, **mud3**, and **mud3cr**. The tables below record measured wall clock time in seconds for an increasing number of processors, **mp**. For each example and grid size, either the least expensive relaxation method (point relaxation with 5 multigrid cycles) or the more expensive and robust relaxation method (line relaxations in all directions with 3 multigrid cycles) is executed. This is typical

of the amount of computation needed to solve elliptic problems with the numerical methods embedded in MUDPACK.

The second example illustrates that the cost overhead for parallelization of medium resolution two-dimensional problems can cancel any advantage gained by selecting more than one processor. Once a resolution is chosen, some preliminary timings should be made before using a MUDPACK solver with more than one processor.

**Example 1:** (513 X 769 grid) executing 3 multigrid cycles using bi-directional line relaxations with the multigrid solver **mud2**.

Cray J9		SGI Origin		IBM SP	
time	mp	time	mp	time	mp
16.86	1	8.78	1	9.93	1
9.76	2	6.88	2	5.61	2
6.46	4	5.29	4		
5.33	8	4.30	8		
3.90	16	3.89	16		

**Example 2:** (257 X 193 grid) executing 5 multigrid cycles using red/black Gauss-Seidel point relaxation with the multigrid solver **mud2**.

Cray J9		SGI Origin		IBM SP	
time	mp	time	mp	time	mp
0.38	1	0.27	1	0.17	1
0.42	2	0.49	2	0.15	2
0.44	4	0.95	4		
0.44	8	0.99	8		
0.45	16	1.16	16		

**Example 3:** (95 X 65 X 129 grid) executing 5 multigrid cycles using red/black Gauss-Seidel point relaxation with the multigrid solver **mud3**.

Cray J9	SGI Origin	IBM SP

time	mp	time	mp	time	mp
8.21	1	9.83	1	6.63	1
4.97	2	8.46	2	4.03	2
3.33	4	5.17	4		
2.62	8	4.22	8		
2.54	16	3.13	16		

**Example 4:** (49 X 257 X 49 grid) executing to a prescribed error tolerance with point relaxation and then with line relaxations in 3 directions with the quasi-multigrid solver **mud3cr** (see the documentation **mud3cr.d** and the test program **tmud3cr.f** for a description of the problem approximated).

SGI Origin (point)		SGI Origin (lines)	
time	mp	time	mp
39.6	1	74.2	1

28.7	2	41.1	2
12.0	4	26.8	4
9.8	8	20.0	8
8.9	16	15.7	16

---

## 6. Information on Earlier Versions

The Fortran subroutines in MUDPACK discretize a variety of elliptic Partial Differential Equations (PDEs) and boundary conditions using finite difference formula on grids superimposed on the rectangular solution regions. Then multigrid iterative techniques are used with point, line, or planar relaxation to generate second- and fourth-order approximations to the underlying real and complex, two- and three-dimensional continuous problems.

The MUDPACK software package includes 124 files containing over 100,000 lines of Fortran77 code and documentation. The code may also be compiled with Fortran90 and 95 compilers, but you may need to provide correct flags since the source code is fixed rather than free source form.

The subroutine names, functionality, argument lists, test programs, and documentation files for version 5.0.1 and version 5.0 of MUDPACK are identical. 5.0.1 corrects an error in subroutine mud2sp, namely, an incorrect call to routine rcd2spp is replaced by a call to rmd2spp.

## **VERSION 5.0.1 INCOMPATIBLE WITH VERSIONS EARLIER THAN 4.0**

Version 5.0.1 of MUDPACK is the latest in a series of major revisions since the software was first released in 1990. Changes in argument lists, work space requirements, and file organization make the current version incompatible with versions of MUDPACK built before version 4.0. The functionality has been expanded by adding new solvers. To ensure portability, all MUDPACK 5.0.1 software has been passed through fortran verification software and compiled and executed on different platforms with both fortran77 and fortran90 compilers.

## **OUTLINE OF MAJOR CHANGES BETWEEN VERSIONS 4.0 AND 5.0**

### **(1) Open MP directives**

Open MP directives have been inserted in the time critical portions of each of 44 files within MUDPACK. This was done for do loops where red/black Gauss-Seidel point, line, and planar relaxation (for 3-d problems) are executed at different grid levels and in loops where weighted residual restrictions occur. The relaxation and residual restriction portion of multigrid codes account for at least 90% of the execution time with most problems. It was necessary to restructure line relaxation kernels to allow more efficient parallelization when open MP directives are inserted. The changes did not adversely affect single processor vector performance. See the parallel performance tables above, on this page, for improvement measurements on a variety of machines at the National Center for Atmospheric Research.

### **(2) Code Simplification and modernization.**

Code simplification has been achieved by expanding internal arrays to include virtual boundaries. This has reduced the code complexity required to handle the variety of boundary conditions MUDPACK software allows. Tests have indicated the resulting streamlining has yielded a 10-30% speedup for a single processor depending on the problem and resolution. Use of equivalencing between arrays has been eliminated. Statement labels have been eliminated,

all variable types declared, and nested loops have been streamlined and indented.

### (3) Separation of discretization and approximation calls

The first call to a solver discretizes the continuous elliptic PDE and boundary conditions using standard second-order finite difference formula. Afterwards, calls to generate approximations can be made. This is a natural separation analogous to separating the LUD factorization and solution phase in matrix solvers. Further it allows more appropriate efficiency monitoring of MUDPACK software since the discretization phase is heavily dependent upon user provided subroutines for inputting coefficients and boundary conditions. Only the approximation phase should be monitored for efficiency. Earlier versions of MUDPACK allowed discretization and approximation to occur with the same call.

### (4) Simplification of multigrid options

The variants in multigrid cycling have been simplified in version 5.0. F cycling and unweighted or half weighted residual restriction were deemed unnecessary and have been eliminated. V cycles or W cycles with fully weighted residual restriction are retained as the optimal choices. More general multigrid cycling is allowed but flagged as a nonfatal error indicating probable inefficient use of multigrid. The prolongation operator can still be either linear or cubic interpolation.

### (5) New three-dimensional solvers

New "hybrid" three-dimensional multigrid/direct method solvers **muh3** and **cuh3** along with a quasi-multigrid solver, **mud3cr**, for three-dimensional problems with cross derivative terms and possibly oblique derivative boundary have been added to the package. Use of hybrid solvers for nonsingular two- and three-dimensional problems is encouraged since they can be more robust than their "multigrid only" counterparts and they cost only marginally more in computation and storage. The hybrid solvers also provide more choices of grid resolution.

### (6) Four color relaxation

Four color relaxation has replaced red/black relaxation for PDEs with cross derivative terms. This can provide better convergence rates.

### (7) Correction in planar relaxation

If planar relaxation is selected for three-dimensional anisotropic PDEs then full two-dimensional multigrid cycling is implemented for each plane visited during three dimensional multigrid cycling. Earlier versions of MUDPACK did not use full two-dimensional cycling.

---

## 7. MUDPACK and Multigrid References

- [1] J. Adams, "Multigrid Software for Elliptic Partial Differential Equations: MUDPACK," NCAR Technical Note-357+STR, Feb. 1991, 51 pages. The link for this document is <http://nldr.library.ucar.edu/repository/assets/technotes/asset-000-000-000-167.pdf>.
- [2] J. Adams, "MUDPACK: Multigrid Fortran Software for the Efficient Solution of Linear Elliptic Partial Differential Equations," Applied Math. and Comput. Vol.34, Nov 1989, pp.113-146.
- [3] J. Adams, "FMG Results with the Multigrid Software Package MUDPACK," Proceedings of the Fourth Copper Mountain Conference on Multigrid, SIAM, 1989, pp.1-12.
- [4] J. Adams, "Fortran Subprograms for Finite Difference Formula," J. Comp. Phys., Vol 26, Jan 1978, pp. 113-116.
- [5] J. Adams, P. Swarztrauber, R. Sweet, "Efficient Fortran Subprograms for the Solution of Elliptic Partial Differential Equations," Elliptic Problem Solvers, Academic Press, 1982, pp.333-390.

- [6] J. Adams, "New Software for Elliptic Partial Differential Equations," Computing Facility Notes 55, November 1978
- [7] J. Adams, "Comparison of direct and iterative methods for approximating nonseparable elliptic PDEs at NCAR," SCD Computing News, Vol 10, Nov. 1989, pp.12-14.
- [8] J. Adams, R. Garcia, B. Gross, J. Hack, D. Haidvogel, and V. Pizzo, "Applications of Multigrid Software in the Atmospheric Sciences," Monthly Weather Review, Vol. 120 # 7, July 1992, pp. 1447-1458.
- [9] J. Adams, "MUDPACK: Multigrid Software for Linear Elliptic Partial Differential Equations," SCD UserDoc, Version 2.0, NCAR, February 1990.
- [10] J. Adams, "Recent Enhancements in MUDPACK, A Multigrid Software Package for Elliptic Partial Differential Equations," Applied Math. and Comp., Vol. 43, May 1991, pp.79-94.
- [11] J. Adams, "MUDPACK-2: Multigrid Software for Elliptic Partial Differential Equations on Uniform Grids with any Resolution," Applied. Math. and Comp., Vol. 53, February 1993, pp. 235-249.
- [12] J. Adams, and P. Smolarkiewicz, "Modified multigrid for 3D elliptic equations with cross derivatives", Applied Math. Comput., Vol. 121, 2001, pp. 301-312.
- [13] A. Brandt, "Multi-level Adaptive Solutions to Boundary Value Problems," Math. Comp., 31, 1977, pp.333-390.
- [14] W. Briggs, "A Multigrid Tutorial," SIAM, Philadelphia, 1987.
- [15] B. Buzbee, G. Golub, and C. Nielson, "On direct methods for solving Poisson's equations," SIAM J. Numer. Anal., 7, 1970, pp.627-656.
- [16] S. Fulton, R. Ciesielski, and W. Schubert, Multigrid methods for elliptic problems: a review. Monthly Weather Review, 114:943-959 (1986).
- [17] W. Gentzsch, "Vectorization of Computer Programs with Applications to Computational Fluid Dynamics," Vieweg & Sohn, 1984 (246 pages).

- [18] W. Hackbush and U. Trottenberg, "Multigrid Methods," Springer-Verlag, Berlin, 1982.
- [19] D. Jespersen, "Multigrid Methods for Partial Differential Equations." Studies in Numerical Analysis, Vol.24, MAA, 1984.
- [20] J. Mandel and S. Parter, "On the Multigrid F-Cycle," Applied Math. and Comput., Vol 37, 1990, pp.19-36.
- [21] S. McCormick, "Multigrid Methods," Vol 3 of SIAM Frontiers Series, SIAM, Philadelphia, 1987.
- [22] V. Pereyra, "Highly Accurate Numerical Solution of Casilinear Elliptic Boundary-Value Problems in n Dimensions," Math. Comp., 24, 1970, pp.771-783.
- [23] D. Sato, "PERFMON: The Cray Performance Monitor Utility," SCD UserDoc, Version 2.0, NCAR, March 1989.
- [24] S. Schaffer, "Higher Order Multigrid Methods," Math. Comp., Vol 43, July 1984, pp. 89-115.
- [25] P. Swarztrauber, "Fast Poisson Solvers," Studies in Numerical Analysis, Math. Assoc. America, 1985, pp. 319-369.
- [26] R. Sweet, "A Parallel and Vector Variant of the Cyclic Reduction Algorithm," SIAM J. Sci. and Stat. Comp., Vol. 9, July 1988, pp. 761-766.
- [27] C. Thole and U. Trottenberg, "Basic Smoothing Procedures for the Multigrid treatment of Elliptic 3D Operators," Applied Math. and Comp., 19, 1986, pp. 333-345.

## **8. Obtaining MUDPACK Software**

MUDPACK solver routines and test programs can be downloaded from the NCAR web page for MUDPACK. You will see a download tab on that page. When you click on it, you will be asked to agree to the licensing terms for the package.

---

## **9. Acknowledgements**

Steve McCormick introduced the author to the multigrid community and provided numerous helpful suggestions including the use of planar relaxation with the three-dimensional solvers. The importance of adjusting discretization coefficients at coarser grid levels for PDEs with nonzero first-order terms was pointed out by Klauss Steuben. Achi Brandt provided a complimentary foreword for the MUDPACK technical note [1]. A conversation with Achi Brandt affirmed that the default multigrid options in MUDPACK are a good choice and that the use of deferred corrections in obtaining fourth-order approximations with multigrid is a reasonable strategy. Dave Kennison, now retired, formerly of the NCAR graphics group provided the grid coarsening figure at the start of this document.

---

[Return to beginning of this document](#)

## Text Below Contains Internal Files Referenced by Above Links

### CUD2

```
C
C      file cud2.d
C
C      * * * * * * * * * * * * * * * * * * * * *
* * * * * * *
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *               University Corporation for Atmospheric
Research           *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK version 5.0.1
*
C      *
*
C      *               A Fortran Package of Multigrid
*
C      *
*
C      *               Subroutines and Example Programs
*
C      *
*
C      *               for Solving Elliptic Partial Differential
Equations        *
C      *
*
C      *               by
*
```

```
C      *
*
C      *                               John Adams
*
C      *
*
C      *                               of
*
C      *
*
C      *                               the National Center for Atmospheric
Research          *
C      *
*
C      *                               Boulder, Colorado (80307)
U.S.A.          *
C      *
*
C      *                               which is sponsored by
*
C      *
*
C      *                               the National Science Foundation
*
C      *
*
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C
C ... file cud2.d
C
C     contains documentation for:
C     subroutine
cud2(iparm,fparm,work,coef,bndyc,rhs,phi,mgopt,ierror)
C     A sample fortran driver is file "tcud2.f".
C
C ... required MUDPACK files
C
C     cudcom.f
C
C ... purpose
C
C     subroutine cud2 automatically discretizes and
attempts to compute
```

```

c      the second-order difference approximation to the
complex 2-d
c      linear nonseparable elliptic partial differential
equation on a
c      rectangle. the approximation is generated on a
uniform grid covering
c      the rectangle (see mesh description below).
boundary conditions
c      may be specified (dirchlet), periodic, or mixed
derivative in any
c      combination. the form of the pde solved is:
c
c
c      cxx(x,y)*pxx + cyy(x,y)*pyy + cx(x,y)*px +
cy(x,y)*py +
c
c      ce(x,y)*p(x,y) = r(x,y).
c
c
c      pxx,pyy,px,py are second and first partial
derivatives of the
c      unknown real solution function p(x,y) with respect
to the
c      independent variables x,y. cxx,cyy,cx,cy,ce are
the known
c      complex coefficients of the elliptic pde and
r(x,y) is the known
c      complex right hand side of the equation. The real
and imaginary
c      parts of cxx and cyy should be positive for all
x,y in the solution
c      region. Nonseparability means some of the
coefficients depend on
c      both x and y. Otherwise the PDE is separable and
subroutine
c      cud2sp should be used instead of cud2
c
c
c ... mesh description . . .
c
c      the approximation is generated on a uniform nx by
ny grid. the grid
c      is superimposed on the rectangular solution region
c

```

```
c [xa,xb] x [yc,yd].  
c  
c let  
c  
c dlx = (xb-xa) / (nx-1), dly = (yd-yc) / (ny-1)  
c  
c be the uniform grid increments in the x,y  
directions. then  
c  
c xi=xa+(i-1)*dlx, yj=yc+(j-1)*dly  
c  
c for i=1,...,nx and j=1,...,ny denote the x,y  
uniform mesh points  
c  
c  
c ... language  
c  
c fortran90/fortran77  
c  
c  
c ... portability  
c  
c mudpack5.0.1 software has been compiled and tested  
with Fortran77  
c and Fortran90 on a variety of platforms.  
c  
c ... methods  
c  
c details of the methods employed by the solvers in  
mudpack are given  
c in [1,9]. [1,2,9] contain performance  
measurements on a variety of  
c elliptic pdes (see "references" in the file  
"readme"). in summary:  
c  
c *** discretization and solution (second-order solvers)  
(see [1])  
c  
c the pde and boundary conditions are automatically  
discretized at all  
c grid levels using second-order finite difference  
formula. diagonal  
c dominance at coarser grid levels is maintained in  
the presence of
```

```
c      nonzero first-order terms by adjusting the second-
order coefficient
c      when necessary. the resulting block tri-diagonal
linear system is
c      approximated using multigrid iteration
[10,11,13,15,16,18]. version
c      5.0.1 of mudpack uses only fully weighted residual
restriction. defaults
c      include cubic prolongation and w(2,1) cycles.
these can be overridden
c      with selected multigrid options (see "mgopt").
error control based on
c      maximum relative differences is available. full
multigrid cycling (fmg)
c      or cycling beginning or restarting at the finest
grid level can be
c      selected. a menu of relaxation methods including
gauss-seidel point,
c      line relaxation(s) (in any combination of
directions) and planar
c      relaxation (for three-dimensional anisotropic
problems) are provided.
c      all methods use ordering based on alternating
points (red/black),
c      lines, or planes for cray vectorization and
improved convergence
c      rates [14].
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
estimate (i.e.,
c      discretization level error is reached) then this
can be improved to a
c      fourth-order estimate using the technique of
"deferred corrections."
c      the values in the solution array are used to
generate a fourth-order
c      approximation to the truncation error. second-
order finite difference
c      formula are used to approximate third and fourth
partial derivatives
c      of the solution function [3]. the truncation
```

```
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side.  the default
multigrid options
c      are used to compute the fourth-order correction
term which is added
c      to the original solution array.
c
c
c ... references (partial)
c
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
c      Solution of Linear Elliptic Partial Differential
Equations,"
c      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
c
c      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
c      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
c      1989, pp.1-12.
c      .
c      .
c      .
c      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
Haidvogel, and V. Pizzo,
c      "Applications of Multigrid Software in the
Atmospheric Sciences,"
c      Mon. Wea. Rev.,vol. 120 # 7, July 1992, pp. 1447-
1458.
c      .
c      .
c      .
c      [9] J. Adams, "Recent Enhancements in MUDPACK, a
Multigrid Software
c      package for Elliptic Partial Differential
Equations," Applied Math.
c      and Comp., 1991, vol. 43, May 1991, pp. 79-94.
c
c      [10]J. Adams, "MUDPACK-2: Multigrid Software for
Approximating
```

```
c      Elliptic Partial Differential Equations on Uniform
Grids with
c      any Resolution," Applied Math. and Comp., 1993,
vol. 53, February
c      1993, pp. 235-249
C      .
C      .
C      .
C
C ... argument description
C
C
C ****
***** *****
C *** input arguments
***** *****
C
***** *****
***** *****
C
C
C ... iparm
C
C      an integer vector of length 17 used to pass
integer
C      arguments. iparm is set internally and
defined as
C      follows:
C
C
C ... intl=iparm(1)
C
C      an initialization argument. intl=0 must be
input
C      on an initial call. in this case input
arguments will
C      be checked for errors and the elliptic
partial differential
C      equation and boundary conditions will be
discretized using
C      second order finite difference formula.
C
C ***      An approximation is NOT generated after an
```

```
intl=0 call!
c           cud2 should be called with intl=1 to
approximate the elliptic
c           PDE discretized by the intl=0 call.  intl=1
should also
c           be input if cud2 has been called earlier and
only the
c           values in in rhs (see below) or gbdy (see
bndyc below)
c           or phi (see below) have changed.  This will
bypass
c           redundant pde discretization and argument
checking
c           and save computational time.  Some examples
of when
c           intl=1 calls should be used are:
c
c           (0) after a intl=0 argument checking and
discretization call
c
c           (1) cud2 is being recalled for additional
accuracy.  In
c           this case iguess=iparm(12)=1 should also
be used.
c
c           (2) cud2 is being called every time step in a
time dependent
c           problem (see discussion below) where the
elliptic operator
c           does not depend on time.
c
c           (3) cud2 is being used to solve the same
elliptic equation
c           for several different right hand sides
(igueess=0 should
c           probably be used for each new righthand
side).
c
c           intl = 0 must be input before calling with
intl = 1 when any
c           of the following conditions hold:
c
c           (a) the initial call to cud2
```

```
c           (b) any of the integer arguments other than
iguess=iparm(12)
c           or maxcy=iparm(13) or mgopt have changed
since the previous
c           call.
c
c           (c) any of the floating point arguments other
than tolmax=
c           fparm(5) have changed since the previous
call
c
c           (d) any of the coefficients input by coef
(see below) have
c           changed since the previous call
c
c           (e) any of the "alfa" coefficients input by
bndyc (see below)
c           have changed since the previous call.
c
c           If any of (a) through (e) are true then the
elliptic PDE
c           must be discretized or rediscretized.  If
none of (a)
c           through (e) holds, calls can be made with
intl=1.
c           Incorrect calls with intl=1 will produce
erroneous results.
c   ***      The values set in the saved work space "work"
(see below) with
c           an intl=0 call must be preserved with
subsequent intl=1 calls.
c
c           MUDPACK software performance should be
monitored for intl=1
c           calls.  The intl=0 discretization call
performance depends
c           primarily on the efficiency or lack of
efficiency of the
c           user provided subroutines for pde
coefficients and
c           boundary conditions.
c
c
c ... nxa=iparm(2)
```

```

C
C           flags boundary conditions on the edge x=xa
C
C           = 0 if p(x,y) is periodic in x on [xa,xb]
C           (i.e., p(x+xb-xa,y) = p(x,y) for all x,y
C           (if nxa=0 then nxb=0 is required, see
ierror = 2)
C
C           = 1 if p(xa,y) is specified (this must be input
thru phi(1,j))
C
C           = 2 if there are mixed derivative boundary
conditions at x=xa
C           (see bndyc)
C
C
C ... nxb=iparm(3)
C
C           flags boundary conditions on the edge x=xb
C
C           = 0 if p(x,y) is periodic in x on [xa,xb]
C           (i.e., p(x+xb-xa,y) = p(x,y) for all x,y)
C           (if nxb=0 then nxa=0 is required, see
ierror = 2)
C
C           = 1 if p(xb,y) is specified (this must be input
thru phi(nx,j))
C
C           = 2 if there are mixed derivative boundary
conditions at x=xb
C           (see bndyc)
C
C
C ... nyc=iparm(4)
C
C           flags boundary conditions on the edge y=yc
C
C           = 0 if p(x,y) is periodic in y on [yc,yd]
C           (i.e., p(x,y+yd-yc) = p(x,y) for all x,y
C           (if nyc=0 then nyd=0 is required, see
ierror = 2)
C
C           = 1 if p(x,yc) is specified (this must be input
thru phi(i,1))

```

```

c
c      = 2 if there are mixed derivative boundary
conditions at y=yc
c          (see bndyc)
c
c
c ... nyd=iparm(5)
c
c      flags boundary conditions on the edge y=yd
c
c      = 0 if p(x,y) is periodic in y on [yc,yd]
c          (i.e., p(x,y+yd-yc) = p(x,y) for all x,y
c          (if nyd=0 then nyc=0 is required, see
ierror = 2)
c
c      = 1 if p(x,yd) is specified (this must be input
thru phi(i,ny))
c
c      = 2 if there are mixed derivative boundary
conditions at y=yd
c          (see bndyc)
c
c
c *** grid size arguments
c
c
c ... ixp = iparm(6)
c
c      an integer greater than one which is used in
defining the number
c          of grid points in the x direction (see nx =
iparm(10)). "ixp+1"
c          is the number of points on the coarsest x
grid visited during
c          multigrid cycling. ixp should be chosen as
small as possible.
c          recommended values are the small primes 2 or
3.
c          larger values can reduce multigrid
convergence rates considerably,
c          especially if line relaxation in the x
direction is not used.
c          if ixp > 2 then it should be 2 or a small odd
value since a power

```

```

c           of 2 factor of ixp can be removed by
increasing iex = iparm(8)
c           without changing nx = iparm(10) .
c
c
c ... jyq = iparm(7)
c
c           an integer greater than one which is used in
defining the number
c           of grid points in the y direction (see ny =
iparm(11)). "jyq+1"
c           is the number of points on the coarsest y
grid visited during
c           multigrid cycling. jyq should be chosen as
small as possible.
c           recommended values are the small primes 2 or
3.
c           larger values can reduce multigrid
convergence rates considerably,
c           especially if line relaxation in the y
direction is not used.
c           if jyq > 2 then it should be 2 or a small odd
value since a power
c           of 2 factor of jyq can be removed by
increasing jey = iparm(9)
c           without changing ny = iparm(11) .
c
c
c ... iex = iparm(8)
c
c           a positive integer exponent of 2 used in
defining the number
c           of grid points in the x direction (see nx =
iparm(10)) .
c           iex .le. 50 is required. for efficient
multigrid cycling,
c           iex should be chosen as large as possible and
ixp=iparm(8)
c           as small as possible within grid size
constraints when
c           defining nx.
c
c
c ... jey = iparm(9)

```

```

c
c      a positive integer exponent of 2 used in
defining the number
c      of grid points in the y direction (see ny =
iparm(11)).
c      jey .le. 50 is required. for efficient
multigrid cycling,
c      jey should be chosen as large as possible and
jyq=iparm(7)
c      as small as possible within grid size
constraints when
c      defining ny.

c
c
c
c ... nx = iparm(10)
c
c      the number of equally spaced grid points in
the interval [xa,xb]
c      (including the boundaries). nx must have the
form
c
c      nx = ixp*(2** (iex-1)) + 1
c
c      where ixp = iparm(6), iex = iparm(8).
c
c
c ... ny = iparm(11)
c
c      the number of equally spaced grid points in
the interval [yc,yd]
c      (including the boundaries). ny must have the
form:
c
c      ny = jyq*(2** (jey-1)) + 1
c
c      where jyq = iparm(7), jey = iparm(9).
c
c
c *** example
c
c      suppose a solution is wanted on a 33 by 97
grid. then
c      ixp=2, jyq=6 and iex=jey=5 could be used. a

```

```

better
c           choice would be ixp=2, jyq=3, and iex=5,
jey=6.
c
c
c *** note
c
c       let G be the nx by ny fine grid on which the
approximation is
c       generated and let n = max0(iex,jey).  in mudpack,
multigrid
c       cycling is implemented on the ascending chain of
grids
c
c           G(1) < ... < G(k) < ... < G(n) = G.
c
c       each G(k) (k=1,...,n) has mx(k) by my(k) grid
points
c       given by:
c
c           mx(k) = ixp*[2**max0(iex+k-n,1)-1] + 1
c
c           my(k) = jyq*[2**max0(jey+k-n,1)-1] + 1
c
c
c
c ... iguess=iparm(12)
c
c           = 0 if no initial guess to the pde is
provided
c
c           = 1 if an initial guess to the pde is at the
finest grid
c           level is provided in phi (see below)
c
c       comments on iguess = 0 or 1 . . .
c
c       even if iguess = 0, phi must be initialized at all
grid points (this
c       is not checked).  phi can be set to 0.0 at non-
dirchlet grid points
c       if nothing better is available.  the values set in
phi when iguess = 0
c       are passed down and serve as an initial guess to

```

```

the pde at the coarsest
c      grid level where cycling commences.  in this
sense, values input in
c      phi always serve as an initial guess.  setting
iguess = 0 forces full
c      multigrid cycling beginning at the coarsest and
finishing at the finest
c      grid level.
c
c      if iguess = 1 then the values input in phi are an
initial guess to the
c      pde at the finest grid level where cycling begins.
this option should
c      be used only if a "very good" initial guess is
available (as, for
c      example, when restarting from a previous iguess=0
call).
c
c      time dependent problems . . .
c
c *** assume we are solving an elliptic pde every time
step in a
c      marching problem of the form:
c
c      l(p(t)) = r(t)
c
c      where the differential operator "l" has no time
dependence,
c      "p(t)" is the solution and "r(t)" is the right
hand side at
c      current time "t". let "dt" be the increment
between time steps.
c      then p(t) can be used as an initial guess to
p(t+dt) with
c      intl = 1 when solving
c
c      l(p(t+dt)) = r(t+dt).
c
c      after the first two time steps, rather than
continue, it would
c      be better to define the "correction" term:
c
c      e(t,dt) = p(t+dt) - p(t)
c

```

```

c      this clearly satisfies the equation
c
c      l(e(t,dt)) = r(t+dt) - r(t).
c
c      this should be solved with iguess = 0 and intl =
1. boundary
c      conditions for e(t,dt) are obtained from the
boundary conditions
c      for p(t) by subtracting given values at t from
given values at
c      t+dt. for example if
c
c      d(p(t))/dx = f(t), d(p(t+dt))/dx = f(t+dt)
c
c      at some x boundary then e(t,dt) satisfies the
derivative
c      boundary condition
c
c      d(e(t,dt))/dx = f(t+dt) - f(t).
c
c      e(t,dt) can be preset to 0.0 (at nondirichlet
points) or (if p(t-dt)
c      is saved) to p(t)-p(t-dt). with iguess = 0, these
values will serve
c      as an initial guess to e(t,dt) at the coarsest
grid level. this
c      approach has the advantage that a full sequence of
multigrid cycles,
c      beginning at the coarsest grid level, is invoked
every time step in
c      solving for e(t,dt). a few digits of accuracy in
e(t,dt), which is
c      ordinarily much smaller than p(t), will yield
several more digits of
c      accuracy in the final approximation:
c
c      p(t+dt) = p(t) + e(t,dt).
c
c      using this approach to integrate in time will give
more accuracy
c      then using p(t) as an initial guess to p(t+dt) for
all time steps.
c      it does require additional storage.
c

```

```
c      if the differential operator "l" has time
dependence (either thru
c      the coefficients in the pde or the coefficients in
the derivative
c      boundary conditions) then use p(t) as an initial
guess to p(t+dt)
c      when solving
c
c          l(t+dt) (p(t+dt)) = r(t+dt)
c
c      with intl = 0 for all time steps (the
discretization must be repeated
c      for each new "t"). either iguess = 0 (p(t) will
then be an initial
c      guess at the coarsest grid level where cycles will
commence) or
c      iguess = 1 (p(t) will then be an initial guess at
the finest grid
c      level where cycles will remain fixed) can be
tried.
c
c
c ... maxcy = iparm(13)
c
c      the exact number of cycles executed between
the finest (nx by
c      ny) and the coarsest ((ixp+1) by (jyq+1))
grid levels when
c      tolmax=fparm(5)=0.0 (no error control). when
tolmax > 0.0
c      is input (error control) then maxcy is a
limit on the number
c      of cycles between the finest and coarsest
grid levels. in
c      any case, at most maxcy*(iprert+ipost)
relaxation sweeps are
c      are performed at the finest grid level (see
iprer=mgopt(2),
c      ipost=mgopt(3) below). when multigrid
iteration is working
c      "correctly" only a few are required for
convergence. large
c      values for maxcy should not be necessary.
c
```

```

c
c ... method = iparm(14) determines the method of
relaxation
c           (gauss-seidel based on alternating points
or lines)
c
c           = 0 for point relaxation
c
c           = 1 for line relaxation in the x direction
c
c           = 2 for line relaxation in the y direction
c
c           = 3 for line relaxation in both the x and y
direction
c
c
c *** choice of method. . .
c
c     let fx represent the quantity
cabs(cxx(x,y))/dlx**2 over the solution region.
c
c     let fy represent the quantity
cabs(cyy(x,y))/dly**2 over the solution region
c
c     if fx,fy are roughly the same size and do not vary
too much over
c     the solution region choose method = 0. if this
fails try method=3.
c
c     if fx is much greater than fy choose method = 1.
c
c     if fy is much greater than fx choose method = 2
c
c     if neither fx or fy dominates over the solution
region and they
c     both vary considerably choose method = 3.
c
c
c ... length = iparm(15)
c
c           the length of the complex work space provided
in vector work (see below).
c           let isx = 0 if method = 0 or method = 2
c           let isx = 3 if method = 1 or method = 3 and

```

```

nxa.ne.0
c           let isx = 5 if method = 1 or method = 3 and
nxa.eq.0
c           let jsy = 0 if method = 0 or method = 1
c           let jsy = 3 if method = 2 or method = 3 and
nyc.ne.0
c           let jsy = 5 if method = 2 or method = 3 and
nyc.eq.0
c           then . . .
c
c           length =
4*[nx*ny*(10+isx+jsy)+8*(nx+ny+2)]/3
c
c           will suffice in most cases. the exact
minimal work space
c           length required for the current nx,ny and
method is output
c           in iparm(16) (even if iparm(15) is too
small). this will be
c           less then the value given by the simplified
formula above
c           in most cases.

c
c
c ... fparm
c
c           a floating point vector of length 6 used to
efficiently
c           pass floating point arguments. fparm is set
internally
c           in cud2 and defined as follows . . .
c
c
c ... xa=fparm(1), xb=fparm(2)
c
c           the range of the x independent variable. xa
must
c           be less than xb
c
c
c ... yc=fparm(3), yd=fparm(4)
c
c           the range of the y independent variable. yc
must

```

```

c           be less than yd.
c
c
c ... tolmax = fparm(5)
c
c           when input positive, tolmax is a maximum
relative error tolerance
c           used to terminate the relaxation iterations.
assume phi1(i,j)
c           and phi2(i,j) are the last two computed
approximations at all
c           grid points of the finest grid level. if we
define
c
c           phdif = max(cabs(phi2(i,j)-phi1(i,j)))
for all i,j
c
c           and
c
c           phmax = max(cabs(phi2(i,j))) for all i,j
c
c           then "convergence" is considered to have
occurred if and only if
c
c           phdif/phmax < tolmax.
c
c
c           if tolmax=fparm(5)=0.0 is input then there is
no error control
c           and maxcy cycles from the finest grid level
are executed. maxcy
c           is a limit which cannot be exceeded even with
error control.
c           *** calls with tolmax=0.0, when appropriate
because of known
c           convergence behavior, are more efficient than
calls with tolmax
c           positive (i.e., if possible DO NOT use error
control!).
c
c ... work
c
c           a one dimensional complex saved work space
(see iparm(15) for

```

```

c           length) which must be preserved from the
previous call when
c           calling with intl=iparm(1)=1.
c
c ... bndyc
c
c           a subroutine with arguments
(kbdy,xory,alfa,gbdy) which
c           are used to input mixed boundary conditions
to cud2. bndyc
c           must be declared "external" in the program
calling cud2.
c           the boundaries are numbered one thru four and
the mixed
c           derivative boundary conditions are described
below (see the
c           sample driver code "tcud2.f" for an example
of how bndyc is
c           can beset up).
c
c           * * * * * * * * * * * * * * * * y=yd
c           *         kbdy=4      *
c           *                   *
c           *                   *
c           *                   *
c           *         kbdy=1      kbdy=2  *
c           *                   *
c           *                   *
c           *                   *
c           *         kbdy=3      *
c           * * * * * * * * * * * * * * * * y=yc
c
c           x=xa                  x=xb
c
c
c           (1) the kbdy=1 boundary
c
c           this is the edge x=xa where nxa=iparm(2)=2
flags
c           a mixed boundary condition of the form
c
c           dp/dx + alfxa(y)*p(xa,y) = gbdxa(y)
c
c           in this case kbdy=1,xory=y will be input to

```

```

bndyc and
c           alfa, gbdy corresponding to alfxa(y), gbdxa(y)
must be returned.

c
c
c           (2) the kbdy=2 boundary
c
c           this is the edge x=xb where nxb=iparm(3)=2
flags
c           a mixed boundary condition of the form
c
c           dp/dx + alfxb(y) *p(xb,y) = gbdxb(y)
c
c           in this case kbdy=2, xory=y, will be input to
bndyc and
c           alfa, gbdy corresponding to alfxb(y), gbdxb(y)
must be returned.

c
c
c           (3) the kbdy=3 boundary
c
c           this is the edge y=yc where nyc=iparm(4)=2
flags
c           a mixed boundary condition of the form
c
c           dp/dy + alfyc(x) *p(x,yc) = gbdyc(x)
c
c           in this case kbdy=3, xory=x will be input to
bndyc and
c           alfa, gbdy corresponding to alfyc(x), gbdyc(x)
must be returned.

c
c
c           (4) the kbdy=4 boundary
c
c           this is the edge y=yd where nyd=iparm(5)=2
flags
c           a mixed boundary condition of the form
c
c           dp/dy + alfyd(x) *p(x,yd) = gbdyd(x)
c
c           in this case kbdy=4, xory=x will be input to
bndyc and
c           alfa, gbdy corresponding to alfyd(x), gbdyd(x)

```

```
must be returned.  
c  
c  
c *** bndyc must provide the mixed boundary  
condition values  
c           in correspondence with those flagged in  
iparm(2) thru  
c           iparm(5). if all boundaries are specified or  
periodic  
c           cud2 will never call bndyc. even then it  
must be entered  
c           as a dummy subroutine. bndyc must be declared  
"external"  
c           in the routine calling cud2. the actual name  
chosen may  
c           be different.  
alfxa,alfxb,alfyc,alfyd,gbdxa,gbdxb,gbdyc,  
c           gbdyd must all be declared type complex.  
c  
c ... coef  
c  
c           a subroutine with arguments  
(x,y,cxx,cyy,cx,cy,ce) which  
c           provides the known complex coefficients for  
the elliptic pde at  
c           any real grid point (x,y). the name chosen in  
the calling routine  
c           may be different where the coefficient routine  
must be declared  
c           "external."  
c  
c ... rhs  
c  
c           a complex array dimensioned nx by ny which  
contains the given  
c           right hand side values on the uniform 2-d  
mesh.  
c  
c           rhs(i,j) = r(xi,yj) for i=1,...,nx and  
j=1,...,ny  
c  
c ... phi  
c  
c           a complex array dimensioned nx by ny. on
```

```
input phi must contain
c           specified boundary values. for example, if
nyd=iparm(5)=1
c           then phi(i,ny) must be set equal to p(xi,yd)
for i=1,...nx
c           prior to calling cud2. these values are
preserved by cud2.
c           if an initial guess is provided
(iguess=iparm(11)=1) it must
c           be input thru phi.
c
c
c ***      if no initial guess is given (iguess=0) then
phi must still
c           be initialized at all grid points (this is
not checked). these
c           values will serve as an initial guess to the
pde at the coarsest
c           grid level after a transfer from the fine
solution grid. set phi
c           equal to to 0.0 at all internal and non-
specified boundaries
c           grid points if nothing better is available.
c
c
c ... mgopt
c
c           an integer vector of length 4 which allows
the user to select
c           among various multigrid options. if
mgopt(1)=0 is input then
c           a default set of multigrid arguments (chosen
for robustness)
c           will be internally selected and the
remaining values in mgopt
c           will be ignored. if mgopt(1) is nonzero
then the arguments
c           in mgopt are set internally and defined as
follows: (see the
c           basic coarse grid correction algorithm
below)
c
c
c     kcycle = mgopt(1)
```

```

c
c           = 0 if default multigrid options are to be
used
c
c           = 1 if v cycling is to be used (the least
expensive per cycle)
c
c           = 2 if w cycling is to be used (the
default)
c
c           > 2 if more general k cycling is to be used
c           *** warning--values larger than 2 increase
c           the execution time per cycle
considerably and
c           result in the nonfatal error ierror =
-5
c           which indicates inefficient multigrid
cycling.
c
c     iprер = mgopt(2)
c
c           the number of "pre-relaxation" sweeps
executed before the
c           residual is restricted and cycling is
invoked at the next
c           coarser grid level (default value is 2
whenever mgopt(1)=0)
c
c     ipost = mgopt(3)
c
c           the number of "post relaxation" sweeps
executed after cycling
c           has been invoked at the next coarser grid
level and the residual
c           correction has been transferred back
(default value is 1
c           whenever mgopt(1)=0).
c
c *** if iprер, ipost, or (especially) kcycle is greater
than 2
c           than inefficient multigrid cycling has probably
been chosen and
c           the nonfatal error (see below) ierror = -5 will be
set. note

```

```
c      this warning may be overridden by any other
nonzero value
c      for ierror.
c
c      interpol = mgopt(4)
c
c          = 1 if multilinear prolongation
(interpolation) is used to
c          transfer residual corrections and the pde
approximation
c          from coarse to fine grids within full
multigrid cycling.
c
c          = 3 if multicubic prolongation
(interpolation) is used to
c          transfer residual corrections and the pde
approximation
c          from coarse to fine grids within full
multigrid cycling.
c          (this is the default value whenever
mgopt(1)=0).
c
c *** the default values (2,2,1,3) in the vector mgopt
were chosen for
c      robustness. in some cases v(2,1) cycles with
linear prolongation will
c      give good results with less computation
(especially in two-dimensions).
c      this was the default and only choice in an
earlier version of mudpack
c      (see [1]) and can be set with the integer vector
(1,2,1,1) in mgopt.
c
c *** the schedules for one full multigrid cycle
(iguess=0) using v(2,1)
c      cycles and w(2,1) cycles are depicted for a four
level grid below.
c      the number of relaxation sweeps when each grid is
visited are indicated.
c      the "*" stands for prolongation of the full
approximation and the "."
c      stands for transfer of residuals and residual
corrections within the
c      coarse grid correction algorithm (see below). all
```

```

version 5.0.1
c      mudpack solvers use only fully weighted residual
restriction
c
c      one fmg with v(2,1) cycles:
c
c
c      -----
c      level 4
c          * .
c          *
c      -----
c      level 3
c          * .
c          *
c      -----
c      level 2
c          * .
c          *
c      -----
c      level 1
c
c
c      one fmg with w(2,1) cycles:
c
c      -----
c      level 4
c          * .
c          .
c      -----
c      level 3
c          * .
c          .
c      -----
c      level 2
c          * . .
c          . . .
c          . . .
c          . . .
c          . . .
c      -----
c      level 1
c
c
c      the form of the "recursive" coarse grid correction
cycling used
c      when kcyle.ge.0 is input is described below in
pseudo-algorithmic
c      language. it is implemented non-recursively in

```

```

fortran in mudpack.

c
c      algorithm
cgc(k,l(k),u(k),r(k),kcycle,iprer,ipost,iressw,intpol)
c
c *** approximately solve l(k)*u(k) = r(k) using
multigrid iteration
c *** k is the current grid level
c *** l(k) is the discretized pde operator at level k
c *** u(k) is the initial guess at level k
c *** r(k) is the right hand side at level k
c *** i(k,k-1) is the restriction operator from level k
to level k-1
c *** (the form of i(k,k-1) depends on iressw)
c *** i(k-1,k) is the prolongation operator from level
k-1 to level k
c *** (the form of i(k-1,k) depends on intpol)
c
c      begin algorithm cgc
c
c ***     pre-relax at level k
c
c      . do (i=1,iprer)
c
c      . . relax(l(k),u(k),r(k))
c
c      . end do
c
c      . if (k > 1) then
c
c ***      restrict the residual from level k to level k-
1
c
c      . . r(k-1) = i(k,k-1) (r(k)-l(k)*u(k))
c
c      . . kount = 0
c
c      . . repeat
c
c ***      solve for the residual correction at level k-1
in u(k-1)
c ***      using algorithm cgc "kcycle" times (this is
the recursion)
c

```

```

C      . . . kount = kount+1
C
C      . . . invoke cgc(k-1,l(k-1),u(k-1),r(k-
1),kcycle,iprer,ipost,iresw)
C
C
C      . . until (kount.eq.kcycle)
C
C ***      transfer residual correction in u(k-1) to
level k
C ***      with the prolongation operator and add to u(k)
C
C      . . u(k) = u(k) + i(k-1,k) (u(k-1))
C
C      . end if
C
C ***      post relax at level k
C
C      . do (i=1,ipost)
C
C      . . relax(l(k),u(k),r(k))
C
C      . end do
C
C      . return
C
C
C      end algorithm cgc
C
C
C
C ****
C ****
C *** output arguments
C ****
C ****
C ****
C
C ****
C ****
C
C
C ... iparm(16) *** set for intl=0 calls only
C
C          on output iparm(16) contains the actual
complex work space length
C          required. this will usually be less than

```

```

that given by the
c           simplified formula for length=iparm(15) (see
as input argument)
c
c
c ... iparm(17) *** set for intl=1 calls only
c
c           on output iparm(17) contains the actual
number of multigrid cycles
c           between the finest and coarsest grid levels
used to obtain the
c           approximation when error control (tolmax >
0.0) is set.
c
c
c ... fparm(6) *** set for intl=1 calls with fparm(5)
> 0. only
c
c           on output fparm(6) contains the final
computed maximum relative
c           difference between the last two iterates at
the finest grid level.
c           fparm(6) is computed only if there is error
control (tolmax > 0.0)
c           assume phil(i,j,k) and phi2(i,j,k) are the
last two computed
c           values for phi(i,j,k) at all points of the
finest grid level.
c           if we define
c
c           phdif = max(cabs(phi2(i,j)-phil(i,j)))
over all i,j
c
c           and
c
c           phmax = max(cabs(phi2(i,j))) over all i,j
c
c           then
c
c           fparm(6) = phdif/phmax
c
c           is returned whenever phmax > 0.0. in the
degenerate case
c           phmax = 0.0, fparm(6) = phdif is returned.

```

```
C
C
C ... work
C
C           on output the complex space work contains
intermediate values that
C           must not be destroyed if cud2 is to be called
again with intl=1
C
C
C ... phi    *** for intl=1 calls only
C
C           on output phi(i,j) contains the approximation
to p(xi,yj)
C           for all mesh points i = 1,...,nx and
j=1,...,ny.  the last
C           computed iterate in phi is returned even if
convergence is
C           not obtained
C
C
C ... ierror
C
C           For intl=iparm(1)=0 initialization calls,
ierror is an
C           error flag that indicates invalid input
arguments when
C           returned positive and nonfatal warnings when
returned
C           negative. Argument checking and
discretization
C           is bypassed for intl=1 calls which can only
return
C           ierror = -1 or 0 or 1.
C
C
C     non-fatal warnings * * *
C
C
C     ==5 if kcycle=mgopt(1) is greater than 2. values
larger than 2 results
C           in an algorithm which probably does far more
computation than
C           necessary.  kcycle = 1 (v cycles)  or kcycle=2
```

```

(w cycles) should
c           suffice for most problems. ierror = -5 is
also set if either
c           iprер = mgopt(2) or ipost=mgopt(3) is greater
than 2. the
c           ierror=-5 flag is overridden by any other
fatal or non-fatal
c           error.

c
c     =-4 if there are dominant nonzero first order
terms in the pde which
c           make it "hyperbolic" at the finest grid level.
numerically, this
c           happens if:
c
c           cabs(cx)*dlx > 2.*cabs(cxx)      (dlx = (xb-
xa) / (nx-1))
c
c                           (or)
c
c           cabs(cy)*dly > 2.*cabs(cyy)      (dly = (yd-
yc) / (ny-1))
c
c
c           at some fine grid point (xi,yj). if an
adjustment is not made the
c           condition can lead to a matrix coming from the
discretization
c           which is not diagonally dominant and
divergence is possible. since
c           the condition is "likely" at coarser grid
levels for pde's with
c           nonzero first order terms, the adjustments
(actually first order
c           approximations to the pde)

c
c
c           cxx = cmplx(0.5*cabs(cx)*dx,0.0)
c
c           cyy = cmplx(0.5*cabs(cy)*dy,0.0)
c
c
c           (here dx,dy are the x,y mesh sizes of the
subgrid)

```

```

c
c      are made when necessary to preserve
convergence. if made
c      at the finest grid level, it can lead to
convergence to an
c      erroneous solution (flagged by ierror = -4).
a possible remedy
c      is to increase resolution. the ierror = -4
flag overrides the
c      nonfatal ierror = -5 flag.

c
c
c      ==3  if the continuous elliptic pde is singular.
this means the
c      boundary conditions are periodic or pure
derivative at all
c      boundaries and ce(x,y) =(0.0,0.0) for all
x,y. a solution is still
c      attempted but convergence may not occur due
to ill-conditioning
c      of the linear system coming from the
discretization. the
c      ierror = -3 flag overrides the ierror=-4 and
ierror=-5 nonfatal
c      flags.

c
c
c      ==2  if the pde is not elliptic
c
c      real(cxx)*real(cyy).le.0.0 or
aimag(cxx)*aimag(cyy).le.0.0
c
c      in this case a solution is still attempted
although convergence
c      may not occur due to ill-conditioning of the
linear system.
c      the ierror = -2 flag overrides the ierror=-
5,-4,-3 nonfatal
c      flags.

c
c
c      ==1  if convergence to the tolerance specified in
tolmax=fparm(5)>0.
c      is not obtained in maxcy=iparm(13) multigrid

```

```

cycles between the
c           coarsest (ixp+1,jyq+1) and finest (nx,ny)
grid levels.
c           in this case the last computed iterate is
still returned.
c           the ierror = -1 flag overrides all other
nonfatal flags
c
c
c       no errors * * *
c
c       =
c
c       fatal argument errors * * *
c
c       = 1 if intl=iparm(1) is not 0 on initial call or
not 0 or 1
c           on subsequent calls
c
c       = 2 if any of the boundary condition flags
nxa,nxb,nyc,nyd
c           in iparm(2),iparm(3),iparm(4),iparm(5) are not
0,1 or 2
c           or if nxa,nxb or nyc,nyd are not pairwise
zero.
c
c       = 3 if min0(ixp,jyq) < 2 (ixp = iparm(6), jyq =
iparm(7))
c
c       = 4 if min0(iex,jey) < 1 (iex = iparm(8), jey =
iparm(9)) or
c           if max0(iex,jey) > 50
c
c       = 5 if nx.ne.ixp*2** (iex-1)+1 or
ny.ne.jyq*2** (jey-1)+1
c           (nx = iparm(10), ny = iparm(11))
c
c       = 6 if iguess = iparm(12) is not equal to 0 or 1
c
c       = 7 if maxcy = iparm(13) < 1
c
c       = 8 if method = iparm(14) is not 0,1,2, or 3
c
c       = 9 if length = iparm(15) is too small (see

```

```

iparm(16) on output
c           for minimum required work space length)
c
c       =10 if xa > xb or yc > yd
c
c       (xa=fparm(1),xb=fparm(2),yc=fparm(3),yd=fparm(4))
c
c       =11 if tolmax = fparm(5) < 0.0
c
c       errors in setting multigrid options * * * (see
also ierror=-5)
c
c       =12 if kcycle = mgopt(1) < 0 or
c             if iprer = mgopt(2) < 1 or
c             if ipost = mgopt(3) < 1 or
c             if interpol = mgopt(4) is not 1 or 3
c
c
c ****
*
c ****
*
c ****
c       end of cud2 documentation
c
c
c ****
**
c ****
**
c

```

```
C  
C      file cud24.d  
C  
C      * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C      *  
*  
C      *          copyright (c) 2008 by UCAR  
*  
C      *  
*  
C      *          University Corporation for Atmospheric  
Research      *  
C      *  
*  
C      *          all rights reserved  
*  
C      *  
*  
C      *          MUDPACK version 5.0.1  
*  
C      *  
*  
C      *          A Fortran Package of Multigrid  
*  
C      *  
*  
C      *          Subroutines and Example Programs  
*  
C      *  
*  
C      *          for Solving Elliptic Partial Differential  
Equations      *  
C      *  
*  
C      *          by  
*  
C      *  
*  
C      *          John Adams  
*  
C      *  
*
```

```
C      *
      of
*
C      *
*
C      *           the National Center for Atmospheric
Research          *
C      *
*
C      *           Boulder, Colorado (80307)
U.S.A.          *
C      *
*
C      *           which is sponsored by
*
C      *
*
C      *           the National Science Foundation
*
C      *
*
C      *           ****
* * * * *
C
C ... file cud24.d
C
C     contains documentation for subroutine
cud24(work,phi,ierror)
C     A sample fortran driver is file "tcud24.f".
C
C ... required MUDPACK files
C
C     cud2.f, cudcom.f
C
C ... purpose
C
C     cud24 attempts to improve the estimate in phi,
obtained by calling
C     cud2, from second to fourth order accuracy. see
the file "cud2.d"
C     for a detailed discussion of the complex elliptic
pde approximated and
C     arguments "work,phi" which are also part of the
argument list for
C     cud2.
```

```
C
C ... assumptions
C
C      * phi contains a second-order approximation from
an earlier cud2 call
C
C      * arguments "work,phi" are the same used in
calling cud2
C
C      * "work,phi" have not changed since the last call
to cud2
C
C      * the finest grid level contains at least 6
points in each direction
C
C
C *** warning
C
C      if the first assumption is not true then a fourth
order approximation
C      cannot be produced in phi. the last assumption
(adequate grid size)
C      is the only one checked. failure in any of the
others can result in
C      in an undetectable error.
C
C ... language
C
C      fortran90/fortran77
C
C ... portability
C
C      mudpack5.0.1 software has been compiled and tested
with fortran77
C      and fortran90 on a variety of platforms.
C
C ... methods
C
C      details of the methods employeed by the solvers in
mudpack are given
C      in [1,9]. [1,2,9] contain performance
measurements on a variety of
C      elliptic pdes (see "references" in the file
"readme"). in summary:
```

```
c
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
estimate (i.e.,
c      discretization level error is reached) then this
can be improved to a
c      fourth-order estimate using the technique of
"deferred corrections"
c      the values in the solution array are used to
generate a fourth-order
c      approximation to the truncation error. second-
order finite difference
c      formula are used to approximate third and fourth
partial derivatives
c      of the solution function [3]. the truncation
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side. the default
multigrid options
c      are used to compute the fourth-order correction
term which is added
c      to the original solution array.

c
c
c ... references (partial)
c
c
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
c      Solution of Linear Elliptic Partial Differential
Equations,"
c      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
c
c      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
c      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
c      1989, pp.1-12.
c      .
```

```
C .
C .
C      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
Haidvogel, and V. Pizzo,
C      "Applications of Multigrid Software in the
Atmospheric Sciences,"
C      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-
1458.
C .
C .
C .
C      [9] J. Adams, "Recent Enhancements in MUDPACK, a
Multigrid Software
C      package for Elliptic Partial Differential
Equations," Applied Math.
C      and Comp., 1991, vol. 43, May 1991, pp. 79-94.
C
C      [10] J. Adams, "MUDPACK-2: Multigrid Software for
Approximating
C      Elliptic Partial Differential Equations on Uniform
Grids with
C      any Resolution," Applied Math. and Comp., 1993,
vol. 53, February
C      1993, pp. 235-249
C .
C .
C .
C
C ... error argument
C
C      = 0 if no error is detected
C
C      = 30 if min0(nx,ny) < 6 where nx,ny are the fine
grid sizes
C          in the x,y directions.
C
C
C
*****
*****
```

```
C      end of cud24 documentation
C
C
*****
*****
```

```
C
```

```
*****
```

```
C
```

```
*****
```

```
*****
```

```
C
```

---

## CUD24CR

```
C
C      file cud24cr.d
C
C      * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *           copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *           all rights reserved
*
C      *
*
C      *           MUDPACK version 5.0.1
*
C      *
*
C      *           A Fortran Package of Multigrid
*
C      *
```

```
*  
C      *          Subroutines and Example Programs  
*  
C      *  
*  
C      *      for Solving Elliptic Partial Differential  
Equations      *  
C      *  
*  
C      *          by  
*  
C      *  
*  
C      *          John Adams  
*  
C      *  
*  
C      *          of  
*  
C      *  
*  
C      *          the National Center for Atmospheric  
Research      *  
C      *  
*  
C      *          Boulder, Colorado (80307)  
U.S.A.      *  
C      *  
*  
C      *          which is sponsored by  
*  
C      *  
*  
C      *          the National Science Foundation  
*  
C      *  
*  
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C  
C ... file cud24cr.d  
C  
C     contains documentation for:  
C     subroutine cud24cr(work,coef,bndyc,phi,ierror)
```

```
c      A sample fortran driver is file "tcud24cr.f".
c
c ... required MUDPACK files
c
c      cud2cr.f, cudcom.f
c
c ... purpose
c
c      cud24cr attempts to improve the estimate in phi,
obtained by calling
c      cud2cr, from second to fourth order accuracy.
see the file "cud2cr.d"
c      for a detailed discussion of the elliptic pde
approximated and
c      arguments "work,coef,bndyc,phi" which are also
part of the argument
c      list for cud2cr
c
c ... assumptions
c
c      * phi contains a second-order approximation from
an earlier cud2cr call
c
c      * arguments "work,coef,bndyc,phi" are the same
used in calling cud2cr
c
c      * "work,coef,bndyc,phi" have not changed since
the last call to cud2cr
c
c      * the finest grid level contains at least 6
points in each direction
c
c
c *** warning
c
c      if the first assumption is not true then a fourth
order approximation
c      cannot be produced in phi. the last assumption
(adequate grid size)
c      is the only one checked. failure in any of the
others can result in
c      in an undetectable error.
c
c ... language
```

```
c
c      fortran90/fortran77
c
c ... portability
c
c      mudpack5.0.1 software has been compiled and tested
with fortran77
c      and fortran90 on a variety of platforms.
c
c ... methods
c
c      details of the methods employed by the solvers in
mudpack are given
c      in [1,9]. [1,2,9] contain performance
measurements on a variety of
c      elliptic pdes (see "references" in the file
"readme"). in summary:
c
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
estimate (i.e.,
c      discretization level error is reached) then this
can be improved to a
c      fourth-order estimate using the technique of
"deferred corrections"
c      the values in the solution array are used to
generate a fourth-order
c      approximation to the truncation error. second-
order finite difference
c      formula are used to approximate third and fourth
partial derivatives
c      of the solution function [3]. the truncation
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side. the default
multigrid options
c      are used to compute the fourth-order correction
term which is added
c      to the original solution array.
c
```

```
C
C ... references (partial)
C
C
C      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
C      Solution of Linear Elliptic Partial Differential
Equations,"
C      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
C
C      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
C      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
C      1989, pp.1-12.
C      .
C      .
C      .
C      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
Haidvogel, and V. Pizzo,
C      "Applications of Multigrid Software in the
Atmospheric Sciences,"
C      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-
1458.
C      .
C      .
C      .
C      [9] J. Adams, "Recent Enhancements in MUDPACK, a
Multigrid Software
C      package for Elliptic Partial Differential
Equations," Applied Math.
C      and Comp., 1991, vol. 43, May 1991, pp. 79-94.
C
C      [10]J. Adams, "MUDPACK-2: Multigrid Software for
Approximating
C      Elliptic Partial Differential Equations on Uniform
Grids with
C      any Resolution," Applied Math. and Comp., 1993,
vol. 53, February
C      1993, pp. 235-249
C      .
C      .
C      .
```

```
c ... error argument
c
c      = 0 if no error is detected
c
c      = 30 if min0(nx,ny) < 6 where nx,ny are the fine
grid sizes
c          in the x,y directions.
c
c
c
*****
*****
```

C

```
*****
```

C

```
*****
```

C

```
*****
```

C

```
*****
```

C end of cud24cr documentation

C

```
*****
```

CUD24SP

```
C  
C      file cud24sp.d  
C  
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * * *  
C      *  
*  
C      *          copyright (c) 2008 by UCAR  
*
```

```
C      *
*
C      *      University Corporation for Atmospheric
Research          *
C      *
*
C      *                  all rights reserved
*
C      *
*
C      *                  MUDPACK  version 5.0.1
*
C      *
*
C      *                  A Fortran Package of Multigrid
*
C      *
*
C      *                  Subroutines and Example Programs
*
C      *
*
C      *      for Solving Elliptic Partial Differential
Equations          *
C      *
*
C      *                  by
*
C      *
*
C      *                  John Adams
*
C      *
*
C      *                  of
*
C      *
*
C      *                  the National Center for Atmospheric
Research          *
C      *
*
C      *                  Boulder, Colorado (80307)
U.S.A.          *
```

```
C      *
*
C      *                               which is sponsored by
*
C      *
*
C      *                               the National Science Foundation
*
C      *
*
C      * * * * * * * * * * * * * * * *
* * * * * * * *
C
C ... file cud24sp.d
C
C      contains documentation for subroutine
cud24sp(work,phi,ierror)
C      A sample fortran driver is file "tcud24sp.f".
C
C ... required MUDPACK files
C
C      cud2sp.f, cudcom.f
C
C ... purpose
C
C      cud24sp attempts to improve the estimate in phi,
obtained by calling
C      cud2sp, from second to fourth order accuracy.
see the file "cud2sp.d"
C      for a detailed discussion of the complex elliptic
pde approximated and
C      arguments "work,phi" which are also part of the
argument list for
C      cud2sp.
C
C ... assumptions
C
C      * phi contains a second-order approximation from
an earlier cud2sp call
C
C      * arguments "work,phi" are the same used in
calling cud2sp
C
C      * "work,phi" have not changed since the last call
```

```
to cud2sp
c
c      * the finest grid level contains at least 6
points in each direction
c
c
c *** warning
c
c      if the first assumption is not true then a fourth
order approximation
c      cannot be produced in phi. the last assumption
(adequate grid size)
c      is the only one checked. failure in any of the
others can result in
c      in an undetectable error.
c
c ... language
c
c      fortran90/fortran77
c
c ... portability
c
c      mudpack5.0.1 software has been compiled and tested
with fortran77
c      and fortran90 on a variety of platforms.
c
c ... methods
c
c      details of the methods employed by the solvers in
mudpack are given
c      in [1,9]. [1,2,9] contain performance
measurements on a variety of
c      elliptic pdes (see "references" in the file
"readme"). in summary:
c
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
estimate (i.e.,
c      discretization level error is reached) then this
can be improved to a
c      fourth-order estimate using the technique of
```

```
"deferred corrections"
c      the values in the solution array are used to
generate a fourth-order
c      approximation to the truncation error. second-
order finite difference
c      formula are used to approximate third and fourth
partial derivatives
c      of the solution function [3]. the truncation
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side. the default
multigrid options
c      are used to compute the fourth-order correction
term which is added
c      to the original solution array.

c
c
c ... references (partial)

c
c
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
c      Solution of Linear Elliptic Partial Differential
Equations,"
c      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.

c
c      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
c      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
c      1989, pp.1-12.

c      .
c      .
c      .

c      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
Haidvogel, and V. Pizzo,
c      "Applications of Multigrid Software in the
Atmospheric Sciences,"
c      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-
1458.

c      .
c      .
```

```
C .
C      [9] J. Adams, "Recent Enhancements in MUDPACK, a
Multigrid Software
C      package for Elliptic Partial Differential
Equations," Applied Math.
C      and Comp., 1991, vol. 43, May 1991, pp. 79-94.
C
C      [10]J. Adams, "MUDPACK-2: Multigrid Software for
Approximating
C      Elliptic Partial Differential Equations on Uniform
Grids with
C      any Resolution," Applied Math. and Comp., 1993,
vol. 53, February
C      1993, pp. 235-249
C .
C .
C .
C ...
C ... error argument
C
C      = 0 if no error is detected
C
C      = 30 if min0(nx,ny) < 6 where nx,ny are the fine
grid sizes
C          in the x,y directions.
C
C
C ****
*****
C ****
*****
C
C ****
*****
C
C      end of cud24sp documentation
C
C ****
*****
C ****
*****
C
C ****
*****
```

CUD2CR

```
c      file cud2cr.d
c
c  . . . . .
c  .
c  .
c  .
c  .           copyright (c) 2008 by UCAR
c  .
c  .
c  .
c  .       UNIVERSITY CORPORATION for ATMOSPHERIC
RESEARCH   .
c  .
c  .
c  .           all rights reserved
c  .
c  .
c  .
c  .
c  .           MUDPACK version 5.0.1
c  .
c  .
c  .
c  . . . . .
c  .
c  .
c  .
c  ...
c
c
c  ... file cud2cr.d
c
c      contains documentation for the complex mudpack
solver:
c      subroutine
cud2cr(iparm,fparm,work,coef,bndyc,rhs,phi,mgopt,ierror)
c      a sample fortran driver is file "tcud2cr.f".
c
```

```

c ... required mudpack files
c
c      cudcom.f
c
c ... purpose
c
c      subroutine cud2cr automatically discretizes and
attempts to compute
c      the second-order difference approximation to the
complex 2-D
c      linear nonseparable elliptic partial differential
equation with cross
c      derivative term on a rectangle.  the approximation
is generated on a
c      uniform grid covering the rectangle (see mesh
description below).
c      boundary conditions may be specified (dirchlet),
periodic, or mixed
c      oblique derivative (see bndyc) in any combination.
the form of the pde
c      approximated is:
c
c
c      cxx(x,y)*pxx + cxy(x,y)*pxy + cyy(x,y)*pyy +
cx(x,y)*px +
c
c      cy(x,y)*py + ce(x,y)*p(x,y) = r(x,y).
c
c
c      pxx,pxy,pyy,px,py are second and first partial
derivatives of the
c      unknown complex solution function p(x,y) with
respect to the
c      independent variables x,y.  cxx,cxy,cyy,cx,cy,ce
are the known
c      complex coefficients of the elliptic pde and
r(x,y) is the known
c      complex right hand side of the equation.  The real
parts of cxx,cyy
c      or the imaginary parts of cxx,cyy should be
positive for all x,y
c      in the solution region (see ierror=-2).
Nonseparability means some
c      of the coefficients depend on both x and y.  if

```

```
the PDE is separable
c      and cxy = 0 then subroutine cud2sp should be used.
c
c
c ... mesh description . . .
c
c      the approximation is generated on a uniform nx by
ny grid.  the grid
c      is superimposed on the rectangular solution region
c
c            [xa,xb] x [yc,yd].
c
c      let
c
c            dlx = (xb-xa) / (nx-1), dly = (yd-yc) / (ny-1)
c
c      be the uniform grid increments in the x,y
directions. then
c
c            xi=xa+(i-1)*dlx,   yj=yc+(j-1)*dly
c
c      for i=1,...,nx and j=1,...,ny denote the x,y
uniform mesh points
c
c
c ... language
c
c      fortran90/fortran77
c
c
c ... portability
c
c      mudpack5.0.1 software has been compiled and tested
with fortran77
c      and fortran90 on a variety of platforms.
c
c ... methods
c
c      details of the methods employeed by the solvers in
mudpack are given
c      in [1,9].  [1,2,9] contain performance
measurements on a variety of
c      elliptic pdes (see "references" in the file
"readme").  in summary:
```

```
c
c *** discretization and solution (second-order solvers)
(see [1])
c
c      the pde and boundary conditions are automatically
discretized at all
c      grid levels using second-order finite difference
formula. diagonal
c      dominance at coarser grid levels is maintained in
the presence of
c      nonzero first-order terms by adjusting the second-
order coefficient
c      when necessary. the resulting block tri-diagonal
linear system is
c      approximated using multigrid iteration
[10,11,13,15,16,18]. version
c      5.0.1 of mudpack uses only fully weighted residual
restriction. defaults
c      include cubic prolongation and w(2,1) cycles.
these can be overridden
c      with selected multigrid options (see "mgopt"). .
error control based on
c      maximum relative differences is available. full
multigrid cycling (fmg)
c      or cycling beginning or restarting at the finest
grid level can be
c      selected. a menu of relaxation methods including
gauss-seidel point,
c      line relaxation(s) (in any combination of
directions) and planar
c      relaxation (for three-dimensional anisotropic
problems) are provided.
c      all methods use ordering based on alternating
points (red/black),
c      lines, or planes for cray vectorization and
improved convergence
c      rates [14].
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
estimate (i.e.,
c      discretization level error is reached) then this
```

```
can be improved to a
c      fourth-order estimate using the technique of
"deferred corrections."
c      the values in the solution array are used to
generate a fourth-order
c      approximation to the truncation error. second-
order finite difference
c      formula are used to approximate third and fourth
partial derivatives
c      of the solution function [3]. the truncation
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side. the default
multigrid options
c      are used to compute the fourth-order correction
term which is added
c      to the original solution array.

c
c
c ... references (partial)
c
c
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
c      Solution of Linear Elliptic Partial Differential
Equations,"
c      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
c
c      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
c      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
c      1989, pp.1-12.
c      .
c      .
c      .
c      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
Haidvogel, and V. Pizzo,
c      "Applications of Multigrid Software in the
Atmospheric Sciences,"
c      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-
1458.
```

```
C .
C .
C .
C     [9] J. Adams, "Recent Enhancements in MUDPACK, a
Multigrid Software
C     package for Elliptic Partial Differential
Equations," Applied Math.
C     and Comp., 1991, vol. 43, May 1991, pp. 79-94.
C
C     [10]J. Adams, "MUDPACK-2: Multigrid Software for
Approximating
C     Elliptic Partial Differential Equations on Uniform
Grids with
C     any Resolution," Applied Math. and Comp., 1993,
vol. 53, February
C     1993, pp. 235-249
C .
C .
C .
C
C ... argument description
C
C
C
*****
*****  

*****  

C *** input arguments
*****  

*****  

C
*****
*****  

*****  

C
C
C ... iparm
C
C     an integer vector of length 17 used to pass
integer
C     arguments. iparm is set internally and
defined as
C         follows:
C
C
C ... intl=iparm(1)
C
```

```
c      an initialization argument.  intl=0  must be
input
c      on an initial call.  in this case input
arguments will
c      be checked for errors and the elliptic
partial differential
c      equation and boundary conditions will be
discretized using
c      second order finite difference formula.
c
c ***      an approximation is not generated after an
intl=0 call!
c      cud2cr should be called with intl=1 to
approximate the elliptic
c      pde discretized by the intl=0 call.  intl=1
should also
c      be input if cud2cr has been called earlier
and only the
c      values in in rhs (see below) or gbdy (see
bndyc below)
c      or phi (see below) have changed.  this will
bypass
c      redundant pde discretization and argument
checking
c      and save computational time.  some examples
of when
c      intl=1 calls should be used are:
c
c      (0) after a intl=0 argument checking and
discretization call
c
c      (1) cud2cr is being recalled for additional
accuracy.  in
c      this case iguess=iparm(12)=1 should also
be used.
c
c      (2) cud2cr is being called every time step in
a time dependent
c      problem (see discussion below) where the
elliptic operator
c      does not depend on time.
c
c      (3) cud2cr is being used to solve the same
elliptic equation
```

c for several different right hand sides  
(iguess=0 should  
c probably be used for each new righthand  
side).  
c  
c intl = 0 must be input before calling with  
intl = 1 when any  
c of the following conditions hold:  
c  
c (a) the initial call to cud2cr  
c (b) any of the integer arguments other than  
iguess=iparm(12)  
c or maxcy=iparm(13) or mgopt have changed  
since the previous  
c call.  
c  
c (c) any of the floating point arguments other  
than tolmax=  
c fparm(5) have changed since the previous  
call  
c  
c (d) any of the coefficients input by coef  
(see below) have  
c changed since the previous call  
c  
c (e) any of the "alfa" coefficients input by  
bndyc (see below)  
c have changed since the previous call.  
c  
c if any of (a) through (e) are true then the  
elliptic pde  
c must be discretized or redisccretized. if  
none of (a)  
c through (e) holds, calls can be made with  
intl=1.  
c incorrect calls with intl=1 will produce  
erroneous results.  
c \*\*\* the values set in the saved work space "work"  
(see below) with  
c an intl=0 call must be preserved with  
subsequent intl=1 calls.  
c  
c MUDPACK software performance should be

```

monitored for intl=1
c           calls.  The intl=0 discretization call
performance depends
c           primarily on the efficiency or lack of
efficiency of the
c           user provided subroutines for pde
coefficients and
c           boundary conditions.
c
c
c ... nxa=iparm(2)
c
c           flags boundary conditions on the edge x=xa
c
c           = 0 if p(x,y) is periodic in x on [xa,xb]
c           (i.e., p(x+xb-xa,y) = p(x,y) for all x,y
c           (if nxa=0 then nxb=0 is required, see
ierror = 2)
c
c           = 1 if p(xa,y) is specified (this must be input
thru phi(1,j))
c
c           = 2 if there are mixed derivative boundary
conditions at x=xa
c           (see bndyc)
c
c
c ... nxb=iparm(3)
c
c           flags boundary conditions on the edge x=xb
c
c           = 0 if p(x,y) is periodic in x on [xa,xb]
c           (i.e., p(x+xb-xa,y) = p(x,y) for all x,y
c           (if nxb=0 then nxa=0 is required, see
ierror = 2)
c
c           = 1 if p(xb,y) is specified (this must be input
thru phi(nx,j))
c
c           = 2 if there are mixed derivative boundary
conditions at x=xb
c           (see bndyc)
c
c

```

```

c ... nyc=iparm(4)
c
c           flags boundary conditions on the edge y=yc
c
c           = 0 if p(x,y) is periodic in y on [yc,yd]
c               (i.e., p(x,y+yd-yc) = p(x,y) for all x,y
c               (if nyc=0 then nyd=0 is required, see
ierror = 2)
c
c           = 1 if p(x,yc) is specified (this must be input
thru phi(i,1))
c
c           = 2 if there are mixed derivative boundary
conditions at y=yc
c               (see bndyc)
c
c
c ... nyd=iparm(5)
c
c           flags boundary conditions on the edge y=yd
c
c           = 0 if p(x,y) is periodic in y on [yc,yd]
c               (i.e., p(x,y+yd-yc) = p(x,y) for all x,y
c               (if nyd=0 then nyc=0 is required, see
ierror = 2)
c
c           = 1 if p(x,yd) is specified (this must be input
thru phi(i,ny))
c
c           = 2 if there are mixed derivative boundary
conditions at y=yd
c               (see bndyc)
c
c
c *** grid size arguments
c
c
c ... ixp = iparm(6)
c
c           an integer greater than one which is used in
defining the number
c           of grid points in the x direction (see nx =
iparm(10)). "ixp+1"
c           is the number of points on the coarsest x

```

```
grid visited during
c           multigrid cycling.  ixp should be chosen as
small as possible.
c           recommended values are the small primes 2 or
3.
c           larger values can reduce multigrid
convergence rates considerably,
c           especially if line relaxation in the x
direction is not used.
c           if ixp > 2 then it should be 2 or a small odd
value since a power
c           of 2 factor of ixp can be removed by
increasing iex = iparm(8)
c           without changing nx = iparm(10).
c
c
c ... jyq = iparm(7)
c
c           an integer greater than one which is used in
defining the number
c           of grid points in the y direction (see ny =
iparm(11)). "jyq+1"
c           is the number of points on the coarsest y
grid visited during
c           multigrid cycling.  jyq should be chosen as
small as possible.
c           recommended values are the small primes 2 or
3.
c           larger values can reduce multigrid
convergence rates considerably,
c           especially if line relaxation in the y
direction is not used.
c           if jyq > 2 then it should be 2 or a small odd
value since a power
c           of 2 factor of jyq can be removed by
increasing jey = iparm(9)
c           without changing ny = iparm(11).
c
c
c ... iex = iparm(8)
c
c           a positive integer exponent of 2 used in
defining the number
c           of grid points in the x direction (see nx =
```

```

iparm(10)).
c           iex .le. 50 is required.  for efficient
multigrid cycling,
c           iex should be chosen as large as possible and
ixp=iparm(8)
c           as small as possible within grid size
constraints when
c           defining nx.
c
c
c ... jey = iparm(9)
c
c           a positive integer exponent of 2 used in
defining the number
c           of grid points in the y direction (see ny =
iparm(11)).
c           jey .le. 50 is required.  for efficient
multigrid cycling,
c           jey should be chosen as large as possible and
jyq=iparm(7)
c           as small as possible within grid size
constraints when
c           defining ny.
c
c
c
c ... nx = iparm(10)
c
c           the number of equally spaced grid points in
the interval [xa,xb]
c           (including the boundaries).  nx must have the
form
c
c           nx = ixp*(2** (iex-1)) + 1
c
c           where ixp = iparm(6), iex = iparm(8).
c
c
c ... ny = iparm(11)
c
c           the number of equally spaced grid points in
the interval [yc,yd]
c           (including the boundaries).  ny must have the
form:

```

```

C
C           ny = jyq* (2** (jey-1)) + 1
C
C           where jyq = iparm(7), jey = iparm(9).
C
C
C *** example
C
C           suppose a solution is wanted on a 33 by 97
grid. then
C           ixp=2, jyq=6 and iex=jey=5 could be used. a
better
C           choice would be ixp=2, jyq=3, and iex=5,
jey=6.
C
C *** note
C
C           let g be the nx by ny fine grid on which the
approximation is
C           generated and let n = max0(iex,jey). in mudpack,
multigrid
C           cycling is implemented on the ascending chain of
grids
C
C           g(1) < ... < g(k) < ... < g(n) = g.
C
C           each g(k) (k=1,...,n) has mx(k) by my(k) grid
points
C           given by:
C
C           mx(k) = ixp* [2** (max0(iex+k-n,1)-1)] + 1
C
C           my(k) = jyq* [2** (max0(jey+k-n,1)-1)] + 1
C
C
C
C ... iguess=iparm(12)
C
C           = 0 if no initial guess to the pde is
provided
C
C           = 1 if an initial guess to the pde is at the
finest grid
C           level is provided in phi (see below)

```

```

c
c      comments on iguess = 0 or 1 . . .
c
c      even if iguess = 0, phi must be initialized at all
grid points (this
c      is not checked).  phi can be set to 0.0 at non-
dirchlet grid points
c      if nothing better is available.  the values set in
phi when iguess = 0
c      are passed down and serve as an initial guess to
the pde at the coarsest
c      grid level where cycling commences.  in this
sense, values input in
c      phi always serve as an initial guess.  setting
iguess = 0 forces full
c      multigrid cycling beginning at the coarsest and
finishing at the finest
c      grid level.
c
c      if iguess = 1 then the values input in phi are an
initial guess to the
c      pde at the finest grid level where cycling begins.
this option should
c      be used only if a "very good" initial guess is
available (as, for
c      example, when restarting from a previous iguess=0
call).
c
c      time dependent problems . . .
c
c *** assume we are solving an elliptic pde every time
step in a
c      marching problem of the form:
c
c          l(p(t)) = r(t)
c
c      where the differential operator "l" has no time
dependence,
c      "p(t)" is the solution and "r(t)" is the right
hand side at
c      current time "t". let "dt" be the increment
between time steps.
c      then p(t) can be used as an initial guess to
p(t+dt) with

```

```

c      intl = 1 when solving
c
c      l(p(t+dt)) = r(t+dt) .
c
c      after the first two time steps, rather than
c continue, it would
c      be better to define the "correction" term:
c
c      e(t,dt) = p(t+dt) - p(t)
c
c      this clearly satisfies the equation
c
c      l(e(t,dt)) = r(t+dt) - r(t) .
c
c      this should be solved with iguess = 0 and intl =
1. boundary
c      conditions for e(t,dt) are obtained from the
boundary conditions
c      for p(t) by subtracting given values at t from
given values at
c      t+dt. for example if
c
c      d(p(t))/dx = f(t), d(p(t+dt))/dx = f(t+dt)
c
c      at some x boundary then e(t,dt) satisfies the
derivative
c      boundary condition
c
c      d(e(t,dt))/dx = f(t+dt) - f(t) .
c
c      e(t,dt) can be preset to 0.0 (at nondirichlet
points) or (if p(t-dt)
c      is saved) to p(t)-p(t-dt). with iguess = 0, these
values will serve
c      as an initial guess to e(t,dt) at the coarsest
grid level. this
c      approach has the advantage that a full sequence of
multigrid cycles,
c      beginning at the coarsest grid level, is invoked
every time step in
c      solving for e(t,dt). a few digits of accuracy in
e(t,dt), which is
c      ordinarily much smaller than p(t), will yield
several more digits of

```

```

c      accuracy in the final approximation:
c
c      p(t+dt) = p(t) + e(t,dt).
c
c      using this approach to integrate in time will give
more accuracy
c      then using p(t) as an initial guess to p(t+dt) for
all time steps.
c      it does require additional storage.
c
c      if the differential operator "l" has time
dependence (either thru
c      the coefficients in the pde or the coefficients in
the derivative
c      boundary conditions) then use p(t) as an initial
guess to p(t+dt)
c      when solving
c
c      l(t+dt) (p(t+dt)) = r(t+dt)
c
c      with intl = 0 for all time steps (the
discretization must be repeated
c      for each new "t"). either iguess = 0 (p(t) will
then be an initial
c      guess at the coarsest grid level where cycles will
commence) or
c      iguess = 1 (p(t) will then be an initial guess at
the finest grid
c      level where cycles will remain fixed) can be
tried.
c
c
c ... maxcy = iparm(13)
c
c      the exact number of cycles executed between
the finest (nx by
c      ny) and the coarsest ((ixp+1) by (jyq+1))
grid levels when
c      tolmax=fparm(5)=0.0 (no error control). when
tolmax > 0.0
c      is input (error control) then maxcy is a
limit on the number
c      of cycles between the finest and coarsest
grid levels. in

```

```

c      any case, at most maxcy*(iprter+ipost)
relaxation sweeps are
c      are performed at the finest grid level (see
iprter=mgopt(2),
c      ipost=mgopt(3) below). when multigrid
iteration is working
c      "correctly" only a few are required for
convergence. large
c      values for maxcy should not be necessary.
c
c
c ... method = iparm(14) determines the method of
relaxation
c      (gauss-seidel based on alternating points
or lines)
c
c      = 0 for point relaxation
c
c      = 1 for line relaxation in the x direction
c
c      = 2 for line relaxation in the y direction
c
c      = 3 for line relaxation in both the x and y
direction
c
c
c *** choice of method. . .
c
c      let fx represent the quantity cxx(x,y)/dlx**2 over
the solution region.
c
c      let fy represent the quantity cyy(x,y)/dly**2 over
the solution region
c
c      if fx,fy are roughly the same size and do not vary
too much over
c      the solution region choose method = 0. if this
fails try method=3.
c
c      if fx is much greater than fy choose method = 1.
c
c      if fy is much greater than fx choose method = 2
c
c      if neither fx or fy dominates over the solution

```

```
region and they
c      both vary considerably choose method = 3.
c
c
c ... length = iparm(15)
c
c           the length of the work space provided in
complex work
c           space "work")
c
c           let isx = 0 if method = 0 or method = 2
c           let isx = 3 if method = 1 or method = 3 and
nxa.ne.0
c           let isx = 5 if method = 1 or method = 3 and
nxa.eq.0
c           let jsy = 0 if method = 0 or method = 1
c           let jsy = 3 if method = 2 or method = 3 and
nyc.ne.0
c           let jsy = 5 if method = 2 or method = 3 and
nyc.eq.0
c           then . . .
c
c           length =
[7* (nx+2) * (ny+2)+4* (11+isx+jsy) *nx*ny]/3
c
c           will suffice in most cases.  the exact
minimal work space
c           length required for the current nx,ny and
method is output
c           in iparm(16) (even if iparm(15) is too
small).  this will be
c           less then the value given by the simplified
formula above
c           in most cases.
c
c
c ... fparm
c
c           a floating point vector of length 6 used to
efficiently
c           pass floating point arguments.  fparm is set
internally
c           in cud2cr and defined as follows . . .
c
```

```
C
C ... xa=fparm(1), xb=fparm(2)
C
C           the range of the x independent variable. xa
must
C           be less than xb
C
C
C ... yc=fparm(3), yd=fparm(4)
C
C           the range of the y independent variable. yc
must
C           be less than yd.
C
C
C ... tolmax = fparm(5)
C
C           when input positive, tolmax is a maximum
relative error tolerance
C           used to terminate the relaxation iterations.
assume phi1(i,j)
C           and phi2(i,j) are the last two computed
approximations at all
C           grid points of the finest grid level. if we
define
C
C           phdif = max(abs(phi2(i,j)-phi1(i,j))) for
all i,j
C
C           and
C
C           phmax = max(abs(phi2(i,j))) for all i,j
C
C           then "convergence" is considered to have
occurred if and only if
C
C           phdif/phmax < tolmax.
C
C
C           if tolmax=fparm(5)=0.0 is input then there is
no error control
C           and maxcy cycles from the finest grid level
are executed. maxcy
C           is a limit which cannot be exceeded even with
```

```

error control.
c      *** calls with tolmax=0.0, when appropriate
because of known
c      convergence behavior, are more efficient than
calls with tolmax
c      positive (i.e., if possible do not use error
control!).
c
c ... work
c
c      a complex saved work space (see iparm(15) for
size) which
c      must be preserved from the previous call when
calling with
c      intl=iparm(1)=1.
c
c ... bndyc
c
c      a subroutine with arguments
(kbdy,xory,alfa,beta,gama,gbdy) which
c      are used to input mixed boundary conditions
to cud2cr. bndyc
c      must be declared "external" in the program
calling cud2cr. kbdy
c      is type integer, xory real, and
alfa,beta,gama,gbdy type complex.
c      the boundaries are numbered one thru four and
the mixed
c      derivative boundary conditions are described
below (see the
c      sample driver code "tcud2cr.f" for an example
of how bndyc is
c      can beset up).

c      * * * * * * * * * * * * * * * y=yd
c      *      kbdy=4      *
c      *
c      *
c      *
c      *
c      kbdy=1      kbdy=2      *
c      *
c      *
c      *
c      *
c      kbdy=3      *

```

```

* * * * * * * * * * * * * * * * y=yc
c
c           x=xa                      x=xb
c
c
c   (1)   the kbdy=1 boundary
c
c   this is the edge x=xa where nxa=iparm(2) = 2
flags
c   a mixed boundary condition of the form
c
c           alfxa(y)*px + betxa(y)*py +
gamxa(y)*p(xa,y) = gbdxa(y)
c
c   in this case kbdy=1,xory=y will be input to
bndyc and
c   alfa,beta,gama,gbody corresponding to
alfxa(y),betxa(y),gamxa(y),
c   gbdxa(y) must be returned. alfxa(y) = 0. is
not allowed for any y.
c   (see ierror = 13)
c
c   (2)   the kbdy=2 boundary
c
c   this is the edge x=xb where nxb=iparm(3) = 2
flags
c   a mixed boundary condition of the form
c
c           alfxb(y)*px + betxb(y)*py +
gamxb(y)*p(xb,y) = gbdxb(y)
c
c   in this case kbdy=2,xory=y will be input to
bndyc and
c   alfa,beta,gama,gbody corresponding to
alfxb(y),betxb(y),gamxb(y),
c   gbdxb(y) must be returned. alfxb(y) = 0.0 is
not allowed for any y.
c   (see ierror = 13)
c
c   (3)   the kbdy=3 boundary
c
c   this is the edge y=yc where nyc=iparm(4) = 2
flags

```

```

c           a mixed boundary condition of the form
c
c           alfyC(x)*px + betyC(x)*py +
gamyC(x)*p(x,yc) = gbdyC(x)
c
c           in this case kbdy=3,xory=x will be input to
bndyC and
c           alfa,beta,gama,gbdy corresponding to
alfyC(x),betyC(x),gamyC(x),
c           gbdyC(x) must be returned. betyC(x) = 0.0 is
not allowed for any x.
c           (see ierror = 13)
c
c   (4)    the kbdy=4 boundary
c
c           this is the edge y=yd where nyd=iparm(5) = 2
flags
c           a mixed boundary condition of the form
c
c           alfyd(x)*px + betyd(x)*py +
gamyd(x)*p(x,yd) = gbdyd(x)
c
c           in this case kbdy=4,xory=x will be input to
bndyC and
c           alfa,beta,gama,gbdy corresponding to
alfyd(x),betyd(x),gamyd(x),
c           gbdyd(x) must be returned. betyd(x) = 0.0 is
not allowed for any x.
c           (see ierror = 13)
c
c
c ***      bndyC must provide the mixed boundary
condition values
c           in correspondence with those flagged in
iparm(2) thru
c           iparm(5). if all boundaries are specified or
periodic
c           cud2cr will never call bndyC. even then it
must be entered
c           as a dummy subroutine. bndyC must be declared
"external"
c           in the routine calling cud2cr. the actual
name chosen may
c           be different.

```

```

c
c
c ... coef
c
c           a subroutine with arguments
(x,y,cxx,cxy,cyy,cx,cy,ce) which
c           provides the known complex coefficients for
the elliptic pde at
c           any grid point (x,y). the name chosen in the
calling routine
c           may be different where the coefficient routine
must be declared
c           "external."
c
c ... rhs
c
c           a complex array dimensioned nx by ny which
contains the given
c           right hand side values on the uniform 2-d
mesh.
c
c           rhs(i,j) = r(xi,yj) for i=1,...,nx and
j=1,...,ny
c
c ... phi
c
c           a complex array dimensioned nx by ny. on
input phi must contain
c           specified boundary values. for example, if
nyd=iparm(5)=1
c           then phi(i,ny) must be set equal to p(xi,yd)
for i=1,...nx
c           prior to calling cud2cr. these values are
preserved by cud2cr.
c           if an initial guess is provided
(iguess=iparm(11)=1) it must
c           be input thru phi.
c
c
c ***      if no initial guess is given (iguess=0) then
phi must still
c           be initialized at all grid points (this is
not checked). these
c           values will serve as an initial guess to the

```

```
pde at the coarsest
c           grid level after a transfer from the fine
solution grid.  set phi
c           equal to to 0.0 at all internal and non-
specified boundaries
c           grid points if nothing better is available.
c
c
c ... mgopt
c
c           an integer vector of length 4 which allows
the user to select
c           among various multigrid options.  if
mgopt(1)=0 is input then
c           a default set of multigrid arguments (chosen
for robustness)
c           will be internally selected and the
remaining values in mgopt
c           will be ignored.  if mgopt(1) is nonzero
then the arguments
c           in mgopt are set internally and defined as
follows: (see the
c           basic coarse grid correction algorithm
below)
c
c
c     kcycle = mgopt(1)
c
c           = 0 if default multigrid options are to be
used
c
c           = 1 if v cycling is to be used (the least
expensive per cycle)
c
c           = 2 if w cycling is to be used (the
default)
c
c           > 2 if more general k cycling is to be used
c           *** warning--values larger than 2 increase
c           the execution time per cycle
considerably and
c           result in the nonfatal error ierror =
-5
c           which indicates inefficient multigrid
```

```
cycling.  
c  
c      iprер = mgopt(2)  
c  
c          the number of "pre-relaxation" sweeps  
executed before the  
c          residual is restricted and cycling is  
invoked at the next  
c          coarser grid level (default value is 2  
whenever mgopt(1)=0)  
c  
c      ipost = mgopt(3)  
c  
c          the number of "post relaxation" sweeps  
executed after cycling  
c          has been invoked at the next coarser grid  
level and the residual  
c          correction has been transferred back  
(default value is 1  
c          whenever mgopt(1)=0).  
c  
c *** if iprер, ipost, or (especially) kcycle is greater  
than 2  
c      than inefficient multigrid cycling has probably  
been chosen and  
c      the nonfatal error (see below) ierror = -5 will be  
set. note  
c      this warning may be overridden by any other  
nonzero value  
c      for ierror.  
c  
c      intpol = mgopt(4)  
c  
c          = 1 if multilinear prolongation  
(interpolation) is used to  
c          transfer residual corrections and the pde  
approximation  
c          from coarse to fine grids within full  
multigrid cycling.  
c  
c          = 3 if multicubic prolongation  
(interpolation) is used to  
c          transfer residual corrections and the pde  
approximation
```

```

c           from coarse to fine grids within full
multigrid cycling.
c           (this is the default value whenever
mgopt(1)=0).
c
c *** the default values (2,2,1,3) in the vector mgopt
were chosen for
c       robustness. in some cases v(2,1) cycles with
linear prolongation will
c       give good results with less computation
(especially in two-dimensions).
c       this was the default and only choice in an
earlier version of mudpack
c       (see [1]) and can be set with the integer vector
(1,2,1,1) in mgopt.
c
c *** the schedules for one full multigrid cycle
(iguess=0) using v(2,1)
c       cycles and w(2,1) cycles are depicted for a four
level grid below.
c       the number of relaxation sweeps when each grid is
visited are indicated.
c       the "*" stands for prolongation of the full
approximation and the "."
c       stands for transfer of residuals and residual
corrections within the
c       coarse grid correction algorithm (see below). all
version 5.0.1
c       mudpack solvers use only fully weighted residual
restriction
c
c       one fmg with v(2,1) cycles:
c
c
c       -----
c       level 4
c               * .
c               * .
c       -----
c               2-----1-----1-----2-----1-----
c
c       level 3
c               * .
c               * .
c       -----
c               2-----1-----2-----1-----2-----1-----
c
c       level 2

```

```

C          *
C          *   .
C          .   .
C          ---3---3-----3-----3-----
C          ----- level 1
C
C
C      one fmg with w(2,1) cycles:
C
C      -----
C      -----2-----
--1--      level 4
C          *
C          .
C          -----2-----1---2-----3-----
1----      level 3
C          *
C          .   .
C          ----2---1---2---3---1-----2---3---1---2---3---1-
C          ----- level 2
C          *
C          .   .   .   .   .   .   .   .   .   .   .
C          --6---6-----6---6-----6---6-----6---6-----6---6---
C          ----- level 1
C
C
C      the form of the "recursive" coarse grid correction
C      cycling used
C      when kcycle.ge.0 is input is described below in
C      pseudo-algorithmic
C      language. it is implemented non-recursively in
C      fortran in mudpack.
C
C      algorithm
cgc(k,l(k),u(k),r(k),kcycle,iprer,ipost,iresw,intpol)
C
C *** approximately solve l(k)*u(k) = r(k) using
C      multigrid iteration
C *** k is the current grid level
C *** l(k) is the discretized pde operator at level k
C *** u(k) is the initial guess at level k
C *** r(k) is the right hand side at level k
C *** i(k,k-1) is the restriction operator from level k
C      to level k-1
C *** (the form of i(k,k-1) depends on iresw)
C *** i(k-1,k) is the prolongation operator from level
C      k-1 to level k
C *** (the form of i(k-1,k) depends on interpol)

```

```

C
C      begin algorithm cgc
C
C ***      pre-relax at level k
C
C      . do (i=1,iprer)
C
C      . . relax(l(k),u(k),r(k))
C
C      . end do
C
C      . if (k > 1) then
C
C ***      restrict the residual from level k to level k-
1
C
C      . . r(k-1) = i(k,k-1) (r(k)-l(k)*u(k))
C
C      . . kount = 0
C
C      . . repeat
C
C ***      solve for the residual correction at level k-1
in u(k-1)
C ***      using algorithm cgc "kcycle" times (this is
the recursion)
C
C      . . . kount = kount+1
C
C      . . . invoke cgc(k-1,l(k-1),u(k-1),r(k-
1),kcycle,iprer,ipost,iresh)
C
C
C      . . until (kount.eq.kcycle)
C
C ***      transfer residual correction in u(k-1) to
level k
C ***      with the prolongation operator and add to u(k)
C
C      . . u(k) = u(k) + i(k-1,k) (u(k-1))
C
C      . end if
C
C ***      post relax at level k

```

```

C
C      . do (i=1,ipost)
C
C      . . relax(l(k),u(k),r(k))
C
C      . end do
C
C      . return
C
C      end algorithm cgc
C
C
C
C ****
C *** output arguments
C ****
C
C ****
C ... iparm(16) *** set for intl=0 calls only
C
C          on output iparm(16) contains the actual work
C space length
C          required. this will usually be less than
C that given by the
C          simplified formula for length=iparm(15) (see
C as input argument)
C
C
C ... iparm(17) *** set for intl=1 calls only
C
C          on output iparm(17) contains the actual
C number of multigrid cycles
C          between the finest and coarsest grid levels
C used to obtain the
C          approximation when error control (tolmax >
C 0.0) is set.
C
C
C ... fparm(6) *** set for intl=1 calls with fparm(5)
C > 0. only

```

```

c
c          on output fparm(6) contains the final
computed maximum relative
c          difference between the last two iterates at
the finest grid level.
c          fparm(6) is computed only if there is error
control (tolmax > 0.0)
c          assume phi1(i,j,k) and phi2(i,j,k) are the
last two computed
c          values for phi(i,j,k) at all points of the
finest grid level.
c          if we define
c
c          phdif = max(abs(phi2(i,j)-phi1(i,j)))
over all i,j
c
c          and
c
c          phmax = max(abs(phi2(i,j))) over all i,j
c
c          then
c
c          fparm(6) = phdif/phmax
c
c          is returned whenever phmax > 0.0. in the
degenerate case
c          phmax = 0.0, fparm(6) = phdif is returned.
c
c
c ... work
c
c          on output work contains intermediate values
that must not
c          be destroyed if cud2cr is to be called again
with intl=1
c
c
c ... phi    *** for intl=1 calls only
c
c          on output phi(i,j) contains the approximation
to p(xi,yj)
c          for all mesh points i = 1,...,nx and
j=1,...,ny. the last
c          computed iterate in phi is returned even if

```

```
convergence is
c           not obtained
c
c
c ... ierror
c
c           for intl=iparm(1)=0 initialization calls,
ierror is an
c           error flag that indicates invalid input
arguments when
c           returned positive and nonfatal warnings when
returned
c           negative. argument checking and
discretization
c           is bypassed for intl=1 calls which can only
return
c           ierror = -1 or 0 or 1.
c
c
c           non-fatal warnings * * *
c
c
c     ==5 if kcycle=mgopt(1) is greater than 2. values
larger than 2 results
c           in an algorithm which probably does far more
computation than
c           necessary. kcycle = 1 (v cycles) or kcycle=2
(w cycles) should
c           suffice for most problems. ierror = -5 is
also set if either
c           iprer = mgopt(2) or ipost=mgopt(3) is greater
than 2. the
c           ierror=-5 flag is overridden by any other
fatal or non-fatal
c           error.
c
c     ==4 if there are dominant nonzero first order
terms in the pde which
c           make it "hyperbolic" at the finest grid level.
numerically, this
c           happens if:
c
c           abs(cx)*dlx > 2.*abs(cxx)      (dlx = (xb-
xa) / (nx-1))
```

```

c
c                               (or)
c
c           abs(cy)*dly > 2.*abs(cyy)      (dly = (yd-
yc) / (ny-1))
c
c
c           at some fine grid point (xi,yj). if an
adjustment is not made the
c           condition can lead to a matrix coming from the
discretization
c           which is not diagonally dominant and
divergence is possible. since
c           the condition is "likely" at coarser grid
levels for pde's with
c           nonzero first order terms, the adjustments
(actually first order
c           approximations to the pde)
c
c
c           cxx = amax1(cxx,0.5*abs(cx)*dx)
c
c                               (and)
c
c           cyy = amax1(cyy,0.5*abs(cy)*dy)
c
c
c           (here dx,dy are the x,y mesh sizes of the
subgrid)
c
c           are made to preserve convergence of multigrid
iteration. if made
c           at the finest grid level, it can lead to
convergence to an
c           erroneous solution (flagged by ierror = -4).
a possible remedy
c           is to increase resolution. the ierror = -4
flag overrides the
c           nonfatal ierror = -5 flag.
c
c
c           ==3 if the continuous elliptic pde is singular.
this means the
c           boundary conditions are periodic or pure

```

```

derivative at all
c           boundaries and ce(x,y) = 0.0 for all x,y. a
solution is still
c           attempted but convergence may not occur due
to ill-conditioning
c           of the linear system coming from the
discretization. the
c           ierror = -3 flag overrides the ierror=-4 and
ierror=-5 nonfatal
c           flags.

c
c
c     =-2 if the pde is not elliptic (i.e.,
cxx*cyy.le.0.0 for some (xi,yj))
c           in this case a solution is still attempted
although convergence
c           may not occur due to ill-conditioning of the
linear system.
c           the ierror = -2 flag overrides the ierror=-
5,-4,-3 nonfatal
c           flags.

c
c
c     =-1 if convergence to the tolerance specified in
tolmax=fparm(5)>0.
c           is not obtained in maxcy=iparm(13) multigrid
cycles between the
c           coarsest (ixp+1,jyq+1) and finest (nx,ny)
grid levels.
c           in this case the last computed iterate is
still returned.
c           the ierror = -1 flag overrides all other
nonfatal flags

c
c
c     no errors * * *

c
c     = 0

c
c     fatal argument errors * * *

c
c     = 1 if intl=iparm(1) is not 0 on initial call or
not 0 or 1
c           on subsequent calls

```

```

C
C      = 2 if any of the boundary condition flags
nxa,nxb,nyc,nyd
C          in iparm(2),iparm(3),iparm(4),iparm(5) are not
0,1 or 2
C          or if nxa,nxb or nyc,nyd are not pairwise
zero.
C
C      = 3 if mino(ixp,jyq) < 2 (ixp = iparm(6), jyq =
iparm(7))
C
C      = 4 if min0(iex,jey) < 1 (iex = iparm(8), jey =
iparm(9)) or
C          if max0(iex,jey) > 50
C
C      = 5 if nx.ne.ixp*2** (iex-1)+1 or
ny.ne.jyq*2** (jey-1)+1
C          (nx = iparm(10), ny = iparm(11))
C
C      = 6 if iguess = iparm(12) is not equal to 0 or 1
C
C      = 7 if maxcy = iparm(13) < 1
C
C      = 8 if method = iparm(14) is not 0,1,2, or 3
C
C      = 9 if length = iparm(15) is too small (see
iparm(16) on output
C          for minimum required work space length)
C
C      =10 if xa >= xb or yc >= yd
C
C (xa=fparm(1),xb=fparm(2),yc=fparm(3),yd=fparm(4))
C
C      =11 if tolmax = fparm(5) < 0.0
C
C      errors in setting multigrid options * * * (see
also ierror=-5)
C
C      =12 if kcycle = mgopt(1) < 0 or
C          if iprer = mgopt(1) < 1 or
C          if ipost = mgopt(3) < 1 or
C          if interpol = mgopt(4) is not 1 or 3
C
C      =13 if there is a pure tangential derivative along

```

```

a mixed derivative
c           boundary (e.g., nyd = 2 and betyd(x) = 0.0 for
some
c           grid point x along y = yd)
c
c     =14 if there is the "singular" condition described
below at a
c           corner which is the intersection of two
derivative boundaries.
c
c           (1) the corner (xa,yc) if nxa=nyc=2 and
c               alfxa(yc)*betyc(xa)-alfyc(xa)*betxa(yc) =
0.0.
c
c           (2) the corner (xa,yd) if nxa=nyd=2 and
c               alfxa(yd)*betyd(xa)-alfyd(xa)*betxa(yd) =
0.0.
c
c           (3) the corner (xb,yc) if nxb=nyc=2 and
c               alfxb(yc)*betyc(xb)-alfyc(xb)*betxb(yc) =
0.0.
c
c           (4) the corner (xb,yd) if nxb=nyd=2 and
c               alfxb(yd)*betyd(xb)-alfyd(xb)*betxb(yd) =
0.0.

c *** the conditions described in ierror = 13 or 14 will
lead to division
c       by zero during discretization if undetected.
c
c
c ****
*
c ****
*
c
c       end of cud2cr documentation
c
c ****
**
c
```

```
*****  
**  
C  
C
```

---

## CUD2SP

```
C  
C      file cud2sp.d  
C  
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C      *  
*  
C      *          copyright (c) 2008 by UCAR  
*  
C      *  
*  
C      *          University Corporation for Atmospheric  
Research           *  
C      *  
*  
C      *          all rights reserved  
*  
C      *  
*  
C      *          MUDPACK version 5.0.1  
*  
C      *  
*  
C      *          A Fortran Package of Multigrid  
*  
C      *  
*  
C      *          Subroutines and Example Programs  
*  
C      *  
*
```

```
C      *      for Solving Elliptic Partial Differential
Equations      *
C      *
*
C      *                      by
*
C      *
*
C      *                      John Adams
*
C      *
*
C      *                      of
*
C      *
*
C      *      the National Center for Atmospheric
Research      *
C      *
*
C      *      Boulder, Colorado (80307)
U.S.A.      *
C      *
*
C      *      which is sponsored by
*
C      *
*
C      *      the National Science Foundation
*
C      *
*
C      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *
*      *      *      *      *      *
C
C ... file cud2sp.d
C
C      contains documentation for:
C      subroutine
cud2sp(iparm,fparm,work,cofx,cofy,bndyc,rhs,phi,mgopt,ie
rror)
C      A sample fortran driver is file "tcud2sp.f".
C
C ... required MUDPACK files
```

```

c
c      cudcom.f
c
c
c ... purpose
c
c      subroutine cud2sp automatically discretizes and
attempts to compute
c      the second-order difference approximation to the
complex two-dimensional
c      linear separable elliptic partial differential
equation on a
c      rectangle. the approximation is generated on a
uniform grid covering
c      the rectangle (see mesh description below).
boundary conditions
c      may be specified (dirchlet), periodic, or mixed
derivative in any
c      combination. the form of the pde solved is:
c
c
c      cxx(x)*pxx + cx(x)*px + cex(x)*p(x,y) +
c
c      cyy(y)*pyy + cy(y)*py + cey(y)*p(x,y) =
r(x,y)
c
c      pxx,pyy,px,py are second and first partial
derivatives of the
c      unknown complex solution function p(x,y) with
respect to the
c      independent variables x,y. cxx,cx,cex,cyy,cy,cey
are the known
c      complex coefficients of the elliptic pde and
r(x,y) is the known
c      complex right hand side of the equation. cxx and
cyy should be
c      positive for all x,y in the solution region. If
some of the
c      coefficients depend on both x and y then the PDE
is nonseparable.
c      In this case subroutine cud2 must be used instead
of cud2sp
c      (see the file cud2.d)
c
c

```

```
c ... mesh description . . .
c
c      the approximation is generated on a uniform nx by
ny grid.  the grid
c      is superimposed on the rectangular solution region
c
c            [xa,xb] x [yc,yd].
c
c      let
c
c            dlx = (xb-xa) / (nx-1), dly = (yd-yc) / (ny-1)
c
c      be the uniform grid increments in the x,y
directions. then
c
c            xi=xa+(i-1)*dlx,   yj=yc+(j-1)*dly
c
c      for i=1,...,nx and j=1,...,ny denote the x,y
uniform mesh points
c
c
c ... language
c
c      fortran90/fortran77
c
c
c ... portability
c
c      mudpack5.0.1 software has been compiled and tested
with Fortran77
c      and Fortran90 on a variety of platforms.
c
c ... methods
c
c      details of the methods employed by the solvers in
mudpack are given
c      in [1,9].  [1,2,9] contain performance
measurements on a variety of
c      elliptic pdes (see "references" in the file
"readme").  in summary:
c
c *** discretization and solution (second-order solvers)
(see [1])
c
```

```
c      the pde and boundary conditions are automatically
c      discretized at all
c      grid levels using second-order finite difference
c      formula. diagonal
c      dominance at coarser grid levels is maintained in
c      the presence of
c      nonzero first-order terms by adjusting the second-
c      order coefficient
c      when necessary. the resulting block tri-diagonal
c      linear system is
c      approximated using multigrid iteration
[10,11,13,15,16,18]. version
c      5.0.1 of mudpack uses only fully weighted residual
c      restriction. defaults
c      include cubic prolongation and w(2,1) cycles.
c      these can be overridden
c      with selected multigrid options (see "mgopt"). error
c      control based on
c      maximum relative differences is available. full
c      multigrid cycling (fmg)
c      or cycling beginning or restarting at the finest
c      grid level can be
c      selected. a menu of relaxation methods including
c      gauss-seidel point,
c      line relaxation(s) (in any combination of
c      directions) and planar
c      relaxation (for three-dimensional anisotropic
c      problems) are provided.
c      all methods use ordering based on alternating
c      points (red/black),
c      lines, or planes for cray vectorization and
c      improved convergence
c      rates [14].
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
c      estimate (i.e.,
c      discretization level error is reached) then this
c      can be improved to a
c      fourth-order estimate using the technique of
c      "deferred corrections."
c      the values in the solution array are used to
```

```
generate a fourth-order
c      approximation to the truncation error. second-
order finite difference
c      formula are used to approximate third and fourth
partial derivatives
c      of the solution function [3]. the truncation
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side. the default
multigrid options
c      are used to compute the fourth-order correction
term which is added
c      to the original solution array.

c
c
c ... references (partial)

c
c
c [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
c      Solution of Linear Elliptic Partial Differential
Equations,"
c      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.

c
c      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
c      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
c      1989, pp.1-12.

c      .
c      .
c      .

c      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
Haidvogel, and V. Pizzo,
c      "Applications of Multigrid Software in the
Atmospheric Sciences,"
c      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-
1458.

c      .
c      .
c      .

c      [9] J. Adams, "Recent Enhancements in MUDPACK, a
```

```
Multigrid Software
c      package for Elliptic Partial Differential
Equations," Applied Math.
c      and Comp., 1991, vol. 43, May 1991, pp. 79-94.
c
c      [10]J. Adams, "MUDPACK-2: Multigrid Software for
Approximating
c      Elliptic Partial Differential Equations on Uniform
Grids with
c      any Resolution," Applied Math. and Comp., 1993,
vol. 53, February
c      1993, pp. 235-249
c      .
c      .
c      .
c
c ... argument description
c
c
c
c ****
c *** input arguments
c ****
c
c ****
c
c
c ... iparm
c
c      an integer vector of length 17 used to pass
integer
c      arguments. iparm is set internally and
defined as
c      follows:
c
c
c ... intl=iparm(1)
c
c      an initialization argument. intl=0 must be
input
c      on an initial call. in this case input
arguments will
```

```
c           be checked for errors and the elliptic
partial differential
c           equation and boundary conditions will be
discretized using
c           second order finite difference formula.
c
c ***      An approximation is NOT generated after an
intl=0 call!
c           cud2sp should be called with intl=1 to
approximate the elliptic
c           PDE discretized by the intl=0 call.  intl=1
should also
c           be input if cud2sp has been called earlier
and only the
c           values in in rhs (see below) or gbdy (see
bndyc below)
c           or phi (see below) have changed.  This will
bypass
c           redundant pde discretization and argument
checking
c           and save computational time.  Some examples
of when
c           intl=1 calls should be used are:
c
c           (0) after a intl=0 argument checking and
discretization call
c
c           (1) cud2sp is being recalled for additional
accuracy.  In
c               this case iguess=iparm(12)=1 should also
be used.
c
c           (2) cud2sp is being called every time step in
a time dependent
c               problem (see discussion below) where the
elliptic operator
c               does not depend on time.
c
c           (3) cud2sp is being used to solve the same
elliptic equation
c               for several different right hand sides
(iguess=0 should
c               probably be used for each new righthand
side).
```

```
c
c           intl = 0 must be input before calling with
intl = 1 when any
c           of the following conditions hold:
c
c           (a) the initial call to cud2sp
c
c           (b) any of the integer arguments other than
iguess=iparm(12)
c           or maxcy=iparm(13) or mgopt have changed
since the previous
c           call.
c
c           (c) any of the floating point arguments other
than tolmax=
c           fparm(5) have changed since the previous
call
c
c           (d) any of the coefficients input by
cofx,cofy (see below) have
c           changed since the previous call
c
c           (e) any of the constant "alfa" coefficients
input by bndyc
c           (see below) have changed since the
previous call.
c
c           If any of (a) through (e) are true then the
elliptic PDE
c           must be discretized or rediscritized.  If
none of (a)
c           through (e) holds, calls can be made with
intl=1.
c           Incorrect calls with intl=1 will produce
erroneous results.
c ***      The values set in the saved work space "work"
(see below) with
c           an intl=0 call must be preserved with
subsequent intl=1 calls.
c
c           MUDPACK software performance should be
monitored for intl=1
c           calls.  The intl=0 discretization call
performance depends
```

```

c      primarily on the efficiency or lack of
efficiency of the
c      user provided subroutines for pde
coefficients and
c      boundary conditions.

c
c
c ... nxa=iparm(2)
c
c      flags boundary conditions on the edge x=xa
c
c      = 0 if p(x,y) is periodic in x on [xa,xb]
c          (i.e., p(x+xb-xa,y) = p(x,y) for all x,y
c          (if nxa=0 then nxb=0 is required, see
ierror = 2)
c
c      = 1 if p(xa,y) is specified (this must be input
thru phi(1,j))
c
c      = 2 if there are mixed derivative boundary
conditions at x=xa
c          (see bndyc)
c
c
c ... nxb=iparm(3)
c
c      flags boundary conditions on the edge x=xb
c
c      = 0 if p(x,y) is periodic in x on [xa,xb]
c          (i.e., p(x+xb-xa,y) = p(x,y) for all x,y
c          (if nxb=0 then nxa=0 is required, see
ierror = 2)
c
c      = 1 if p(xb,y) is specified (this must be input
thru phi(nx,j))
c
c      = 2 if there are mixed derivative boundary
conditions at x=xb
c          (see bndyc)
c
c
c ... nyc=iparm(4)
c
c      flags boundary conditions on the edge y=y

```

```

C
C      = 0 if p(x,y) is periodic in y on [yc,yd]
C          (i.e., p(x,y+yd-yc) = p(x,y) for all x,y
C          (if nyc=0 then nyd=0 is required, see
ierror = 2)
C
C      = 1 if p(x,yc) is specified (this must be input
thru phi(i,1))
C
C      = 2 if there are mixed derivative boundary
conditions at y=yc
C          (see bndyc)
C
C
C ... nyd=iparm(5)
C
C      flags boundary conditions on the edge y=yd
C
C      = 0 if p(x,y) is periodic in y on [yc,yd]
C          (i.e., p(x,y+yd-yc) = p(x,y) for all x,y
C          (if nyd=0 then nyc=0 is required, see
ierror = 2)
C
C      = 1 if p(x,yd) is specified (this must be input
thru phi(i,ny))
C
C      = 2 if there are mixed derivative boundary
conditions at y=yd
C          (see bndyc)
C
C
C *** grid size arguments
C
C
C ... ixp = iparm(6)
C
C      an integer greater than one which is used in
defining the number
C          of grid points in the x direction (see nx =
iparm(10)). "ixp+1"
C          is the number of points on the coarsest x
grid visited during
C          multigrid cycling. ixp should be chosen as
small as possible.

```

```
c           recommended values are the small primes 2 or
3.
c           larger values can reduce multigrid
convergence rates considerably,
c           especially if line relaxation in the x
direction is not used.
c           if ixp > 2 then it should be 2 or a small odd
value since a power
c           of 2 factor of ixp can be removed by
increasing iex = iparm(8)
c           without changing nx = iparm(10).
c
c
c ... jyq = iparm(7)
c
c           an integer greater than one which is used in
defining the number
c           of grid points in the y direction (see ny =
iparm(11)). "jyq+1"
c           is the number of points on the coarsest y
grid visited during
c           multigrid cycling. jyq should be chosen as
small as possible.
c           recommended values are the small primes 2 or
3.
c           larger values can reduce multigrid
convergence rates considerably,
c           especially if line relaxation in the y
direction is not used.
c           if jyq > 2 then it should be 2 or a small odd
value since a power
c           of 2 factor of jyq can be removed by
increasing jey = iparm(9)
c           without changing ny = iparm(11).
c
c
c ... iex = iparm(8)
c
c           a positive integer exponent of 2 used in
defining the number
c           of grid points in the x direction (see nx =
iparm(10)).
c           iex .le. 50 is required. for efficient
multigrid cycling,
```

```

c           iex should be chosen as large as possible and
ixp=iparm(8)
c           as small as possible within grid size
constraints when
c           defining nx.
c
c
c ... jey = iparm(9)
c
c           a positive integer exponent of 2 used in
defining the number
c           of grid points in the y direction (see ny =
iparm(11)).
c           jey .le. 50 is required. for efficient
multigrid cycling,
c           jey should be chosen as large as possible and
jyq=iparm(7)
c           as small as possible within grid size
constraints when
c           defining ny.
c
c
c
c ... nx = iparm(10)
c
c           the number of equally spaced grid points in
the interval [xa,xb]
c           (including the boundaries). nx must have the
form
c
c           nx = ixp* (2** (iex-1)) + 1
c
c           where ixp = iparm(6), iex = iparm(8).
c
c
c ... ny = iparm(11)
c
c           the number of equally spaced grid points in
the interval [yc,yd]
c           (including the boundaries). ny must have the
form:
c
c           ny = jyq* (2** (jey-1)) + 1
c

```

```

C           where jyq = iparm(7), jey = iparm(9).
C
C
C *** example
C
C           suppose a solution is wanted on a 33 by 97
grid.  then
C           ixp=2, jyq=6 and iex=jey=5 could be used.  a
better
C           choice would be ixp=2, jyq=3, and iex=5,
jey=6.
C
C
C *** note
C
C           let G be the nx by ny fine grid on which the
approximation is
C           generated and let n = max0(iex,jey).  in mudpack,
multigrid
C           cycling is implemented on the ascending chain of
grids
C
C           G(1) < ... < G(k) < ... < G(n) = G.
C
C           each G(k) (k=1,...,n) has mx(k) by my(k) grid
points
C           given by:
C
C           mx(k) = ixp*[2**max0(iex+k-n,1)-1] + 1
C
C           my(k) = jyq*[2**max0(jey+k-n,1)-1] + 1
C
C
C
C ... iguess=iparm(12)
C
C           = 0 if no initial guess to the pde is
provided
C
C           = 1 if an initial guess to the pde is at the
finest grid
C           level is provided in phi (see below)
C
C           comments on iguess = 0 or 1 . . .

```

```
c
c      even if iguess = 0, phi must be initialized at all
grid points (this
c      is not checked).  phi can be set to 0.0 at non-
dirchlet grid points
c      if nothing better is available.  the values set in
phi when iguess = 0
c      are passed down and serve as an initial guess to
the pde at the coarsest
c      grid level where cycling commences.  in this
sense, values input in
c      phi always serve as an initial guess.  setting
iguess = 0 forces full
c      multigrid cycling beginning at the coarsest and
finishing at the finest
c      grid level.

c
c      if iguess = 1 then the values input in phi are an
initial guess to the
c      pde at the finest grid level where cycling begins.
this option should
c      be used only if a "very good" initial guess is
available (as, for
c      example, when restarting from a previous iguess=0
call).

c
c      time dependent problems . . .
c
c *** assume we are solving an elliptic pde every time
step in a
c      marching problem of the form:
c
c      l(p(t)) = r(t)
c
c      where the differential operator "l" has no time
dependence,
c      "p(t)" is the solution and "r(t)" is the right
hand side at
c      current time "t". let "dt" be the increment
between time steps.
c      then p(t) can be used as an initial guess to
p(t+dt) with
c      intl = 1 when solving
c
```

```

c           l(p(t+dt)) = r(t+dt).
c
c       after the first two time steps, rather than
c       continue, it would
c       be better to define the "correction" term:
c
c           e(t,dt) = p(t+dt) - p(t)
c
c       this clearly satisfies the equation
c
c           l(e(t,dt)) = r(t+dt) - r(t).
c
c       this should be solved with iguess = 0 and intl =
1. boundary
c       conditions for e(t,dt) are obtained from the
boundary conditions
c       for p(t) by subtracting given values at t from
given values at
c       t+dt. for example if
c
c           d(p(t))/dx = f(t), d(p(t+dt))/dx = f(t+dt)
c
c       at some x boundary then e(t,dt) satisfies the
derivative
c       boundary condition
c
c           d(e(t,dt))/dx = f(t+dt) - f(t).
c
c       e(t,dt) can be preset to 0.0 (at nondirichlet
points) or (if p(t-dt)
c       is saved) to p(t)-p(t-dt). with iguess = 0, these
values will serve
c       as an initial guess to e(t,dt) at the coarsest
grid level. this
c       approach has the advantage that a full sequence of
multigrid cycles,
c       beginning at the coarsest grid level, is invoked
every time step in
c       solving for e(t,dt). a few digits of accuracy in
e(t,dt), which is
c       ordinarily much smaller than p(t), will yield
several more digits of
c       accuracy in the final approximation:
c

```

```

c           p(t+dt) = p(t) + e(t,dt).
c
c       using this approach to integrate in time will give
more accuracy
c       then using p(t) as an initial guess to p(t+dt) for
all time steps.
c       it does require additional storage.
c
c       if the differential operator "l" has time
dependence (either thru
c       the coefficients in the pde or the coefficients in
the derivative
c       boundary conditions) then use p(t) as an initial
guess to p(t+dt)
c       when solving
c
c           l(t+dt) (p(t+dt)) = r(t+dt)
c
c       with intl = 0 for all time steps (the
discretization must be repeated
c       for each new "t"). either iguess = 0 (p(t) will
then be an initial
c       guess at the coarsest grid level where cycles will
commence) or
c       iguess = 1 (p(t) will then be an initial guess at
the finest grid
c       level where cycles will remain fixed) can be
tried.
c
c
c ... maxcy = iparm(13)
c
c           the exact number of cycles executed between
the finest (nx by
c           ny) and the coarsest ((ixp+1) by (jyq+1))
grid levels when
c           tolmax=fparm(5)=0.0 (no error control). when
tolmax > 0.0
c           is input (error control) then maxcy is a
limit on the number
c           of cycles between the finest and coarsest
grid levels. in
c           any case, at most maxcy*(iprert+ipost)
relaxation sweeps are

```

```

c           are performed at the finest grid level (see
iprер=mgopt(2),
c           ipost=mgopt(3) below). when multigrid
iteration is working
c           "correctly" only a few are required for
convergence. large
c           values for maxcy should not be necessary.
c
c
c ... method = iparm(14) determines the method of
relaxation
c           (gauss-seidel based on alternating points
or lines)
c
c           = 0 for point relaxation
c
c           = 1 for line relaxation in the x direction
c
c           = 2 for line relaxation in the y direction
c
c           = 3 for line relaxation in both the x and y
direction
c
c
c *** choice of method. . .
c
c     let fx represent the quantity cxx(x,y)/dlx**2 over
the solution region.
c
c     let fy represent the quantity cyy(x,y)/dly**2 over
the solution region
c
c     if fx,fy are roughly the same size and do not vary
too much over
c     the solution region choose method = 0. if this
fails try method=3.
c
c     if fx is much greater than fy choose method = 1.
c
c     if fy is much greater than fx choose method = 2
c
c     if neither fx or fy dominates over the solution
region and they
c     both vary considerably choose method = 3.

```

```
c
c
c ... length = iparm(15)
c
c           the length of the complex work space provided
in vector work (see below).
c           let isx = 0 if method = 0 or method = 2
c           let isx = 3 if method = 1 or method = 3 and
nxa.ne.0
c           let isx = 5 if method = 1 or method = 3 and
nxa.eq.0
c           let jsy = 0 if method = 0 or method = 1
c           let jsy = 3 if method = 2 or method = 3 and
nyc.ne.0
c           let jsy = 5 if method = 2 or method = 3 and
nyc.eq.0
c           then . . .
c
c           length = nx*ny* (5+3*(isx+jsy)/2) +
10* (nx+ny)
c
c           will suffice in all cases but very small nx
and ny.
c           the exact minimal work space length required
for the
c           current set of input arugments is output in
iparm(16).
c           (even if iparm(15) is too small). this will
be usually
c           be less then the value given by the
simplified formula
c           above. * Notice that cud2sp requires
considerably less
c           work space than the nonseparable solver cud2
if
c           and only if method=0 is chosen.
c
c ... fparm
c
c           a floating point vector of length 6 used to
efficiently
c           pass floating point arguments. fparm is set
internally
c           in cud2sp and defined as follows . . .
```

```

C
C
C ... xa=fparm(1), xb=fparm(2)
C
C           the range of the x independent variable. xa
must
C           be less than xb
C
C
C ... yc=fparm(3), yd=fparm(4)
C
C           the range of the y independent variable. yc
must
C           be less than yd.
C
C
C ... tolmax = fparm(5)
C
C           when input positive, tolmax is a maximum
relative error tolerance
C           used to terminate the relaxation iterations.
assume phi1(i,j)
C           and phi2(i,j) are the last two computed
approximations at all
C           grid points of the finest grid level. if we
define
C
C           phdif = max(cabs(phi2(i,j)-phi1(i,j)))
for all i,j
C
C           and
C
C           phmax = max(cabs(phi2(i,j))) for all i,j
C
C           then "convergence" is considered to have
occurred if and only if
C
C           phdif/phmax < tolmax.
C
C
C           if tolmax=fparm(5)=0.0 is input then there is
no error control
C           and maxcy cycles from the finest grid level
are executed. maxcy

```

```
c           is a limit which cannot be exceeded even with
error control.
c           *** calls with tolmax=0.0, when appropriate
because of known
c           convergence behavior, are more efficient than
calls with tolmax
c           positive (i.e., if possible DO NOT use error
control!).
c
c ... work
c
c           a one dimensional complex saved work space
(see iparm(15) for
c           length) which must be preserved from the
previous call when
c           calling with intl=iparm(1)=1.
c
c ... bndyc
c
c           a subroutine with arguments
(kbdy,xory,alfa,gbdy) which
c           are used to input mixed boundary conditions
to cud2sp. bndyc
c           must be declared "external" in the program
calling cud2sp.
c           the boundaries are numbered one thru four and
the mixed
c           derivative boundary conditions are described
below (see the
c           sample driver code "tcud2sp.f" for an example
of how bndyc is
c           can beset up).
c
c           * * * * * * * * * * * * * * * * y=yd
c           *         kbdy=4          *
c           *
c           *
c           *
c           *         kbdy=1          kbdy=2  *
c           *
c           *
c           *
c           *
c           *         kbdy=3          *
c           * * * * * * * * * * * * * * * * y=yc
```

```

C           x=xa                      x=xb
C
C
C           (1) the kbdy=1 boundary
C
C           this is the edge x=xa where nxa=iparm(2)=2
flags      a mixed boundary condition of the form
C
C           dp/dx + alfxa*p(xa,y) = gbdxa(y)
C
C           in this case kbdy=1,xory=y will be input to
bndyc and
C           alfa,gbdy corresponding to alfxa,gbdxa(y)
must be returned
C
C
C           (2) the kbdy=2 boundary
C
C           this is the edge x=xb where nxb=iparm(3)=2
flags      a mixed boundary condition of the form
C
C           dp/dx + alfxb*p(xb,y) = gbdxb(y)
C
C           in this case kbdy=2,xory=y, will be input to
bndyc and
C           alfa,gbdy corresponding to alfxb,gbdxb(y)
must be returned.
C
C
C           (3) the kbdy=3 boundary
C
C           this is the edge y=yc where nyc=iparm(4)=2
flags      a mixed boundary condition of the form
C
C           dp/dy + alfyc*p(x,yc) = gbdyc(x)
C
C           in this case kbdy=3,xory=x will be input to
bndyc and
C           alfa,gbdy corresponding to alfyc,gbdyc(x)
must be returned.

```

```

c
c
c      (4) the kbdy=4 boundary
c
c      this is the edge y=yd where nyd=iparm(5)=2
flags
c      a mixed boundary condition of the form
c
c          dp/dy + alfyd*p(x,yd) = gbdyd(x)
c
c      in this case kbdy=4,xory=x will be input to
bndyc and
c      alfa,gbody corresponding to alfyd,gbdyd(x)
must be returned.
c
c
c ***      alfxa,alfxb,alfyc,alfyd must be complex
constants and gbody type
c      complex for cud2sp.  Use cud2 if any of these
depend on x or y.
c      bndyc must provide the mixed boundary
condition values
c      in correspondence with those flagged in
iparm(2) thru
c      iparm(5).  if all boundaries are specified or
periodic
c      cud2sp will never call bndyc.  even then it
must be entered
c      as a dummy subroutine. bndyc must be declared
"external"
c      in the routine calling cud2sp the actual name
chosen may
c      be different.
c
c
c ... cofx
c
c      a subroutine with arguments (x,cxx,cx,cex)
which provides
c      the known x dependent complex coefficients for
the separable
c      elliptic pde at any x grid point.  the name
chosen in the calling
c      routine may be different where the coefficient

```

```
routine must be declared
c          "external."
c
c ... cofy
c
c          a subroutine with arguments (y,cyy,cy,cey)
which provides
c          the known y dependent complex coefficients for
the separable
c          elliptic pde at any y grid point.  the name
chosen in the calling
c          routine may be different where the coefficient
routine must be declared
c          "external."
c
c ... rhs
c
c          a complex array dimensioned nx by ny which
contains the given
c          right hand side values on the uniform 2-d
mesh.
c
c          rhs(i,j) = r(xi,yj) for i=1,...,nx and
j=1,...,ny
c
c ... phi
c
c          a complex array dimensioned nx by ny.  on
input phi must contain
c          specified boundary values.  for example, if
nyd=iparm(5)=1
c          then phi(i,ny) must be set equal to p(xi,yd)
for i=1,...nx
c          prior to calling cud2sp.  these values are
preserved by cud2sp.
c          if an initial guess is provided
(iguess=iparm(11)=1) it must
c          be input thru phi.
c
c
c ***      if no initial guess is given (iguess=0) then
phi must still
c          be initialized at all grid points (this is
not checked).  these
```

```
c           values will serve as an initial guess to the
c           pde at the coarsest
c           grid level after a transfer from the fine
solution grid. set phi
c           equal to to 0.0 at all internal and non-
specified boundaries
c           grid points if nothing better is available.
c
c
c ... mgopt
c
c           an integer vector of length 4 which allows
the user to select
c           among various multigrid options. if
mgopt(1)=0 is input then
c           a default set of multigrid arguments (chosen
for robustness)
c           will be internally selected and the
remaining values in mgopt
c           will be ignored. if mgopt(1) is nonzero
then the arguments
c           in mgopt are set internally and defined as
follows: (see the
c           basic coarse grid correction algorithm
below)
c
c
c     kcycle = mgopt(1)
c
c           = 0 if default multigrid options are to be
used
c
c           = 1 if v cycling is to be used (the least
expensive per cycle)
c
c           = 2 if w cycling is to be used (the
default)
c
c           > 2 if more general k cycling is to be used
c           *** warning--values larger than 2 increase
c           the execution time per cycle
considerably and
c           result in the nonfatal error ierror =
-5
```

```
c which indicates inefficient multigrid
c cycling.
c
c iprер = mgopt(2)
c
c the number of "pre-relaxation" sweeps
c executed before the
c residual is restricted and cycling is
c invoked at the next
c coarser grid level (default value is 2
c whenever mgopt(1)=0)
c
c ipost = mgopt(3)
c
c the number of "post relaxation" sweeps
c executed after cycling
c has been invoked at the next coarser grid
c level and the residual
c correction has been transferred back
c (default value is 1
c whenever mgopt(1)=0).
c
c *** if iprер, ipost, or (especially) kcycle is greater
c than 2
c than inefficient multigrid cycling has probably
c been chosen and
c the nonfatal error (see below) ierror = -5 will be
c set. note
c this warning may be overridden by any other
c nonzero value
c for ierror.
c
c intpol = mgopt(4)
c
c = 1 if multilinear prolongation
c (interpolation) is used to
c transfer residual corrections and the pde
c approximation
c from coarse to fine grids within full
c multigrid cycling.
c
c = 3 if multicubic prolongation
c (interpolation) is used to
c transfer residual corrections and the pde
```

```

approximation
c           from coarse to fine grids within full
multigrid cycling.
c           (this is the default value whenever
mgopt(1)=0).
c
c *** the default values (2,2,1,3) in the vector mgopt
were chosen for
c       robustness. in some cases v(2,1) cycles with
linear prolongation will
c       give good results with less computation
(especially in two-dimensions).
c       this was the default and only choice in an
earlier version of mudpack
c       (see [1]) and can be set with the integer vector
(1,2,1,1) in mgopt.
c
c *** the schedules for one full multigrid cycle
(iguess=0) using v(2,1)
c       cycles and w(2,1) cycles are depicted for a four
level grid below.
c       the number of relaxation sweeps when each grid is
visited are indicated.
c       the "*" stands for prolongation of the full
approximation and the "."
c       stands for transfer of residuals and residual
corrections within the
c       coarse grid correction algorithm (see below). all
version 5.0.1
c       mudpack solvers use only fully weighted residual
restriction
c
c       one fmg with v(2,1) cycles:
c
c
c       -----
c       -----2-----1-----
c       level 4
c               *
c               .
c               *
c               .
c       -----
c       -----2-----1-----2-----1-----
c       level 3
c               *
c               .
c               *
c               .
c       -----
c       -----2-----1-----2-----1-----2-----1-----

```

```

-----      level 2
C          * . .
C          * . .
C          ---3---3-----3-----3-----
-----      level 1
C
C
C      one fmg with w(2,1) cycles:
C
C      -----
--1---      level 4
C          * .
.
C      -----2-----1-----2-----3-----
1----      level 3
C          * .
C      -----2---1---2---3---1-----2---3---1---2---3---1-
-----      level 2
C          * . . . . . . . . . . . .
C      --6---6-----6---6-----6---6-----6---6-----6---6---
-----      level 1
C
C
C      the form of the "recursive" coarse grid correction
cycling used
c      when kcycle.ge.0 is input is described below in
pseudo-algorithmic
c      language. it is implemented non-recursively in
fortran in mudpack.
c
c      algorithm
cgc(k,l(k),u(k),r(k),kcycle,iprer,ipost,iresw,intpol)
c
c *** approximately solve l(k)*u(k) = r(k) using
multigrid iteration
c *** k is the current grid level
c *** l(k) is the discretized pde operator at level k
c *** u(k) is the initial guess at level k
c *** r(k) is the right hand side at level k
c *** i(k,k-1) is the restriction operator from level k
to level k-1
c *** (the form of i(k,k-1) depends on iresw)
c *** i(k-1,k) is the prolongation operator from level
k-1 to level k

```

```

c *** (the form of i(k-1,k) depends on interpol)
c
c      begin algorithm cgc
c
c ***    pre-relax at level k
c
c      . do (i=1,iprer)
c
c      . . relax(l(k),u(k),r(k))
c
c      . end do
c
c      . if (k > 1) then
c
c ***      restrict the residual from level k to level k-
c 1
c
c      . . r(k-1) = i(k,k-1) (r(k)-l(k)*u(k))
c
c      . . kount = 0
c
c      . . repeat
c
c ***      solve for the residual correction at level k-1
c in u(k-1)
c ***      using algorithm cgc "kcycle" times (this is
c the recursion)
c
c      . . . kount = kount+1
c
c      . . . invoke cgc(k-1,l(k-1),u(k-1),r(k-
c 1),kcycle,iprer,ipost,iresh)
c
c
c      . . until (kount.eq.kcycle)
c
c ***      transfer residual correction in u(k-1) to
c level k
c ***      with the prolongation operator and add to u(k)
c
c      . . u(k) = u(k) + i(k-1,k) (u(k-1))
c
c      . end if
c

```

```
c ***  post relax at level k
c
c      . do (i=1,ipost)
c
c      . . relax(l(k),u(k),r(k))
c
c      . end do
c
c      . return
c
c
c      end algorithm cgc
c
c
c
c ****
c *** output arguments
c ****
c
c
c ... iparm(16) *** set for intl=0 calls only
c
c          on output iparm(16) contains the actual work
c space length
c          required. this will usually be less than
c that given by the
c          simplified formula for length=iparm(15) (see
c as input argument)
c
c
c ... iparm(17) *** set for intl=1 calls only
c
c          on output iparm(17) contains the actual
c number of multigrid cycles
c          between the finest and coarsest grid levels
c used to obtain the
c          approximation when error control (tolmax >
c 0.0) is set.
c
c
c ... fparm(6) *** set for intl=1 calls with fparm(5)
```

```

> 0. only
c
c           on output fparm(6) contains the final
computed maximum relative
c           difference between the last two iterates at
the finest grid level.
c           fparm(6) is computed only if there is error
control (tolmax > 0.0)
c           assume phi1(i,j,k) and phi2(i,j,k) are the
last two computed
c           values for phi(i,j,k) at all points of the
finest grid level.
c           if we define
c
c           phdif = max(cabs(phi2(i,j)-phi1(i,j)))
over all i,j
c
c           and
c
c           phmax = max(cabs(phi2(i,j))) over all i,j
c
c           then
c
c           fparm(6) = phdif/phmax
c
c           is returned whenever phmax > 0.0. in the
degenerate case
c           phmax = 0.0, fparm(6) = phdif is returned.
c
c
c ... work
c
c           on output work contains intermediate values
that must not
c           be destroyed if cud2sp is to be called again
with intl=1
c
c
c ... phi    *** for intl=1 calls only
c
c           on output phi(i,j) contains the approximation
to p(xi,yj)
c           for all mesh points i = 1,...,nx and
j=1,...,ny. the last

```

```

c           computed iterate in phi is returned even if
convergence is
c           not obtained
c
c
c ... ierror
c
c           For intl=iparm(1)=0 initialization calls,
ierror is an
c           error flag that indicates invalid input
arguments when
c           returned positive and nonfatal warnings when
returned
c           negative. Argument checking and
discretization
c           is bypassed for intl=1 calls which can only
return
c           ierror = -1 or 0 or 1.
c
c
c           non-fatal warnings * * *
c
c
c           ==5 if kcycle=mgopt(1) is greater than 2. values
larger than 2 results
c           in an algorithm which probably does far more
computation than
c           necessary. kcycle = 1 (v cycles) or kcycle=2
(w cycles) should
c           suffice for most problems. ierror = -5 is
also set if either
c           iprер = mgopt(2) or ipost=mgopt(3) is greater
than 2. the
c           ierror=-5 flag is overridden by any other
fatal or non-fatal
c           error.
c
c           ==4 if there are dominant nonzero first order
terms in the pde which
c           make it "hyperbolic" at the finest grid level.
numerically, this
c           happens if:
c
c           cabs(cx)*dlx > 2.*cabs(cxx)      (dlx = (xb-

```

```

xa) / (nx-1))
C
C                               (or)
C
C           cabs(cy)*dly > 2.*cabs(cyy)    (dly = (yd-
yc) / (ny-1))
C
C
C           at some fine grid point (xi,yj). if an
adjustment is not made the
C           condition can lead to a matrix coming from the
discretization
C           which is not diagonally dominant and
divergence is possible. since
C           the condition is "likely" at coarser grid
levels for pde's with
C           nonzero first order terms, the adjustments
(actually first order
C           approximations to the pde)
C
C
C           cxx = cmplx(0.5*cabs(cx)*dx,0.0)
C
C           cyy = cmplx(0.5*cabs(cy)*dy,0.0)
C
C
C           (here dx,dy are the x,y mesh sizes of the
subgrid)
C
C           are made when necessary to preserve
convergence. if made
C           at the finest grid level, it can lead to
convergence to an
C           erroneous solution (flagged by ierror = -4).
a possible remedy
C           is to increase resolution. the ierror = -4
flag overrides the
C           nonfatal ierror = -5 flag.
C
C
C           ==3 if the continuous elliptic pde is singular.
this means the
C           boundary conditions are periodic or pure
derivative at all

```

```

c      boundaries and ce(x,y) = (0.0,0.0) for all
x,y. a solution is still
c      attempted but convergence may not occur due
to ill-conditioning
c      of the linear system coming from the
discretization. the
c      ierror = -3 flag overrides the ierror=-4 and
ierror=-5 nonfatal
c      flags.

c
c
c      ==2 unlike cud2 and cud2cr, there is no
ellipticity test with cud2sp
c      so this flag is not set
c
c      ==1 if convergence to the tolerance specified in
tolmax=fparm(5)>0.
c      is not obtained in maxcy=iparm(13) multigrid
cycles between the
c      coarsest (ixp+1,jyq+1) and finest (nx,ny)
grid levels.
c      in this case the last computed iterate is
still returned.
c      the ierror = -1 flag overrides all other
nonfatal flags

c
c
c      no errors * * *
c
c      = 0
c
c      fatal argument errors * * *
c
c      = 1 if intl=iparm(1) is not 0 on the first call or
not 0 or 1
c      on subsequent calls
c
c      = 2 if any of the boundary condition flags
nxa,nxb,nyc,nyd
c      in iparm(2),iparm(3),iparm(4),iparm(5) are not
0,1 or 2
c      or if nxa,nxb or nyc,nyd are not pairwise
zero.

c

```

```

c      = 3 if min0(ixp,jyq) < 2 (ixp = iparm(6), jyq =
iparm(7))
c
c      = 4 if min0(iex,jey) < 1 (iex = iparm(8), jey =
iparm(9)) or
c          if max0(iex,jey) > 50
c
c      = 5 if nx.ne.ixp*2** (iex-1)+1 or
ny.ne.jyq*2** (jey-1)+1
c          (nx = iparm(10), ny = iparm(11))
c
c      = 6 if iguess = iparm(12) is not equal to 0 or 1
c
c      = 7 if maxcy = iparm(13) < 1
c
c      = 8 if method = iparm(14) is not 0,1,2, or 3
c
c      = 9 if length = iparm(15) is too small (see
iparm(16) on output
c          for minimum required work space length)
c
c      =10 if xa >= xb or yc >= yd
c
c (xa=fparm(1),xb=fparm(2),yc=fparm(3),yd=fparm(4))
c
c      =11 if tolmax = fparm(5) < 0.0
c
c      errors in setting multigrid options * * * (see
also ierror=-5)
c
c      =12 if kcycle = mgopt(1) < 0 or
c          if iprer = mgopt(1) < 1 or
c          if ipost = mgopt(3) < 1 or
c          if interpol = mgopt(4) is not 1 or 3
c
c ****
*
c ****
*
c
c      end of cud2sp documentation
c

```

```
C
*****
**
C
*****
**
C
C
```

---

## CUD3

```
C
C      file cud3.d
C
C      * * * * * * * * * * * * * * * * * * * * * *
* * * * * * *
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK version 5.0.1
*
C      *
*
C      *               A Fortran Package of Multigrid
*
C      *
```



```

c
c      subroutine cud3 automatically discretizes and
attempts to compute
c      the second order finite difference approximation
to a COMPLEX
c      3-d linear nonseparable elliptic partial
differential equation
c      on a box.  the approximation is generated on a
uniform grid
c      covering the box (see mesh description below).
boundary
c      conditions may be any combination of mixed,
specified (Dirchlet)
c      or periodic.  the form of the pde solved is . . .
c
c      cxx(x,y,z)*pxx + cyy(x,y,z)*pyy +
czz(z,y,z)*pzz +
c
c      cx(x,y,z)*px + cy(x,y,z)*py + cz(x,y,z)*pz +
c
c      ce(x,y,z)*p(x,y,z) = r(x,y,z)
c
c      here cxx,cyy,czz,cx,cy,cz,ce are the known complex
coefficients
c      of the pde; pxx,pyy,pzz,px,py,pz are the second
and first partial
c      derivatives of the unknown complex solution
function p(x,y,z)
c      with respect to the independent variables x,y,z;
r(x,y,z) is
c      is the known complex right hand side of the
elliptic pde.  cxx,cyy
c      and czz should have real and imaginary parts
positive for all (x,y,z)
c      in the solution region.
c
c
c ... mesh description . . .
c
c      the approximation is generated on a uniform nx by
ny by nz grid.
c      the grid is superimposed on the rectangular
solution region
c

```

```

c           [xa,xb] x [yc,yd] x [ze,zf].
c
c      let
c
c          dlx = (xb-xa) / (nx-1), dly = (yd-yc) / (ny-1),
dlz = (zf-ze) / (nz-1)
c
c      be the uniform grid increments in the x,y,z
directions. then
c
c          xi=xa+(i-1)*dlx,   yj=yc+(j-1)*dly, zk =
ze+(k-1)*dlz
c
c      for i=1,...,nx; j=1,...,ny; k=1,...,nz denote the
x,y,z uniform
c      mesh points.
c
c
c ... language
c
c      fortran90/fortran77
c
c
c ... portability
c
c      mudpack5.0.1 software has been compiled and tested
with Fortran77
c      and Fortran90 on a variety of platforms.
c
c
c ... methods
c
c      details of the methods employed by the solvers in
mudpack are given
c      in [1,9]. [1,2,9] contain performance
measurements on a variety of
c      elliptic pdes (see "references" in the file
"readme"). in summary:
c
c *** discretization and solution (second-order solvers)
(see [1])
c
c      the pde and boundary conditions are automatically
discretized at all
c      grid levels using second-order finite difference

```

```
formula. diagonal
c      dominance at coarser grid levels is maintained in
the presence of
c      nonzero first-order terms by adjusting the second-
order coefficient
c      when necessary. the resulting block tri-diagonal
linear system is
c      approximated using multigrid iteration
[10,11,13,15,16,18]. version
c      5.0.1 of mudpack uses only fully weighted residual
restriction. defaults
c      include cubic prolongation and w(2,1) cycles.
these can be overridden
c      with selected multigrid options (see "mgopt"). error control based on
c      maximum relative differences is available. full multigrid cycling (fmg)
c      or cycling beginning or restarting at the finest grid level can be
c      selected. a menu of relaxation methods including gauss-seidel point,
c      line relaxation(s) (in any combination of directions) and planar
c      relaxation (for three-dimensional anisotropic problems) are provided.
c      all methods use ordering based on alternating points (red/black),
c      lines, or planes for cray vectorization and improved convergence
c      rates [14].
c
c *** higher order solution (fourth-order solvers) (see [9,19,21])
c
c      if the multigrid cycling results in a second-order estimate (i.e.,
c      discretization level error is reached) then this can be improved to a
c      fourth-order estimate using the technique of "deferred corrections."
c      the values in the solution array are used to generate a fourth-order
c      approximation to the truncation error. second-order finite difference
```

```
c      formula are used to approximate third and fourth
c      partial derivatives
c      of the solution function [3].  the truncation
c      error estimate is
c      transferred down to all grid levels using weighted
c      averaging where
c      it serves as a new right hand side.  the default
c      multigrid options
c      are used to compute the fourth-order correction
c      term which is added
c      to the original solution array.
c
c
c ... references (partial)
c
c
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
c      for the Efficient
c      Solution of Linear Elliptic Partial Differential
c      Equations,"
c      Applied Math. and Comput. vol.34, Nov 1989,
c      pp.113-146.
c
c      [2] J. Adams, "FMG Results with the Multigrid
c      Software Package MUDPACK,"
c      proceedings of the fourth Copper Mountain
c      Conference on Multigrid, SIAM,
c      1989, pp.1-12.
c      .
c      .
c      .
c      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
c      Haidvogel, and V. Pizzo,
c      "Applications of Multigrid Software in the
c      Atmospheric Sciences,"
c      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-
c      1458.
c      .
c      .
c      .
c      [9] J. Adams, "Recent Enhancements in MUDPACK, a
c      Multigrid Software
c      package for Elliptic Partial Differential
c      Equations," Applied Math.
```

```
c      and Comp., 1991, vol. 43, May 1991, pp. 79-94.  
c  
c      [10]J. Adams, "MUDPACK-2: Multigrid Software for  
Approximating  
c      Elliptic Partial Differential Equations on Uniform  
Grids with  
c      any Resolution," Applied Math. and Comp., 1993,  
vol. 53, February  
c      1993, pp. 235-249  
c      .  
c      .  
c      .  
c ... argument description  
c  
c  
c  
*****  
*****  
c *** input arguments  
*****  
*****  
c  
*****  
*****  
*****  
c  
c  
c ... iparm  
c  
c      an integer vector of length 23 used to  
efficiently pass  
c      integer arguments. iparm is set internally  
in cud3  
c      and defined as follows . . .  
c  
c  
c ... intl=iparm(1)  
c  
c      an initialization argument. intl=0 must be  
input  
c      on an initial call. in this case input  
arguments will  
c      be checked for errors and the elliptic  
partial differential  
c      equation and boundary conditions will be  
discretized using
```

```
c           second order finite difference formula.  
c  
c ***      An approximation is NOT generated after an  
intl=0 call!  
c           cud3 should be called with intl=1 to  
approximate the elliptic  
c           PDE discretized by the intl=0 call.  intl=1  
should also  
c           be input if cud3 has been called earlier and  
only the  
c           values in in rhs (see below) or gbdy (see  
bndyc below)  
c           or phi (see below) have changed.  This will  
bypass  
c           redundant pde discretization and argument  
checking  
c           and save computational time.  Some examples  
of when  
c           intl=1 calls should be used are:  
c  
c           (0) after a intl=0 argument checking and  
discretization call  
c  
c           (1) cud3 is being recalled for additional  
accuracy.  In  
c           this case iguess=iparm(12)=1 should also  
be used.  
c  
c           (2) cud3 is being called every time step in a  
time dependent  
c           problem (see discussion below) where the  
elliptic operator  
c           does not depend on time.  
c  
c           (3) cud3 is being used to solve the same  
elliptic equation  
c           for several different right hand sides  
(iguess=0 should  
c           probably be used for each new righthand  
side).  
c  
c           intl = 0 must be input before calling with  
intl = 1 when any  
c           of the following conditions hold:
```

```
c
c      (a) the initial call to cud3
c
c      (b) any of the integer arguments other than
c           iguess=iparm(12)
c           or maxcy=iparm(13) or mgopt have changed
c           since the previous
c           call.
c
c      (c) any of the floating point arguments other
c           than tolmax=
c           fparm(5) have changed since the previous
c           call
c
c      (d) any of the coefficients input by coef
c           (see below) have
c           changed since the previous call
c
c      (e) any of the "alfa" coefficients input by
c           bndyc (see below)
c           have changed since the previous call.
c
c      If any of (a) through (e) are true then the
c      elliptic PDE
c      must be discretized or rediscretized.  If
c      none of (a)
c      through (e) holds, calls can be made with
c      intl=1.
c      Incorrect calls with intl=1 will produce
c      erroneous results.
c  ***      The values set in the saved work space "work"
c           (see below) with
c           an intl=0 call must be preserved with
c           subsequent intl=1 calls.
c
c      MUDPACK software performance should be
c      monitored for intl=1
c      calls.  The intl=0 discretization call
c      performance depends
c      primarily on the efficiency or lack of
c      efficiency of the
c      user provided subroutines for pde
c      coefficients and
c      boundary conditions.
```

```

C
C
C ... nxa=iparm(2)
C
C           flags boundary conditions on the (y,z) plane
x=xa
C
C           = 0 if p(x,y,z) is periodic in x on [xa,xb]
C           (i.e., p(x+xb-xa,y,z) = p(x,y,z) for all
x,y,z)
C
C           = 1 if p(xa,y,z) is specified (this must be
input thru phi(1,j,k))
C
C           = 2 if there are mixed derivative boundary
conditions at x=xa
C           (see "bndyc" description below where kbdy =
1)
C
C
C ... nxn=iparm(3)
C
C           flags boundary conditions on the (y,z) plane
x=xb
C
C           = 0 if p(x,y,z) is periodic in x on [xa,xb]
C           (i.e., p(x+xb-xa,y,z) = p(x,y,z) for all
x,y,z)
C
C           = 1 if p(xb,y,z) is specified (this must be
input thru phi(nx,j,k))
C
C           = 2 if there are mixed derivative boundary
conditions at x=xb
C           (see "bndyc" description below where kbdy =
2)
C
C
C ... nyc=iparm(4)
C
C           flags boundary conditions on the (x,z) plane
y=yc
C
C           = 0 if p(x,y,z) is periodic in y on [yc,yd]

```

```

c           (i.e., p(x,y+yd-yc,z) = p(x,y,z) for all
x,y,z)

c           = 1 if p(x,yc,z) is specified (this must be
input thru phi(i,1,k))

c           = 2 if there are mixed derivative boundary
conditions at y=yc
c           (see "bndyc" description below where kbdy =
3)

c
c
c ... nyd=iparm(5)

c
c           flags boundary conditions on the (x,z) plane
y=yd

c
c           = 0 if p(x,y,z) is periodic in y on [yc,yd]
c           (i.e., p(x,y+yd-yc,z) = p(x,y,z) for all
x,y,z)

c           = 1 if p(x,yd,z) is specified (this must be
input thru phi(i,ny,k))

c           = 2 if there are mixed derivative boundary
conditions at y=yd
c           (see "bndyc" description below where kbdy =
4)

c
c
c ... nze=iparm(6)

c
c           flags boundary conditions on the (x,y) plane
z=ze

c
c           = 0 if p(x,y,z) is periodic in z on [ze,zf]
c           (i.e., p(x,y,z+zf-ze) = p(x,y,z) for all
x,y,z

c           = 1 if p(x,y,ze) is specified (this must be
input thru phi(i,j,1))

c           = 2 if there are mixed derivative boundary
conditions at z=ze

```

```

c           (see "bndyc" description below where kbdy =
5)
c
c
c ... nzf=iparm(7)
c
c           flags boundary conditions on the (x,y) plane
z=zf
c
c           = 0 if p(x,y,z) is periodic in z on [ze,zf]
c           (i.e., p(x,y,z+zf-ze) = p(x,y,z) for all
x,y,z

c           = 1 if p(x,y,zf) is specified (this must be
input thru phi(i,j,nz))

c           = 2 if there are mixed derivative boundary
conditions at z=zf
c           (see "bndyc" description below where kbdy =
6)
c
c
c *** grid size arguments
c
c
c ... ixp = iparm(8)
c
c           an integer greater than one which is used in
defining the number
c           of grid points in the x direction (see nx =
iparm(14)). "ixp+1"
c           is the number of points on the coarsest x
grid visited during
c           multigrid cycling. ixp should be chosen as
small as possible.
c           recommended values are the small primes 2 or
3 or (possibly) 5.
c           larger values can reduce multigrid
convergence rates considerably,
c           especially if line relaxation in the x
direction is not used.
c           if ixp > 2 then it should be 2 or a small odd
value since a power
c           of 2 factor of ixp can be removed by

```

```

increasing iex = iparm(11)
c           without changing nx = iparm(14)
c
c
c ... jyq = iparm(9)
c
c           an integer greater than one which is used in
defining the number
c           of grid points in the y direction (see ny =
iparm(15)). "jyq+1"
c           is the number of points on the coarsest y
grid visited during
c           multigrid cycling. jyq should be chosen as
small as possible.
c           recommended values are the small primes 2 or
3 or (possibly) 5.
c           larger values can reduce multigrid
convergence rates considerably,
c           especially if line relaxation in the y
direction is not used.
c           if jyq > 2 then it should be 2 or a small odd
value since a power
c           of 2 factor of jyq can be removed by
increasing jey = iparm(12)
c           without changing ny = iparm(15)
c
c
c ... kzs = iparm(10)
c
c           an integer greater than one which is used in
defining the number
c           of grid points in the z direction (see nz =
iparm(16)). "kzs+1"
c           is the number of points on the coarsest z
grid visited during
c           multigrid cycling. kzs should be chosen as
small as possible.
c           recommended values are the small primes 2 or
3 or (possibly) 5.
c           larger values can reduce multigrid
convergence rates considerably,
c           especially if line relaxation in the z
direction is not used.
c           if kzs > 2 then it should be 2 or a small odd

```

```
value since a power
c          of 2 factor of kzs can be removed by
increasing kez = iparm(13)
c          without changing nz = iparm(16)
c
c
c ... iex = iparm(11)
c
c          a positive integer exponent of 2 used in
defining the number
c          of grid points in the x direction (see nx =
iparm(14)).
c          iex .le. 50 is required. for efficient
multigrid cycling,
c          iex should be chosen as large as possible and
ixp=iparm(8)
c          as small as possible within grid size
constraints when
c          defining nx = iparm(14).
c
c
c ... jey = iparm(12)
c
c          a positive integer exponent of 2 used in
defining the number
c          of grid points in the y direction (see ny =
iparm(15)).
c          jey .le. 50 is required. for efficient
multigrid cycling,
c          jey should be chosen as large as possible and
jyq=iparm(9)
c          as small as possible within grid size
constraints when
c          defining ny = iparm(15).
c
c
c ... kez = iparm(13)
c
c          a positive integer exponent of 2 used in
defining the number
c          of grid points in the z direction (see nz =
iparm(16)).
c          kez .le. 50 is required. for efficient
multigrid cycling,
```

```

c           kez should be chosen as large as possible and
kzr=iparm(10)
c           as small as possible within grid size
constraints when
c           defining nz = iparm(16).
c
c
c ... nx = iparm(14)
c
c           the number of equally spaced grid points in
the interval [xa,xb]
c           (including the boundaries).  nx must have the
form
c
c           nx = ixp*(2** (iex-1)) + 1
c
c           where ixp = iparm(8), iex = iparm(11).
c
c
c ... ny = iparm(15)
c
c           the number of equally spaced grid points in
the interval [yc,yd]
c           (including the boundaries).  ny must have the
form:
c
c           ny = jyq*(2** (jey-1)) + 1
c
c           where jyq = iparm(9), jey = iparm(12).
c
c
c ... nz = iparm(16)
c
c           the number of equally spaced grid points in
the interval [ze,zf]
c           (including the boundaries).  nz must have the
form
c
c           nz = kzr*(2** (kez-1)) + 1
c
c           where kzr = iparm(10), kez = iparm(13)
c
c
c *** example

```

```

c
c           suppose a solution is wanted on a 33 by 65 by
97 grid.  then
c           ixp=2, jyq=4, ksr=6 and iex=jey=kez=5 could be
used.  a better
c           choice would be ixp=jyq=2, ksr=3, and iex=5,
jey=kez=6.
c
c *** note
c
c     let G be the nx by ny by nz fine grid on which the
approximation is
c     generated and let n = max0(iex,jey,kez).  in
mudpack, multigrid
c     cycling is implemented on the ascending chain of
grids
c
c           G(1) < ... < G(k) < ... < G(n) = G.
c
c     each g(k) (k=1,...,n) has mx(k) by my(k) by mz(k)
grid points
c     given by:
c
c           mx(k) = ixp*[2**max0(iex+k-n,1)-1] + 1
c
c           my(k) = jyq*[2**max0(jey+k-n,1)-1] + 1
c
c           mz(k) = ksr*[2**max0(kez+k-n,1)-1] + 1
c
c
c
c ... iguess=iparm(17)
c
c           = 0 if no initial guess to the pde is
provided
c           and/or full multigrid cycling beginning
at the
c           coarsest grid level is desired.
c
c           = 1 if an initial guess to the pde at the
finest grid
c           level is provided in phi (see below).  in
this case
c           cycling beginning or restarting at the

```

```
finest grid
c           is initiated.

c
c *** comments on iguess = 0 or 1 . . .
c
c
c     setting iguess=0 forces full multigrid or "fmg"
c     cycling. phi
c     must be initialized at all grid points. it can be
c     set to zero at
c     non-Dirchlet grid points if nothing better is
c     available. the
c     values set in phi when iguess = 0 are passed and
c     down and serve
c     as an initial guess to the pde at the coarsest
c     grid level where
c     multigrid cycling commences.

c
c     if iguess = 1 then the values input in phi are an
c     initial guess to the
c     pde at the finest grid level where cycling begins.
c     this option should
c     be used only if a "very good" initial guess is
c     available (as, for
c     example, when restarting from a previous iguess=0
c     call).

c
c *** time dependent problems . . .
c
c     assume we are solving an elliptic pde every time
c     step in a
c     marching problem of the form:

c
c     l(p(t)) = r(t)
c
c     where the differential operator "l" has no time
c     dependence,
c     "p(t)" is the solution and "r(t)" is the right
c     hand side at
c     current time "t". let "dt" be the increment
c     between time steps.
c     then p(t) can be used as an initial guess to
c     p(t+dt) with
c     intl = 1 when solving
```

```

c
c           l(p(t+dt)) = r(t+dt).
c
c           after the first two time steps, rather than
c continue, it would
c           be better to define the "correction" term:
c
c           e(t,dt) = p(t+dt) - p(t)
c
c           this clearly satisfies the equation
c
c           l(e(t,dt)) = r(t+dt) - r(t).
c
c           this should be solved with iguess = 0 and intl =
1. boundary
c           conditions for e(t,dt) are obtained from the
boundary conditions
c           for p(t) by subtracting given values at t from
given values at
c           t+dt. for example if
c
c           d(p(t))/dx = f(t), d(p(t+dt))/dx = f(t+dt)
c
c           at some x boundary then e(t,dt) satisfies the
derivative
c           boundary condition
c
c           d(e(t,dt))/dx = f(t+dt) - f(t).
c
c           e(t,dt) can be preset to 0.0 (at non-Dirchlet
points) or (if p(t-dt)
c           is saved) to p(t)-p(t-dt). with iguess = 0, these
values will serve
c           as an initial guess to e(t,dt) at the coarsest
grid level. this
c           approach has the advantage that a full sequence of
multigrid cycles,
c           beginning at the coarsest grid level, is invoked
every time step in
c           solving for e(t,dt). a few digits of accuracy in
e(t,dt), which is
c           ordinarily much smaller than p(t), will yield
several more digits of
c           accuracy in the final approximation:

```

```

c
c          p(t+dt) = p(t) + e(t,dt).
c
c      using this approach to integrate in time will give
more accuracy
c      then using p(t) as an initial guess to p(t+dt) for
all time steps.
c      it does require additional storage.
c
c      if the differential operator "l" has time
dependence (either thru
c      the coefficients in the pde or the coefficients in
the derivative
c      boundary conditions) then use p(t) as an initial
guess to p(t+dt)
c      when solving
c
c          l(t+dt) (p(t+dt)) = r(t+dt)
c
c      with intl = 0 for all time steps (the
discretization must be repeated
c      for each new "t"). either iguess = 0 (p(t) will
then be an initial
c      guess at the coarsest grid level where cycles will
commence) or
c      iguess = 1 (p(t) will then be an initial guess at
the finest grid
c      level where cycles will remain fixed) can be
tried.
c
c
c ... maxcy = iparm(18)
c
c          the exact number of cycles executed between
the finest
c          (nx by ny by nz) and the coarsest ((ixp+1) by
(jyq+1) by
c          (kzr+1)) grid levels when tolmax=fparm(7)=0.0
(no error
c          control). when tolmax=fparm(7).gt.0.0 is
input (error control)
c          then maxcy is a limit on the number of cycles
between the
c          finest and coarsest grid levels. in any

```

```
case, at most
c           maxcy*(iprert+ipost) relaxation sweeps are
performed at the
c           finest grid level (see
iprer=mgopt(2),ipost=mgopt(3) below)
c           when multigrid iteration is working
"correctly" only a few
c           cycles are required for convergence. large
values for maxcy
c           should not be required.

c
c
c ... method = iparm(19)
c
c           this sets the method of relaxation (all
relaxation
c           schemes in mudpack use red/black type
ordering)
c
c           = 0 for gauss-seidel pointwise relaxation
c
c           = 1 for line relaxation in the x direction
c
c           = 2 for line relaxation in the y direction
c
c           = 3 for line relaxation in the z direction
c
c           = 4 for line relaxation in the x and y
direction
c
c           = 5 for line relaxation in the x and z
direction
c
c           = 6 for line relaxation in the y and z
direction
c
c           = 7 for line relaxation in the x,y and z
direction
c
c           = 8 for x,y planar relaxation
c
c           = 9 for x,z planar relaxation
c
c           =10 for y,z planar relaxation
```

```

c
c *** if nxa = 0 and nx = 3 at a grid level where line
relaxation in the x
c      direction is flagged then it will be replaced by
gauss-seidel point
c      relaxation at that grid level.
c
c *** if nyc = 0 and ny = 3 at a grid level where line
relaxation in the y
c      direction is flagged then it will be replaced by
gauss-seidel point
c      relaxation at that grid level.
c
c *** if nze = 0 and nz = 3 at a grid level where line
relaxation in the z
c      direction is flagged then it will be replaced by
gauss-seidel point
c      relaxation at that grid level.
c
c      these adjustments are necessary since the
simultaneous tri-diagonal
c      solvers used with line periodic relaxation must
have n > 2 where n
c      is number of unknowns (excluding the periodic
point).

c *** choice of method
c
c      this is very important for efficient convergence.
in some cases
c      experimentation may be required.
c
c      let fx represent the quantity cxx(x,y,z)/dlx**2
over the solution box
c
c      let fy represent the quantity cyy(x,y,z)/dly**2
over the solution box
c
c      let fz represent the quantity czz(x,y,z)/dlz**2
over the solution box
c
c      (0) if fx,fy,fz are roughly the same size and do
not vary too
c          much choose method = 0.  if this fails try

```

```
method = 7.  
c  
c      (1) if fx is much greater than fy,fz and fy,fz  
are roughly the same  
c          size choose method = 1  
c  
c      (2) if fy is much greater than fx,fz and fx,fz  
are roughly the same  
c          size choose method = 2  
c  
c      (3) if fz is much greater than fx,fy and fx,fy  
are roughly the same  
c          size choose method = 3  
c  
c      (4) if fx,fy are roughly the same and both are  
much greater than fz  
c          try method = 4.  if this fails try method = 8  
c  
c      (5) if fx,fz are roughly the same and both are  
much greater than fy  
c          try method = 5.  if this fails try method = 9  
c  
c      (6) if fy,fz are roughly the same and both are  
much greater than fx  
c          try method = 6.  if this fails try method =  
10  
c  
c      (7) if fx,fy,fz vary considerably with none  
dominating try method = 7  
c  
c      (8) if fx and fy are considerably greater than fz  
but not necessarily  
c          the same size (e.g., fx=1000.,fy=100.,fz=1.)  
try method = 8  
c  
c      (9) if fx and fz are considerably greater than fy  
but not necessarily  
c          the same size (e.g., fx=10.,fy=1.,fz=1000.)  
try method = 9  
c  
c      (10)if fy and fz are considerably greater than fx  
but not necessarily  
c          the same size (e.g., fx=1.,fy=100.,fz=10.)  
try method = 10
```

```

c
c
c ... meth2 = iparm(20) determines the method of
relaxation used in the planes
c           when method = 8 or 9 or 10.
c
c
c           as above, let fx,fy,fz represent the
quantities cxx/dlx**2,
cyy/dly**2,czz/dlz**2 over the box.
c
c           (if method = 8)
c
c           = 0 for gauss-seidel pointwise relaxation
c             in the x,y plane for each fixed z
c           = 1 for line relaxation in the x direction
c             in the x,y plane for each fixed z
c           = 2 for line relaxation in the y direction
c             in the x,y plane for each fixed z
c           = 3 for line relaxation in the x and y
direction
c             in the x,y plane for each fixed z
c
c           (1) if fx,fy are roughly the same and vary
little choose meth2 = 0
c           (2) if fx is much greater than fy choose
meth2 = 1
c           (3) if fy is much greater than fx choose
meth2 = 2
c           (4) if none of the above or meth2 = 0 fails
choose meth2 = 3
c
c           (if method = 9)
c
c           = 0 for gauss-seidel pointwise relaxation
with red/black ordering
c             in the x,z plane for each fixed y
c           = 1 for simultaneous line relaxation in the x
direction
c             of the x,z plane for each fixed y
c           = 2 for simultaneous line relaxation in the z
direction
c             of the x,z plane for each fixed y
c           = 3 for simultaneous line relaxation in the x

```

```

and z direction
c          of the x,z plane for each fixed y
c
c          (1) if fx,fz are roughly the same and vary
little choose meth2 = 0
c          (2) if fx is much greater then fz choose
meth2 = 1
c          (3) if fz is much greater then fx choose
meth2 = 2
c          (4) if none of the above or meth2 = 0 fails
choose meth2 = 3
c
c          (if method = 10)
c
c          = 0 for gauss-seidel pointwise relaxation
with red/black ordering
c          in the y,z plane for each fixed x
c          = 1 for simultaneous line relaxation in the y
direction
c          of the y,z plane for each fixed x
c          = 2 for simultaneous line relaxation in the z
direction
c          of the y,z plane for each fixed x
c          = 3 for simultaneous line relaxation in the y
and z direction
c          of the y,z plane for each fixed x
c
c          (1) if fy,fz are roughly the same and vary
little choose meth2 = 0
c          (2) if fy is much greater then fz choose
meth2 = 1
c          (3) if fz is much greater then fy choose
meth2 = 2
c          (4) if none of the above or meth2 = 0 fails
choose meth2 = 3
c
c
c ... length = iparm(21)
c
c          the length of the work space provided in
vector work.
c
c          let isx = 3 if method = 1,4,5 or 7 and
nxa.ne.0

```

```

c           let isx = 5 if method = 1,4,5 or 7 and
nxa.eq.0
c           let isx = 0 if method has any other value
c
c           let jsy = 3 if method = 2,4,6 or 7 and
nyc.ne.0
c           let jsy = 5 if method = 2,4,6 or 7 and
nyc.eq.0
c           let jsy = 0 if method has any other value
c
c           let ksz = 3 if method = 3,5,6 or 7 and
nze.ne.0
c           let ksz = 5 if method = 3,5,6 or 7 and
nze.eq.0
c           let ksz = 0 if method has any other value
c
c
c           then (for method .le.7)
c
c           (1)    length =
(nx+2)*(ny+2)*(nz+2)*(10+isx+jsy+ksz)
c
c           or (for method.gt.7)
c
c           (2)    length = 14*(nx+2)*(ny+2)*(nz+2)
c
c           will usually but not always suffice. The
exact minimal length depends,
c           in a complex way, on the grid size arguments
and method chosen.
c ***      It can be predetermined for the current input
arguments by calling
c           cud3 with length=iparm(21)=0 and printing
iparm(22) or (in f90)
c           dynamically allocating the work space using
the value in iparm(22)
c           in a subsequent cud3 call.
c
c ... fparm
c
c           a floating point vector of length 8 used to
efficiently
c           pass floating point arguments. fparm is set
internally

```

```
c           in cud3 and defined as follows . . .
c
c
c ... xa=fparm(1), xb=fparm(2)
c
c           the range of the x independent variable. xa
must
c           be less than xb
c
c
c ... yc=fparm(3), yd=fparm(4)
c
c           the range of the y independent variable. yc
must
c           be less than yd.
c
c
c ... ze=fparm(5), zf=fparm(6)
c
c           the range of the z independent variable. ze
must
c           be less than zf.
c
c
c ... tolmax = fparm(5)
c
c           when input positive, tolmax is a maximum
relative error tolerance
c           used to terminate the relaxation iterations.
assume phi1(i,j,k)
c           and phi2(i,j,k) are the last two computed
approximations at all
c           grid points of the finest grid level. if we
define
c
c           phdif = max(cabs(phi2(i,j,k)-
phi1(i,j,k))) for all i,j,k
c
c           and
c
c           phmax = max(cabs(phi2(i,j,k))) for all
i,j,k
c
c           then "convergence" is considered to have
```

```
occurred if and only if
c
c          phdif/phmax < tolmax.
c
c
c          if tolmax=fparm(5)=0 is input then there is
no error control
c          and maxcy cycles from the finest grid level
are executed. maxcy
c          is a limit which cannot be exceeded even with
error control.
c          *** calls with tolmax=0.0, when appropriate
because of known
c          convergence behavior, are more efficient than
calls with tolmax
c          positive (i.e., if possible DO NOT use error
control!).
c
c
c ... work
c
c          a complex one dimensional array that must be
provided for work space.
c          see length = iparm(21). the values in work
must be preserved
c          if cud3 is called again with
intl=iparm(1).ne.0 or if cud34
c          is called to improve accuracy.
c
c
c ... bndyc
c
c          a subroutine with arguments
(kbdy,xory,yorz,alfa,gbdy) .
c          which are used to input mixed boundary
conditions to cud3.
c          the boundaries are numbered one thru six and
the form of
c          conditions are described below.
c
c
c          (1) the kbdy=1 boundary
c
c          this is the (y,z) plane x=xa where nxa=iparm(2) =
```

```

2 flags
c      a mixed boundary condition of the form
c
c      dp/dx + alfxa(y,z)*p(xa,y,z) = gbdxa(y,z)
c
c      in this case kbdy=1,xory=y,yorz=z will be input to
bndyc and
c      alfa,gbdy corresponding to alfxa(y,z),gbdxa(y,z)
must be returned.
c
c
c      (2) the kbdy=2 boundary
c
c      this is the (y,z) plane x=xb where nxb=iparm(3) =
2 flags
c      a mixed boundary condition of the form
c
c      dp/dx + alfxb(y,z)*p(xb,y,z) = gbdxb(y,z)
c
c      in this case kbdy=2,xory=y,yorz=z will be input to
bndyc and
c      alfa,gbdy corresponding to alfxb(y,z),gbdxb(y,z)
must be returned.
c
c
c      (3) the kbdy=3 boundary
c
c      this is the (x,z) plane y=yc where nyc=iparm(4) =
2 flags
c      a mixed boundary condition of the form
c
c      dp/dy + alfyc(x,z)*p(x,yc,z) = gbdyc(x,z)
c
c      in this case kbdy=3,xory=x,yorz=z will be input to
bndyc and
c      alfa,gbdy corresponding to alfyc(x,z),gbdyc(x,z)
must be returned.
c
c
c      (4) the kbdy=4 boundary
c
c      this is the (x,z) plane y=yd where nyd=iparm(5) =
2 flags
c      a mixed boundary condition of the form

```

```

C
C      dp/dy + alfyd(x,z)*p(x,yd,z) = gbdyd(x,z)
C
C      in this case kbdy=4,xory=x,yorz=z will be input to
bndyc and
C      alfa,gbdy corresponding to alfyd(x,z),gbdyd(x,z)
must be returned.

C
C
C      (5) the kbdy=5 boundary
C
C      this is the (x,y) plane z=ze where nze=iparm(6) =
2 flags
C      a mixed boundary condition of the form
C
C      dp/dz + alfze(x,y)*p(x,y,ze) = gbdze(x,y)
C
C      in this case kbdy=5,xory=x,yorz=y will be input to
bndyc and
C      alfa,gbdy corresponding to alfze(x,y),gbdze(x,y)
must be returned.

C
C
C      (6) the kbdy=6 boundary
C
C      this is the (x,y) plane z=zf where nzf=iparm(7) =
2 flags
C      a mixed boundary condition of the form
C
C      dp/dz + alfzf(x,y)*p(x,y,zf) = gbdzf(x,y)
C
C      in this case kbdy=6,xory=x,yorz=y will be input to
bndyc and
C      alfa,gbdy corresponding to alfzf(x,y),gbdzf(x,y)
must be returned.

C
C
C *** alfxa,alfyc,alfze nonpositive and
alfxb,alfyd,alfze nonnegative
C      will help maintain matrix diagonal dominance
during discretization
C      aiding convergence.

C
C *** bndyc must provide the mixed boundary condition

```

```

c      values in correspondence with those flagged in
iparm(2)
c      thru iparm(7). if all boundaries are specified
then
c      cud3 will never call bndyc. even then it must be
entered
c      as a dummy subroutine. bndyc must be declared
c      external in the routine calling cud3. the actual
c      name chosen may be different.
c
c
c ... coef
c
c      a subroutine with arguments
(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
c      which provides the known complex coefficients
for the elliptic pde
c      at any grid point (x,y,z). the name chosen in
the calling routine
c      may be different where the coefficient routine
must be declared
c      external.
c
c ... rhs
c
c      a complex array dimensioned nx by ny by nz
which contains
c      the given right hand side values on the
uniform 3-d mesh.
c      rhs(i,j,k) = r(xi,yj,zk) for i=1,...,nx and
j=1,...,ny
c      and k=1,...,nz.
c
c ... phi
c
c      a complex array dimensioned nx by ny by nz .
on input phi must
c      contain specified boundary values and an
initial guess
c      to the solution if flagged (see
iguess=iparm(17)=1). for
c      example, if nyd=iparm(5)=1 then phi(i,ny,k)
must be set
c      equal to p(xi,yd,zk) for i=1,...,nx and

```

```

k=1,...,nz prior to
c           calling cud3.  the specified values are
preserved by cud3.
c
c ***      if no initial guess is given (iguess=0) then
phi must still
c           be initialized at non-Dirchlet grid points
(this is not
c           checked). these values are projected down and
serve as an initial
c           guess to the pde at the coarsest grid level.
set phi to 0.0 at
c           nonDirchlet grid points if nothing better is
available.
c
c
c ... mgopt
c
c           an integer vector of length 4 which allows
the user to select
c           among various multigrid options. if
mgopt(1)=0 is input then
c           a default set of multigrid arguments (chosen
for robustness)
c           will be internally selected and the
remaining values in mgopt
c           will be ignored. if mgopt(1) is nonzero
then the arguments
c           in mgopt are set internally and defined as
follows: (see the
c           basic coarse grid correction algorithm
below)
c
c
c     kcycle = mgopt(1)
c
c           = 0 if default multigrid options are to be
used
c
c           = 1 if v cycling is to be used (the least
expensive per cycle)
c
c           = 2 if w cycling is to be used (the
default)

```

```

c
c          > 2 if more general k cycling is to be used
c          *** warning--values larger than 2 increase
c          the execution time per cycle
considerably and
c          result in the nonfatal error ierror =
-5
c          which indicates inefficient multigrid
cycling.
c
c      iprер = mgopt(2)
c
c          the number of "pre-relaxation" sweeps
executed before the
c          residual is restricted and cycling is
invoked at the next
c          coarser grid level (default value is 2
whenever mgopt(1)=0)
c
c      ipost = mgopt(3)
c
c          the number of "post relaxation" sweeps
executed after cycling
c          has been invoked at the next coarser grid
level and the residual
c          correction has been transferred back
(default value is 1
c          whenever mgopt(1)=0) .
c
c *** if iprер, ipost, or (especially) kcycе is greater
than 2
c          than inefficient multigrid cycling has probably
been chosen and
c          the nonfatal error (see below) ierror = -5 will be
set. note
c          this warning may be overridden by any other
nonzero value
c          for ierror.
c
c      intpol = mgopt(4)
c
c          = 1 if multilinear prolongation
(interpolation) is used to
c          transfer residual corrections and the pde

```

```

approximation
c           from coarse to fine grids within full
multigrid cycling.
c
c           = 3 if multicubic prolongation
(interpolation) is used to
c           transfer residual corrections and the pde
approximation
c           from coarse to fine grids within full
multigrid cycling.
c           (this is the default value whenever
mgopt(1)=0).
c
c *** the default values (2,2,1,3) in the vector mgopt
were chosen for
c   robustness. in some cases v(2,1) cycles with
linear prolongation will
c   give good results with less computation
(especially in two-dimensions).
c   this was the default and only choice in an
earlier version of mudpack
c   (see [1]) and can be set with the integer vector
(1,2,1,1) in mgopt.
c
c *** the schedules for one full multigrid cycle
(iguess=0) using v(2,1)
c   cycles and w(2,1) cycles are depicted for a four
level grid below.
c   the number of relaxation sweeps when each grid is
visited are indicated.
c   the "*" stands for prolongation of the full
approximation and the "."
c   stands for transfer of residuals and residual
corrections within the
c   coarse grid correction algorithm (see below).
c
c   one fmg with v(2,1) cycles:
c
c
c   -----
-----2-----1-
-----      level 4
c           * . .
c           * . .
c   -----2-----1-----2-----1-----

```

```

-----      level 3
C          *
C          *
C          -----2-----1-----2-----1-----2-----1-----
-----      level 2
C          *
C          *
C          ---3---3-----3-----3-----
-----      level 1
C
C
C      one fmg with w(2,1) cycles:
C
C      -----
--1--      level 4
C          *
.
C      -----2-----1-----2-----3-----
1----      level 3
C          *
C      ----2---1---2---3---1-----2---3---1---2---3---1-
-----      level 2
C          * . . . . . . . . . . . .
C      --6---6-----6---6-----6---6-----6---6-----6---6---
-----      level 1
C
C
C      the form of the "recursive" coarse grid correction
cycling used
C      when kcyle.ge.0 is input is described below in
pseudo-algorithmic
C      language. it is implemented non-recursively in
fortran in mudpack.
C
C      algorithm
cgc(k,l(k),u(k),r(k),kcycle,iprer,ipost,iressw,intpol)
C
C *** approximately solve l(k)*u(k) = r(k) using
multigrid iteration
C *** k is the current grid level
C *** l(k) is the discretized pde operator at level k
C *** u(k) is the initial guess at level k
C *** r(k) is the right hand side at level k
C *** i(k,k-1) is the restriction operator from level k

```

```

to level k-1
c *** (the form of i(k,k-1) depends on iresw)
c *** i(k-1,k) is the prolongation operator from level
k-1 to level k
c *** (the form of i(k-1,k) depends on interpol)
c
c      begin algorithm cgc
c
c *** pre-relax at level k
c
c      . do (i=1,iprer)
c
c      . . relax(l(k),u(k),r(k))
c
c      . end do
c
c      . if (k > 1) then
c
c ***      restrict the residual from level k to level k-
1
c
c      . . r(k-1) = i(k,k-1) (r(k)-l(k)*u(k))
c
c      . . kount = 0
c
c      . . repeat
c
c ***      solve for the residual correction at level k-1
in u(k-1)
c ***      using algorithm cgc "kcycle" times (this is
the recursion)
c
c      . . . kount = kount+1
c
c      . . . invoke cgc(k-1,l(k-1),u(k-1),r(k-
1),kcycle,iprer,ipost,iresw)
c
c
c      . . until (kount.eq.kcycle)
c
c ***      transfer residual correction in u(k-1) to
level k
c ***      with the prolongation operator and add to u(k)
c

```

```

C      . . u(k) = u(k) + i(k-1,k) (u(k-1))
C
C      . end if
C
C *** post relax at level k
C
C      . do (i=1,ipost)
C
C      . . relax(l(k),u(k),r(k))
C
C      . end do
C
C      . return
C
C
C      end algorithm cgc
C
C
C ****output
arguments*****
***

C ****
***** arguments ****
***** iparm(22)
C
C      on output iparm(22) contains the actual
complex work space length
C      required for the current grid sizes and
method. This value
C      will be computed and returned even if
iparm(21) is less then
C          iparm(22) (see ierror=9).
C
C
C ... iparm(23)
C
C      if error control is selected (tolmax =
fparm(7) .gt. 0.0) then
C          on output iparm(23) contains the actual
number of cycles executed

```

```

c           between the coarsest and finest grid levels
in obtaining the
c           approximation in phi.  the quantity
(iprert+ipost)*iparm(23) is
c           the number of relaxation sweeps performed at
the finest grid level.
c
c
c ... fparm(8)
c
c           on output fparm(8) contains the final
computed maximum relative
c           difference between the last two iterates at
the finest grid level.
c           fparm(8) is computed only if there is error
control (tolmax.gt.0.)
c           assume phil(i,j,k) and phi2(i,j,k) are the
last two computed
c           values for phi(i,j,k) at all points of the
finest grid level.
c           if we define
c
c           phdif = max(cabs(phi2(i,j,k)-phil(i,j,k)))
for all i,j,k
c
c           and
c
c           phmax = max(cabs(phi2(i,j,k))) for all
i,j,k
c
c           then
c
c           fparm(8) = phdif/phmax
c
c           is returned whenever phmax.gt.0.0.  in the
degenerate case
c           phmax = 0.0, fparm(8) = phdif is returned.
c
c
c ...
c           work
c
c           on output work contains intermediate values
that must not be

```

```

c           destroyed if cud3 is to be called again with
iparm(1)=1 or
c           if cud34 is to be called to improve the
estimate to fourth
c           order.
c
c ... phi
c
c           on output phi(i,j,k) contains the
approximation to
c           p(xi,yj,zk) for all mesh points i=1,...,nx;
j=1,...,ny;
c           k=1,...,nz. the last computed iterate in phi
is returned
c           even if convergence is not obtained (ierror=-
1)
c
c ... ierror
c
c           For intl=iparm(1)=0 initialization calls,
ierror is an
c           error flag that indicates invalid input
arguments when
c           returned positive and nonfatal warnings when
returned
c           negative. Argument checking and
discretization
c           is bypassed for intl=1 calls which can only
return
c           ierror = -1 or 0 or 1.
c
c
c           non-fatal warnings * * *
c
c
c           ==5 if kcycle=mgopt(1) is greater than 2. values
larger than 2 results
c           in an algorithm which probably does far more
computation than
c           necessary. kcycle = 1 (v cycles) or kcycle=2
(w cycles) should
c           suffice for most problems. ierror = -5 is
also set if either
c           iprer = mgopt(2) or ipost=mgopt(3) is greater

```

```

than 2. the
c           ierror=-5 flag is overridden by any other
fatal or non-fatal
c           error.
c
c      ==-4 if there are dominant nonzero first order
terms in the pde which
c           make it "hyperbolic" at the finest grid level.
numerically, this
c           happens if:
c
c           cabs(cx)*dlx > 2.*cabs(cxx)    (dlx = (xb-
xa) / (nx-1))
c
c                           (or)
c
c           cabs(cy)*dly > 2.*cabs(cyy)    (dly = (yd-
yc) / (ny-1))
c
c
c           at some fine grid point (xi,yj). if an
adjustment is not made the
c           condition can lead to a matrix coming from the
discretization
c           which is not diagonally dominant and
divergence is possible. since
c           the condition is "likely" at coarser grid
levels for pde's with
c           nonzero first order terms, the adjustments
(actualy first order
c           approximations to the pde)
c
c           if cabs(cxx) < 0.5*cabs(cx)*dx then
c               cxx = cmplx(0.5*cabs(cx)*dx,0.0)
c
c                           (and)
c
c           if (cabs(cyy) < 0.5*cabs(cy)*dy then
c               cyy = cmplx(0.5*cabs(cy)*dy,0.0)
c
c
c           (here dx,dy are the x,y mesh sizes of the
subgrid)
c

```

```

c      are made to preserve convergence of multigrid
iteration. if made
c      at the finest grid level, it can lead to
convergence to an
c      erroneous solution (flagged by ierror = -4).
a possible remedy
c      is to increase resolution. the ierror = -4
flag overrides the
c      nonfatal ierror = -5 flag.

c
c
c      ==3 if the continuous elliptic pde is singular.
this means the
c      boundary conditions are periodic or pure
derivative at all
c      boundaries and ce(x,y) = 0.0 for all x,y. a
solution is still
c      attempted but convergence may not occur due
to ill-conditioning
c      of the linear system coming from the
discretization. the
c      ierror = -3 flag overrides the ierror=-4 and
ierror=-5 nonfatal
c      flags.

c
c
c      ==2 if the pde is not elliptic (i.e.,
cxx*cyy.le.0.0 for some (xi,yj))
c      in this case a solution is still attempted
although convergence
c      may not occur due to ill-conditioning of the
linear system.
c      the ierror = -2 flag overrides the ierror=-
5,-4,-3 nonfatal
c      flags.

c
c
c      ==1 if convergence to the tolerance specified in
tolmax=fparm(5)>0.
c      is not obtained in maxcy=iparm(13) multigrid
cycles between the
c      coarsest (ixp+1,jyq+1) and finest (nx,ny)
grid levels.
c      in this case the last computed iterate is

```

```
still returned.  
C           the ierror = -1 flag overrides all other  
nonfatal flags  
C  
C  
C       no errors * * *  
C  
C       = 0  
C  
C       fatal argument errors * * *  
C  
C       = 1 if intl=iparm(1) is not 0 on initial call or  
not 0 or 1  
C           on subsequent calls  
C  
C       = 2 if any of the boundary condition flags  
nxa,nxb,nyc,nyd,nze,nzf  
C           in iparm(2) through iparm(7) is not 0,1 or 2 or  
if  
C           (nxa,nxb) or (nyc,nyd) or (nze,nzf) are not  
pairwise zero.  
C  
C       = 3 if mino(ixp,jyq,kzr) < 2  
(ixp=iparm(8),jyq=iparm(9),kzr=iparm(10))  
C  
C       = 4 if min0(iex,jey,kez) < 1  
(iex=iparm(11),jey=iparm(12),kez=iparm(13))  
C           or if max0(iex,jey,kez) > 50  
C  
C       = 5 if nx.ne.ixp*2** (iex-1)+1 or if  
ny.ne.jyq*2** (jey-1)+1 or  
C           if nz.ne.kzr*2** (kez-1)+1  
(nx=iparm(14),ny=iparm(15),nz=iparm(16))  
C  
C       = 6 if iguess = iparm(17) is not equal to 0 or 1  
C  
C       = 7 if maxcy = iparm(18) < 1 (large values for  
maxcy should not be used)  
C  
C       = 8 if method = iparm(19) is less than 0 or  
greater than 10 or  
C           if meth2 = iparm(20) is not 0 or 1 or 2 or 3  
when method > 7.  
C
```

```

c      = 9 if length = iparm(20) is too small (see
iparm(21) on output
c          for minimum required work space length)
c
c      =10 if any of:  xa < xb or yc < yd or ze < zf is
false
c
(xa=fparm(1),xb=fparm(2),yc=fparm(3),yd=fparm(4),ze=fpar
m(5),zf=fparm(6))

c
c      =11 if tolmax = fparm(7) < 0.0
c
c      errors in setting multigrid options * * * (see
also ierror=-5)
c
c      =12 if kcycle = mgopt(1) < 0 or
c          if iprer = mgopt(2) < 1 or
c          if ipost = mgopt(3) < 1 or
c          if interpol = mgopt(4) is not 1 or 3
c
c
*****
*
c
*****
*
c
*****
c      end of cud3 documentation
c
c
*****
**
c
*****
**
c
c

```

```
C  
C      file cud34.d  
C  
C      * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C      *  
*  
C      *          copyright (c) 2008 by UCAR  
*  
C      *  
*  
C      *          University Corporation for Atmospheric  
Research      *  
C      *  
*  
C      *          all rights reserved  
*  
C      *  
*  
C      *          MUDPACK version 5.0.1  
*  
C      *  
*  
C      *          A Fortran Package of Multigrid  
*  
C      *  
*  
C      *          Subroutines and Example Programs  
*  
C      *  
*  
C      *          for Solving Elliptic Partial Differential  
Equations      *  
C      *  
*  
C      *          by  
*  
C      *  
*  
C      *          John Adams  
*  
C      *  
*
```

```
C      *
      *
C      *
      *
C      *           the National Center for Atmospheric
Research          *
C      *
      *
C      *           Boulder, Colorado (80307)
U.S.A.          *
C      *
      *
C      *           which is sponsored by
      *
C      *
      *
C      *           the National Science Foundation
      *
C      *
      *
C      *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *
*   *   *   *   *   *
C
C ... file cud34.d
C
C     contains documentation for subroutine
cud34(work,phi,ierror)
C     A sample fortran driver is file "tcud34.f".
C
C ... required MUDPACK files
C
C     cud3.f, cudcom.f, cud3ln.f, cud3pn.f
C
C ... purpose
C
C     cud34 attempts to improve the estimate in phi,
obtained by calling
C     cud3, from second to fourth order accuracy. see
the file "cud3.d"
C     for a detailed discussion of the elliptic pde
approximated and
C     arguments "work,phi" which are also part of the
argument list for
C     cud3.
```

```
C
C ... assumptions
C
C      * phi contains a second-order approximation from
an earlier cud3 call
C
C      * arguments "work,phi" are the same used in
calling cud3
C
C      * "work,phi" have not changed since the last call
to cud3
C
C      * the finest grid level contains at least 6
points in each direction
C
C
C *** warning
C
C      if the first assumption is not true then a fourth
order approximation
C      cannot be produced in phi. the last assumption
(adequate grid size)
C      is the only one checked. failure in any of the
others can result in
C      in an undetectable error.
C
C ... language
C
C      fortran90/fortran77
C
C ... portability
C
C      mudpack5.0.1 software has been compiled and tested
with fortran77
C      and fortran90 on a variety of platforms.
C
C ... methods
C
C      details of the methods employeed by the solvers in
mudpack are given
C      in [1,9]. [1,2,9] contain performance
measurements on a variety of
C      elliptic pdes (see "references" in the file
"readme"). in summary:
```

```
c
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
estimate (i.e.,
c      discretization level error is reached) then this
can be improved to a
c      fourth-order estimate using the technique of
"deferred corrections"
c      the values in the solution array are used to
generate a fourth-order
c      approximation to the truncation error. second-
order finite difference
c      formula are used to approximate third and fourth
partial derivatives
c      of the solution function [3]. the truncation
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side. the default
multigrid options
c      are used to compute the fourth-order correction
term which is added
c      to the original solution array.

c
c
c ... references (partial)
c
c
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
c      Solution of Linear Elliptic Partial Differential
Equations,"
c      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
c
c      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
c      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
c      1989, pp.1-12.
c      .
```

```
C .
C .
C      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
Haidvogel, and V. Pizzo,
C      "Applications of Multigrid Software in the
Atmospheric Sciences,"
C      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-
1458.
C .
C .
C .
C      [9] J. Adams, "Recent Enhancements in MUDPACK, a
Multigrid Software
C      package for Elliptic Partial Differential
Equations," Applied Math.
C      and Comp., 1991, vol. 43, May 1991, pp. 79-94.
C
C      [10] J. Adams, "MUDPACK-2: Multigrid Software for
Approximating
C      Elliptic Partial Differential Equations on Uniform
Grids with
C      any Resolution," Applied Math. and Comp., 1993,
vol. 53, February
C      1993, pp. 235-249
C .
C .
C .
C
C ... error argument
C
C      = 0 if no error is detected
C
C      = 30 if min0(nx,ny) < 6 where nx,ny are the fine
grid sizes
C          in the x,y directions.
C
C
C
*****
```

```
C      end of cud34 documentation
C
C
*****
*****
```

```
C
```

```
*****
```

```
C
```

```
*****
```

```
*****
```

```
C
```

---

## CUD34SP

```
C
C      file cud34sp.d
C
C      * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *               University Corporation for Atmospheric
Research           *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK version 5.0.1
*
C      *
*
C      *               A Fortran Package of Multigrid
*
C      *
```

```
*  
C      *          Subroutines and Example Programs  
*  
C      *  
*  
C      *      for Solving Elliptic Partial Differential  
Equations      *  
C      *  
*  
C      *          by  
*  
C      *  
*  
C      *          John Adams  
*  
C      *  
*  
C      *          of  
*  
C      *  
*  
C      *          the National Center for Atmospheric  
Research      *  
C      *  
*  
C      *          Boulder, Colorado (80307)  
U.S.A.      *  
C      *  
*  
C      *          which is sponsored by  
*  
C      *  
*  
C      *          the National Science Foundation  
*  
C      *  
*  
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C  
C ... file cud34sp.d  
C  
C      contains documentation for subroutine  
cud34sp(work,phi,ierror)
```

```
c      A sample fortran driver is file "tcud34sp.f".
c
c ... required MUDPACK files
c
c      cud3sp.f, cudcom.f
c
c ... purpose
c
c      cud34sp attempts to improve the estimate in phi,
obtained by calling
c      cud3sp, from second to fourth order accuracy.
see the file "cud3sp.d"
c      for a detailed discussion of the elliptic pde
approximated and
c      arguments "work,phi" which are also part of the
argument list for
c      cud3sp.
c
c ... assumptions
c
c      * phi contains a second-order approximation from
an earlier cud3sp call
c
c      * arguments "work,phi" are the same used in
calling cud3sp
c
c      * "work,phi" have not changed since the last call
to cud3sp
c
c      * the finest grid level contains at least 6
points in each direction
c
c
c *** warning
c
c      if the first assumption is not true then a fourth
order approximation
c      cannot be produced in phi. the last assumption
(adequate grid size)
c      is the only one checked. failure in any of the
others can result in
c      in an undetectable error.
c
c ... language
```

```
c
c      fortran90/fortran77
c
c ... portability
c
c      mudpack5.01 software has been compiled and tested
with fortran77
c      and fortran90 on a variety of platforms.
c
c ... methods
c
c      details of the methods employed by the solvers in
mudpack are given
c      in [1,9]. [1,2,9] contain performance
measurements on a variety of
c      elliptic pdes (see "references" in the file
"readme"). in summary:
c
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
estimate (i.e.,
c      discretization level error is reached) then this
can be improved to a
c      fourth-order estimate using the technique of
"deferred corrections"
c      the values in the solution array are used to
generate a fourth-order
c      approximation to the truncation error. second-
order finite difference
c      formula are used to approximate third and fourth
partial derivatives
c      of the solution function [3]. the truncation
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side. the default
multigrid options
c      are used to compute the fourth-order correction
term which is added
c      to the original solution array.
c
```

```
C
C ... references (partial)
C
C
C      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
C      Solution of Linear Elliptic Partial Differential
Equations,"
C      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
C
C      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
C      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
C      1989, pp.1-12.
C      .
C      .
C      .
C      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
Haidvogel, and V. Pizzo,
C      "Applications of Multigrid Software in the
Atmospheric Sciences,"
C      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-
1458.
C      .
C      .
C      .
C      [9] J. Adams, "Recent Enhancements in MUDPACK, a
Multigrid Software
C      package for Elliptic Partial Differential
Equations," Applied Math.
C      and Comp., 1991, vol. 43, May 1991, pp. 79-94.
C
C      [10]J. Adams, "MUDPACK-2: Multigrid Software for
Approximating
C      Elliptic Partial Differential Equations on Uniform
Grids with
C      any Resolution," Applied Math. and Comp., 1993,
vol. 53, February
C      1993, pp. 235-249
C      .
C      .
C      .
```

```
c ... error argument
c
c      = 0 if no error is detected
c
c      = 30 if min0(nx,ny) < 6 where nx,ny are the fine
grid sizes
c          in the x,y directions.
c
c
c
*****
*****
```

C

```
*****
```

C

```
*****
```

C

```
*****
```

C

```
end of cud34sp documentation
```

C

```
*****
```

C

```
*****
```

C

```
*****
```

C

```
*****
```

CUD3CR

```
C
C      file cud3cr.d
C
C      * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * *
C      *
*
C      *               copyright (c) 2008 by UCAR
*
```

```
C      *
*
C      *      University Corporation for Atmospheric
Research          *
C      *
*
C      *      all rights reserved
*
C      *
*
C      *      MUDPACK version 5.0.1
*
C      *
*
C      *      A Fortran Package of Multigrid
*
C      *
*
C      *      Subroutines and Example Programs
*
C      *
*
C      *      for Solving Elliptic Partial Differential
Equations      *
C      *
*
C      *      by
*
C      *
*
C      *      John Adams
*
C      *
*
C      *      of
*
C      *
*
C      *      the National Center for Atmospheric
Research          *
C      *
*
C      *      Boulder, Colorado (80307)
U.S.A.          *
```

```
C      *
*
C      *                               which is sponsored by
*
C      *
*
C      *                               the National Science Foundation
*
C      *
*
C      * * * * * * * * * * * * * * * *
* * * * * * * *
C
C ... file cud3cr.d
C
C      contains documentation for:
C      subroutine
cud3cr(iparm,fparm,work,coef,bnd3cr,rhs,phi,mgopt,
C
+icros,crsxy,crsxz,crsyz,tol,maxit,iouter,rmax,ierror)
C      A sample fortran driver is file "tcud3cr.f".
C
C ... required MUDPACK files
C
C      cudcom.f
C
C
C ... purpose
C
C      subroutine cud3cr automatically discretizes and
attempts to compute
C      the second order finite difference approximation
to a complex 3-d
C      linear nonseparable elliptic partial differential
equation with
C      cross derivative terms on a box.  the
approximation is generated
C      on a uniform grid covering the box (see mesh
description below).
C      boundary conditions may be any combination of
oblique mixed
C      derivative (see bnd3cr description below),
specified (Dirchlet) or
C      periodic.  the form of the pde in operator
```

notation is

```

C
C           l(p) + lxyz(p) = r(x,y,z)
C
C       where
C
C           l(p) = cxx(x,y,z)*pxx + cyy(x,y,z)*pyy +
Czz(z,y,z)*pzz +
C
C                   cx(x,y,z)*px + cy(x,y,z)*py +
Cz(x,y,z)*pz +
C
C                   ce(x,y,z)*p(x,y,z) = r(x,y,z)
C
C       and
C
C           lxyz(p) = cxy(x,y,z)*pxy + cxz(x,y,z)*pxz +
Cyz(x,y,z)*pyz
C
C       here cxx,cyy,czz,cx,cy,cz,ce,cxy,cxz,cyz are the
known complex
C       coefficients of the pde; pxx,pyy,pzz,px,py,pz are
the second and
C       first partial derivatives of the unknown complex
solution function p
C       with respect to the independent variables x,y,z;
pxy,pxz, and pyz
C       are the second order mixed partial derivatives of
p with respect
C       to xy,xz, and yz. r(x,y,z) is the known complex
right hand side.
C
C
C ... mesh description . . .
C
C       the approximation is generated on a uniform nx by
ny by nz grid.
C       the grid is superimposed on the rectangular
solution region
C
C           [xa,xb] x [yc,yd] x [ze,zf].
C
C       let
C

```

```

c           dlx = (xb-xa) / (nx-1),  dly = (yd-yc) / (ny-1),
dlz = (zf-ze) / (nz-1)
c
c      be the uniform grid increments in the x,y,z
directions. then
c
c           xi=xa+(i-1)*dlx,   yj=yc+(j-1)*dly,  zk =
ze+(k-1)*dlz
c
c      for i=1,...,nx; j=1,...,ny; k=1,...,nz  denote the
x,y,z uniform
c      mesh points.
c
c
c ... methods
c
c
c      subroutine cud3cr is a recent addition to mudpack.
details
c      of the methods employed by the other solvers in
mudpack are in
c      [1,9,10].  [1,2,7,9,10] contain performance
measurements on a variety
c      of elliptic pdes (see "references" in the file
"readme").  the multi-
c      grid methods are described in documentation for
the other solvers.
c
c *** cud3cr differs fundamentally from the other
solvers in mudpack.
c      the full pde including cross derivative terms is
discretized on
c      the INTERIOR of the solution region:
c
c           xa < x < xb,  yc < y < yd,  ze < z < zf
c
c      however, on nonspecified (nondirichlet) boundaries
only l(p) is
c      discretized and the cross derivative term lxyz(p)
is moved to the
c      right hand side of the pde and approximated by
second order finite
c      finite difference formula applied to a previous

```

estimate in  $p(k-1)$ .  
 c     similarly, oblique mixed derivative boundary  
 conditions (see bnd3cr)  
 c     are converted to a "cud3" type mixed normal form  
 using second-order  
 c     finite difference formula applied to a previous  
 estimate  $p(k-1)$  to  
 c     approximate non-normal derivative components. for  
 example if  
 c       the mixed derivative condition  
 c  
 c        $py + a(x, z)*px + b(x, z)*pz + c(x, z)*p(x, yd, z) = gyd(y, z)$   
 c  
 c       is specified on the  $(x, z)$  plane of the upper  $y=yd$   
 boundary (see  
 c       bnd3cr for kbdy=4 below) then cud3cr converts this  
 to the mixed  
 c       normal derivative form  
 c  
 c        $py + c(x, z)*p(x, yd, z) = h(k, x, z)$   
 c  
 c       where the modified right hand side  $h(k, x, z)$  is  
 given by  
 c  
 c        $h(k, x, z) = gyd(x, z) - [a(x, z)*dx(p(k-1)) +$   
 $b(x, z)*dz(p(k-1))]$ .  
 c  
 c        $dx(p(k-1))$  and  $dz(p(k-1))$  are second order finite  
 difference  
 c       approximations to the nonnormal partial  
 derivatives  $px, pz$  using the  
 c       previous estimate in  $p(k-1)$ .  
 c  
 c       the result of full discretization on interior grid  
 points and partial  
 c       discretization with right hand side modifications  
 on boundaries,  
 c       is a linear system which we denote by  
 c  
 c        $D(p(k)) = r - Dxyz(p(k-1))$ .  
 c  
 c        $D$  is the coefficient matrix coming from the  
 discretization and

```

c      Dxyz(p(k-1)) stands for the right hand side
modification obtained
c      by approximating boundary cross derivative terms
and/or nonnormal
c      derivative components from mixed derivative
boundary conditions
c      with second order finite difference formula
applied to p(k-1).
c      with this notation, we formally describe the outer
iteration employed
c      by cud3cr:
c
c      algorithm cud3cr
c      .
c      set k = 0
c      .
c      set p(0) = 0.0 for all nonspecified grid points
c      .
c      repeat
c
c      .. k = k+1
c
c      .. solve D(p(k)) = r - Dxyz(p(k-1)) using
multigrid iteration
c
c      .. set rmax(k) = ||p(k) - p(k-1)|| / ||p(k) ||
c
c      until (rmax(k) < tol or k = maxit)
c      .
c      end cud3cr
c
c      tol is an error tolerance for convergence and
maxit is a limit on
c      the number of outer iterations. both are user
prescribed input
c      arguments to cud3cr. the maximum vector norm ||
|| is used in
c      computing the relative difference between
successive estimates in
c      rmax(k). large values for maxit should not be
used.
c
c
c ... language

```

```
C      fortran90/fortran77
C
C
C ... portability
C
C      mudpack5.0.1 software has been compiled and tested
with Fortran77
C      and Fortran90 on a variety of platforms.
C
C
C ... references (partial list)
C
C      for a complete list see "references" in the
mudpack information and
C      directory file "readme"
C
C      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
C      Solution of Linear Elliptic Partial Differential
Equations,"
C      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
C
C      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
C      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
C      1989, pp.1-12.
C      .
C      .
C      .
C      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
Haidvogel, and V. Pizzo,
C      "Applications of Multigrid Software in the
Atmospheric Sciences,"
C      Mon. Wea. Rev.,vol. 120 # 7, July 1992, pp. 1447-
1458.
C      .
C      .
C      .
C      [9] J. Adams, "Recent Enhancements in MUDPACK, a
Multigrid Software
C      package for Elliptic Partial Differential
```

Equations," Applied Math.  
C and Comp., 1991, vol. 43, May 1991, pp. 79-94.  
C  
C [10] J. Adams, "MUDPACK-2: Multigrid Software for  
Approximating  
C Elliptic Partial Differential Equations on Uniform  
Grids with  
C any Resolution," Applied Math. and Comp., 1993,  
vol. 53, February  
C 1993, pp. 235-249  
C .  
C .  
C .  
C  
C \*\*\*\*\*  
\*\*\*\*\*  
C \*\*\* arguments  
\*\*\*\*\*  
C \*\*\*\*\*  
\*\*\*\*\*  
\*\*\*\*\*  
C  
C arguments iparm, fparm, work, rhs, phi, coef, mgopt are  
the same as  
C those input to cud3 (see cud3.d for a detailed  
description) with the  
C following provisions:  
C  
C (1) the minimum required complex work space length  
for cud3cr is increased  
C by approximately  
C  
C  
nx\*ny\*nz\* (1+8\* (icros (1)+icros (2)+icros (3)) /7 +  
C  
C  
2\* (icros (1)+icros (2)+icros (3)) \* (nx\*ny+nx\*nz+ny\*nz)  
C  
C words over the minimum work space required by  
cud3 (see icros  
C description below). the exact minimal work  
space required  
C by cud3cr for the current set of input

```
arguments is output
c           in iparm(22). * The exact minimal work length
required
c           for the current method and grid size arguments
can be
c           predetermined by calling cud3cr with
iparm(21)=0 and
c           printout of iparm(22) or (in fortran 90 codes)
dynamically
c           allocating work using the the value in
iparm(22) in subsequent
c           calls to cud3cr.
c
c     (2) at least two calls to cud3cr are necessary to
generate an
c           approximation. intl=iparm(1)=0 is required on
the first
c           call. this call will do "once only"
discretization, and
c           set intermediate values in work which must be
preserved
c           for noninitial calls.
c
c     (3) maxcy = iparm(18) must be 1 or 2 (see ierror =
13).
c
c     (4) tolmax = fparm(5) = 0.0 is required. no
"internal" error control
c           is allowed within multigrid cycling (see
cud3.d)
c
c     (5) mgopt(1) = 0 is required. only the default
multigrid
c           options (W(2,1) cycles with cubic
prolongation) can be used
c           with cud3cr
c
c *** new arguments
c
c     the arguments:
bnd3cr,icros,crsxy,crsxz,crsyz,tol,maxit,iouter,rmax
c       are all new to cud3cr. the error argument,
ierror, has been expanded.
c       these are all described below:
```

```

c
c
c ... bnd3cr(kbdy,xory,yorz,a,b,c,g)
c
c      a subroutine with input arguments kbdy,xory,yorz
and output
c      arguments a,b,c,g.  bnd3cr inputs OBLIQUE mixed
derivative
c      conditions at any of the six x,y,z boundaries to
cud3cr.
c      a,b,c,g are all type complex.
c      described below:
c
c          (1) the kbdy=1 boundary
c
c      this is the (y,z) plane x=xa where
nxa=iparm(2)=2 flags
c      an oblique mixed boundary condition of the
form
c
c          px + axa(y,z)*py + bxa(y,z)*pz
+cxa(y,z)*p(xa,y,z) = gxa(y,z)
c
c      in this case kbdy=1,xory=y,yorz=z will be
input to bnd3cr and
c      a,b,c,g corresponding to the known
coefficients axa(y,z),bxa(y,z),
c      cxa(y,z),gxa(y,z) must be returned
c
c
c          (2) the kbdy=2 boundary
c
c      this is the (y,z) plane x=xb where
nxb=iparm(3)=2 flags
c      an oblique mixed boundary condition of the
form
c
c          px + axb(y,z)*py + bxb(y,z)*pz
+cxb(y,z)*p(xb,y,z) = gxb(y,z)
c
c      in this case kbdy=2,xory=y,yorz=z will be
input to bnd3cr and
c      a,b,c,g corresponding to the known
coefficients axb(y,z),bxb(y,z),

```

```

C           cxb(y,z),gxb(y,z) must be returned
C
C           (3) the kbdy=3 boundary
C
C           this is the (x,z) plane y=yc where
nyc=iparm(4)=2 flags
C           an oblique mixed boundary condition of the
form
C
C           py + ayc(x,z)*px + byc(x,z)*pz
+cyc(x,z)*p(x,yc,z) = gyc(x,z)
C
C           in this case kbdy=3,xory=x,yorz=z will be
input to bnd3cr and
C           a,b,c,g corresponding to the known
coefficients ayc(x,z),byc(x,z),
C           cyc(x,z),gyc(x,z) must be returned
C
C
C           (4) the kbdy=4 boundary
C
C           this is the (x,z) plane y=yd where
nyd=iparm(5)=2 flags
C           an oblique mixed boundary condition of the
form
C
C           py + ayd(x,z)*px + byd(x,z)*pz
+cyd(x,z)*p(x,yd,z) = gyd(x,z)
C
C           in this case kbdy=4,xory=x,yorz=z will be
input to bnd3cr and
C           a,b,c,g corresponding to the known
coefficients ayd(x,z),byd(x,z),
C           cyd(x,z),gyd(x,z) must be returned
C
C           (5) the kbdy=5 boundary
C
C           this is the (x,y) plane z=ze where
nze=iparm(6)=2 flags
C           an oblique mixed boundary condition of the
form
C
C           pz + aze(x,y)*px + bze(x,y)*py +
cze(x,y)*p(x,y,ze) = gze(x,y)

```

```

c
c           in this case kbdy=5,xory=x,yorz=y will be
input to bnd3cr and
c           a,b,c,g corresponding to the known
coefficients aze(x,y),bze(x,y),
c           cze(x,y),gze(x,y) must be returned
c
c           (6) the kbdy=6 boundary
c
c           this is the (x,y) plane z=zf where
nzf=iparm(7)=2 flags
c           an oblique mixed boundary condition of the
form
c
c           pz + azf(x,y)*px + bzf(x,y)*py +
czf(x,y)*p(x,y,zf) = gzf(x,y)
c
c           in this case kbdy=6,xory=x,yorz=y will be
input to bnd3cr and
c           a,b,c,g corresponding to the known
coefficients azf(x,y),bzf(x,y),
c           czf(x,y),gzf(x,y) must be returned
c
c
c           bnd3cr must be delcared "external" in the routine
calling cud3cr
c           where its name may be different. bnd3cr must be
entered as a
c           dummy subroutine even if there are no derivative
boundary conditions.
c           for an example of how to set up a subroutine to
input derivative
c           boundary conditions, see the test program
tcud3cr.f
c
c ... icros
c
c           an integer vector argument dimensioned 3 which
flags the presence
c           or absence of cross derivative terms in the pde as
follows:
c
c           icros(1) = 1 if cxy(x,y,z) is nonzero for any
grid point (x,y,z)

```

```

c      icros(1) = 0 if cxy(x,y,z) = (0.0,0.0) for all
grid points (x,y,z)
c
c      icros(2) = 1 if cxz(x,y,z) is nonzero for any
grid point (x,y,z)
c      icros(2) = 0 if cxz(x,y,z) = (0.0,0.0) for all
grid points (x,y,z)
c
c      icros(3) = 1 if cyz(x,y,z) is nonzero for any
grid point (x,y,z)
c      icros(3) = 0 if cyz(x,y,z) = (0.0,0.0) for all
grid points (x,y,z)
c
c
c ... crsxy(x,y,z,cxy)
c
c      if icros(1) = 1 then crsxy is a subroutine with
arguments
c      (x,y,z,cxy) which supplies the xy cross derivative
coefficient
c      cxy at the grid point (x,y,z).  if icros(1) = 0
then crsxy
c      is a dummy subroutine argument (i.e., it must be
provided but
c      will not be invoked).
c
c
c ... crsxz(x,y,z,cxz)
c
c      if icros(2) = 1 then crsxz is a subroutine with
arguments
c      (x,y,z,cxz) which supplies the xz cross derivative
coefficient
c      cxz at the grid point (x,y,z).  if icros(2) = 0
then crsxz
c      is a dummy subroutine argument (i.e., it must be
provided but
c      will not be invoked).
c
c
c ... crsyz(x,y,z,cyz)
c
c      if icros(3) = 1 then crsyz is a subroutine with
arguments

```

```
c      (x,y,z,cyz) which supplies the yz cross derivative
c      coefficient
c      cxy at the grid point (x,y,z). if icros(3) = 0
then crsyz
c      is a dummy subroutine argument (i.e., it must be
provided but
c      will not be invoked).
c
c      crsxy,crsxz,crsyz must be declared "external" in
the routine
c      calling cud3cr. the names chosen for these
routines can be
c      different (see tcud3cr.f for an example)
c
c ... tol
c
c      tol is an error control argument for the outer
iteration employed
c      by cud3cr (see "methods" description above). if
tol > 0.0 is input
c      then tol is a relative error tolerance for
convergence. the outer
c      iteration terminates and convergence is deemed to
have occurred at the
c      k(th) iterate if the maximum relative difference,
rmax(k), satisfies
c
c            def
c            rmax(k) = ||p(k) - p(k-1)|| / ||p(k)|| < tol.
c
c      the last approximation p(maxit) is returned in phi
even if
c      convergence does not occur. the maximum norm ||
|| is used.
c      when tol = 0.0 is input, error control is not
implemented and
c      exactly maxit (see below) outer iterations are
executed in cud3cr.
c      the tol = 0.0 option eliminates unnecessary
computation when
c      the user is certain of the required value for
maxit.
c
c
```

```

c ... maxit
c
c      a limit on the outer iteration loop (see "method"
description)
c      used to approximate the 3-d pde with cross
derivative terms when
c      tol > 0.0. if tol = 0.0 is entered then exactly
maxit outer
c      iterations are performed and only rmax(maxit) is
computed. the
c      total number of relaxation sweeps performed at the
finest grid
c      level is bounded by 3*maxcy*maxit. large values
for maxit should
c      not be used.

c
c
c ****
*****output
arguments*****
***

c
*****space length
*****iparm(22)
c
c      on output iparm(22) contains the actual work
space length
c      required by cud3cr for the current grid sizes
and method.
c      this will be approximately

c
nx*ny*nz* (1+8* (icros(1)+icros(2)+icros(3)) /7 +
c
c
2* (icros(1)+icros(2)+icros(3)) * (nx*ny+nx*nz+ny*nz)
c
c      words longer than the space required by cud3
(see cud3.d)

```

```

C
C
C ... work
C
C           on output work contains intermediate values
that must not be
C           destroyed if cud3cr is to be called again
with iparm(1)=1
C           and iparm(17)=1.
C
C
C ... phi
C
C           on output phi(i,j,k) contains the
approximation to
C           p(xi,yj,zk) for all mesh points i=1,...,nx;
j=1,...,ny;
C           k=1,...,nz. the last computed iterate in phi
is returned
C           even if convergence is not obtained.
C
C
C ... iouter
C
C           the number of outer iterations (see "method"
description above)
C           executed by cud3cr for the current call.
maxit is an upper bound
C           for iouter
C
C
C ... rmax (see tol,maxit descriptions above)
C
C           a maxit dimensioned real vector. if tol >
0.0 is input then
C           rmax(k) for k=1,...,iouter contain the
maximum relative
C           difference between successive estimates.
rmax(k) is
C           given by
C
C           rmax(k) = ||p(k) - p(k-1)|| / ||p(k) ||
C
C           for k=1,...,iouter. the maximum norm || ||

```

```
is used. either
c           iouter < maxit (convergence) or iouter =
maxit is possible.
c           if tol = 0.0 input then exactly maxit outer
iterations are
c           executed and only rmax(maxit) is computed.
in this case
c           rmax(1),...,rmax(maxit-1) are set to 0.0.
the tol = 0.0
c           option eliminates unnecessary computation
when the user is
c           certain of the required value for maxit.
c
c
c ... ierror
c
c           an integer error argument which indicates
fatal errors when
c           returned positive. the negative values -5,-
4,-3,-2,-1 and
c           ierror = 2,3,4,5,6,9,10 have the same meaning
as described for
c           for cud3 (see cud3.d). in addition:
c
c ***      new nonfatal error
c
c           ierror = -10 if tol > 0.0 is input (error
control) and convergence
c           fails in maxit outer iterations. in this
case the latest
c           approximation p(maxit) is returned in phi
(cud3cr can be recalled
c           with iparm(1)=iparm(17)=1 to improve the
approximation as long
c           as all other arguments are unchanged)
c
c ***      new fatal errors
c
c ... ierror
c
c     = 1 if intl=iparm(1) is not 0 on initial call or
not 0 or 1
c           on subsequent calls of if intl=0 and
iguess=iparm(17)=1
```

```

C      = 7 if maxcy = iparm(18) is not 1 or 2
C
C      = 8 if method = iparm(19) is less than 0 or
greater than 7
C          cud3cr does not allow planar relaxation.
meth2=iparm(20)
C          is not used or checked.
C
C      =11 if tolmax = fparm(7) is not 0.0
C
C      =12 if kcycle = mgopt(1) is not 0
C
C      =13 if icros(1) or icros(2) or icros(3) is not 0
or 1
C
C      =14 if tol < 0.0
C
C      =15 if maxit < 1
C
C
*****
*****
C
*****
*****
C
      end of cud3cr documentation
C
C
*****
*****
C
*****
*****
C

```

```
C  
C      file cud3sp.d  
C  
C      * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C      *  
*  
C      *          copyright (c) 2008 by UCAR  
*  
C      *  
*  
C      *          University Corporation for Atmospheric  
Research      *  
C      *  
*  
C      *          all rights reserved  
*  
C      *  
*  
C      *          MUDPACK version 5.0.1  
*  
C      *  
*  
C      *          A Fortran Package of Multigrid  
*  
C      *  
*  
C      *          Subroutines and Example Programs  
*  
C      *  
*  
C      *          for Solving Elliptic Partial Differential  
Equations      *  
C      *  
*  
C      *          by  
*  
C      *  
*  
C      *          John Adams  
*  
C      *  
*
```

```
C      *                                of
*
C      *
*
C      *          the National Center for Atmospheric
Research           *
C      *
*
C      *          Boulder, Colorado (80307)
U.S.A.           *
C      *
*
C      *          which is sponsored by
*
C      *
*
C      *          the National Science Foundation
*
C      *
*
C      *          ****
* * * * * * *
C
C ... file cud3sp.d
C
C     contains documentation for:
C     subroutine
cud3sp(iparm,fparm,work,cofx,cofy,cofz,bndyc,rhs,phi,
C     +             mgopt,ierror)
C     A sample fortran driver is file "tcud3sp.f".
C
C ... required MUDPACK files
C
C     cudcom.f
C
C ... purpose
C
C     subroutine cud3sp automatically discretizes and
attempts to compute
C     the second order finite difference approximation
to a complex three-
C     dimensional linear SEPARABLE elliptic partial
partial differential
C     equation on a box.  the approximation is generated
```

```

on a uniform
c      grid covering the box (see mesh description
below). boundary
c      conditions may be any combination of mixed,
specified (Dirchlet)
c      or periodic. the form of the pde solved is . . .
c
c      cxx(x)*pxx + cx(x)*px + cex(x)*p(x,y,z) +
c
c      cyy(y)*pyy + cy(y)*py + cey(y)*p(x,y,z) +
c
c      czz(z)*pzz + cz(z)*pz + cez(z)*p(x,y,z) =
r(x,y,z)
c
c      here cxx,cx,cex,cyy,cy,cey,czz,cz,cez are the
known complex coefficients
c      of the pde; pxx,pyy,pzz,px,py,pz are the second
and first
c      partial derivatives of the unknown solution
function p(x,y,z)
c      with respect to the independent variables x,y,z;
r(x,y,z) is
c      is the known complex right hand side of the
elliptic pde.

c
c      SEPARABILITY means:
c
c      cxx,cx,cex depend only on x
c      cyy,cy,cey depend only on y
c      czz,cz,cez depend only on z
c
c      For example, LaPlace's equation in Cartesian
coordinates is separable.
c      Nonseparable elliptic PDEs can be approximated
with cud3.
c      cud3sp requires considerably less work space than
cud3.
c
c ... mesh description . . .
c
c      the approximation is generated on a uniform nx by
ny by nz grid.
c      the grid is superimposed on the rectangular
solution region

```

```

C
C           [xa,xb] x [yc,yd] x [ze,zf].
C
C       let
C
C           dlx = (xb-xa) / (nx-1), dly = (yd-yc) / (ny-1),
dlz = (zf-ze) / (nz-1)
C
C       be the uniform grid increments in the x,y,z
directions. then
C
C           xi=xa+(i-1)*dlx,   yj=yc+(j-1)*dly,  zk =
ze+(k-1)*dlz
C
C       for i=1,...,nx; j=1,...,ny; k=1,...,nz  denote the
x,y,z uniform
C       mesh points.
C
C
C ... language
C
C       fortran90/fortran77
C
C
C ... portability
C
C       mudpack5.0.1 software has been compiled and tested
with Fortran77
C       and Fortran90 on a variety of platforms.
C
C ... methods
C
C       details of the methods employed by the solvers in
mudpack are given
C       in [1,9]. [1,2,9] contain performance
measurements on a variety of
C       elliptic pdes (see "references" in the file
"readme"). in summary:
C
C *** discretization and solution (second-order solvers)
(see [1])
C
C       the pde and boundary conditions are automatically
discretized at all

```

```
c      grid levels using second-order finite difference
formula. diagonal
c      dominance at coarser grid levels is maintained in
the presence of
c      nonzero first-order terms by adjusting the second-
order coefficient
c      when necessary. the resulting block tri-diagonal
linear system is
c      approximated using multigrid iteration
[10,11,13,15,16,18]. version
c      5.0.1 of mudpack uses only fully weighted residual
restriction. defaults
c      include cubic prolongation and w(2,1) cycles.
these can be overridden
c      with selected multigrid options (see "mgopt"). error control based on
c      maximum relative differences is available. full
multigrid cycling (fmg)
c      or cycling beginning or restarting at the finest
grid level can be
c      selected. a menu of relaxation methods including
gauss-seidel point,
c      line relaxation(s) (in any combination of
directions) and planar
c      relaxation (for three-dimensional anisotropic
problems) are provided.
c      all methods use ordering based on alternating
points (red/black),
c      lines, or planes for cray vectorization and
improved convergence
c      rates [14].
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
estimate (i.e.,
c      discretization level error is reached) then this
can be improved to a
c      fourth-order estimate using the technique of
"deferred corrections."
c      the values in the solution array are used to
generate a fourth-order
c      approximation to the truncation error. second-
```

```
order finite difference
c      formula are used to approximate third and fourth
partial derivaties
c      of the solution function [3]. the truncation
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side. the default
multigrid options
c      are used to compute the fourth-order correction
term which is added
c      to the original solution array.

c
c
c ... references (partial)
c
c
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
c      Solution of Linear Elliptic Partial Differential
Equations,"
c      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
c
c      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
c      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
c      1989, pp.1-12.
c      .
c      .
c      .
c      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
Haidvogel, and V. Pizzo,
c      "Applications of Multigrid Software in the
Atmospheric Sciences,"
c      Mon. Wea. Rev.,vol. 120 # 7, July 1992, pp. 1447-
1458.
c      .
c      .
c      .
c      [9] J. Adams, "Recent Enhancements in MUDPACK, a
Multigrid Software
c      package for Elliptic Partial Differential
```

```
Equations," Applied Math.  
c      and Comp., 1991, vol. 43, May 1991, pp. 79-94.  
c  
c      [10] J. Adams, "MUDPACK-2: Multigrid Software for  
Approximating  
c      Elliptic Partial Differential Equations on Uniform  
Grids with  
c      any Resolution," Applied Math. and Comp., 1993,  
vol. 53, February  
c      1993, pp. 235-249  
c      .  
c      .  
c      .  
c  
c ... argument description  
c  
c  
c  
*****  
*****  
c *** input arguments  
*****  
c  
*****  
*****  
c  
c  
c ... iparm  
c  
c      an integer vector of length 22 used to  
efficiently pass  
c      integer arguments. iparm is set internally  
in cud3sp  
c      and defined as follows . . .  
c  
c  
c ... intl=iparm(1)  
c  
c      an initialization argument. intl=0 must be  
input  
c      on an initial call. in this case input  
arguments will  
c      be checked for errors and the elliptic  
partial differential
```

```
c           equation and boundary conditions will be
discretized using
c           second order finite difference formula.
c
c ***      An approximation is NOT generated after an
intl=0 call!
c           cud3sp should be called with intl=1 to
approximate the elliptic
c           PDE discretized by the intl=0 call.  intl=1
should also
c           be input if cud3sp has been called earlier
and only the
c           values in in rhs (see below) or gbody (see
bndyc below)
c           or phi (see below) have changed.  This will
bypass
c           redundant pde discretization and argument
checking
c           and save computational time.  Some examples
of when
c           intl=1 calls should be used are:
c
c           (0) after a intl=0 argument checking and
discretization call
c
c           (1) cud3sp is being recalled for additional
accuracy.  In
c           this case iguess=iparm(12)=1 should also
be used.
c
c           (2) cud3sp is being called every time step in
a time dependent
c           problem (see discussion below) where the
elliptic operator
c           does not depend on time.
c
c           (3) cud3sp is being used to solve the same
elliptic equation
c           for several different right hand sides
(igueess=0 should
c           probably be used for each new righthand
side).
c
c           intl = 0 must be input before calling with
```

```
intl = 1 when any
c          of the following conditions hold:
c
c          (a) the initial call to cud3sp
c
c          (b) any of the integer arguments other than
iguess=iparm(12)
c                  or maxcy=iparm(13) or mgopt have changed
since the previous
c                  call.
c
c          (c) any of the floating point arguments other
than tolmax=
c                  fparm(5) have changed since the previous
call
c
c          (d) any of the coefficients input by coef
(see below) have
c                  changed since the previous call
c
c          (e) any of the "alfa" coefficients input by
bndyc (see below)
c                  have changed since the previous call.
c
c          If any of (a) through (e) are true then the
elliptic PDE
c          must be discretized or rediscretized.  If
none of (a)
c          through (e) holds, calls can be made with
intl=1.
c          Incorrect calls with intl=1 will produce
erroneous results.
c  ***      The values set in the saved work space "work"
(see below) with
c          an intl=0 call must be preserved with
subsequent intl=1 calls.
c
c          MUDPACK software performance should be
monitored for intl=1
c          calls.  The intl=0 discretization call
performance depends
c          primarily on the efficiency or lack of
efficiency of the
c          user provided subroutines for pde
```

```
coefficients and
C           boundary conditions.
C
C
C ... nxa=iparm(2)
C
C           flags boundary conditions on the (y,z) plane
x=xa
C
C           = 0 if p(x,y,z) is periodic in x on [xa,xb]
C           (i.e., p(x+xb-xa,y,z) = p(x,y,z) for all
x,y,z)
C
C           = 1 if p(xa,y,z) is specified (this must be
input thru phi(1,j,k))
C
C           = 2 if there are mixed derivative boundary
conditions at x=xa
C           (see "bndyc" description below where kbdy =
1)
C
C
C ... nxb=iparm(3)
C
C           flags boundary conditions on the (y,z) plane
x=xb
C
C           = 0 if p(x,y,z) is periodic in x on [xa,xb]
C           (i.e., p(x+xb-xa,y,z) = p(x,y,z) for all
x,y,z)
C
C           = 1 if p(xb,y,z) is specified (this must be
input thru phi(nx,j,k))
C
C           = 2 if there are mixed derivative boundary
conditions at x=xb
C           (see "bndyc" description below where kbdy =
2)
C
C
C ... nyc=iparm(4)
C
C           flags boundary conditions on the (x,z) plane
y=yc
```

```

C
C           = 0 if p(x,y,z) is periodic in y on [yc,yd]
C           (i.e., p(x,y+yd-yc,z) = p(x,y,z) for all
x,y,z)

C           = 1 if p(x,yc,z) is specified (this must be
input thru phi(i,1,k))

C           = 2 if there are mixed derivative boundary
conditions at y=yc
C           (see "bndyc" description below where kbdy =
3)

C
C
C ... nyd=iparm(5)

C
C           flags boundary conditions on the (x,z) plane
y=yd

C
C           = 0 if p(x,y,z) is periodic in y on [yc,yd]
C           (i.e., p(x,y+yd-yc,z) = p(x,y,z) for all
x,y,z)

C           = 1 if p(x,yd,z) is specified (this must be
input thru phi(i,ny,k))

C           = 2 if there are mixed derivative boundary
conditions at y=yd
C           (see "bndyc" description below where kbdy =
4)

C
C
C ... nze=iparm(6)

C
C           flags boundary conditions on the (x,y) plane
z=ze

C
C           = 0 if p(x,y,z) is periodic in z on [ze,zf]
C           (i.e., p(x,y,z+zf-ze) = p(x,y,z) for all
x,y,z

C           = 1 if p(x,y,ze) is specified (this must be
input thru phi(i,j,1))

```

```

c      = 2 if there are mixed derivative boundary
conditions at z=ze
c      (see "bndyc" description below where kbdy =
5)
c
c
c ... nzf=iparm(7)
c
c      flags boundary conditions on the (x,y) plane
z=zf
c
c      = 0 if p(x,y,z) is periodic in z on [ze,zf]
c      (i.e., p(x,y,z+zf-ze) = p(x,y,z) for all
x,y,z

c      = 1 if p(x,y,zf) is specified (this must be
input thru phi(i,j,nz))

c      = 2 if there are mixed derivative boundary
conditions at z=zf
c      (see "bndyc" description below where kbdy =
6)
c
c
c *** grid size arguments
c
c
c ... ixp = iparm(8)
c
c      an integer greater than one which is used in
defining the number
c      of grid points in the x direction (see nx =
iparm(14)). "ixp+1"
c      is the number of points on the coarsest x
grid visited during
c      multigrid cycling. ixp should be chosen as
small as possible.
c      recommended values are the small primes 2 or
3 or (possibly) 5.
c      larger values can reduce multigrid
convergence rates considerably,
c      especially if line relaxation in the x
direction is not used.
c      if ixp > 2 then it should be 2 or a small odd

```

```
value since a power
c          of 2 factor of ixp can be removed by
increasing iex = iparm(11)
c          without changing nx = iparm(14)
c
c
c ... jyq = iparm(9)
c
c          an integer greater than one which is used in
defining the number
c          of grid points in the y direction (see ny =
iparm(15)). "jyq+1"
c          is the number of points on the coarsest y
grid visited during
c          multigrid cycling. jyq should be chosen as
small as possible.
c          recommended values are the small primes 2 or
3 or (possibly) 5.
c          larger values can reduce multigrid
convergence rates considerably,
c          especially if line relaxation in the y
direction is not used.
c          if jyq > 2 then it should be 2 or a small odd
value since a power
c          of 2 factor of jyq can be removed by
increasing jey = iparm(12)
c          without changing ny = iparm(15)
c
c
c ... kzs = iparm(10)
c
c          an integer greater than one which is used in
defining the number
c          of grid points in the z direction (see nz =
iparm(16)). "kzs+1"
c          is the number of points on the coarsest z
grid visited during
c          multigrid cycling. kzs should be chosen as
small as possible.
c          recommended values are the small primes 2 or
3 or (possibly) 5.
c          larger values can reduce multigrid
convergence rates considerably,
c          especially if line relaxation in the z
```

```
direction is not used.  
c           if kzz > 2 then it should be 2 or a small odd  
value since a power  
c           of 2 factor of kzz can be removed by  
increasing kez = iparm(13)  
c           without changing nz = iparm(16)  
c  
c  
c ... iex = iparm(11)  
c  
c           a positive integer exponent of 2 used in  
defining the number  
c           of grid points in the x direction (see nx =  
iparm(14)).  
c           iex .le. 50 is required. for efficient  
multigrid cycling,  
c           iex should be chosen as large as possible and  
ixp=iparm(8)  
c           as small as possible within grid size  
constraints when  
c           defining nx = iparm(14).  
c  
c  
c ... jey = iparm(12)  
c  
c           a positive integer exponent of 2 used in  
defining the number  
c           of grid points in the y direction (see ny =  
iparm(15)).  
c           jey .le. 50 is required. for efficient  
multigrid cycling,  
c           jey should be chosen as large as possible and  
jyq=iparm(9)  
c           as small as possible within grid size  
constraints when  
c           defining ny = iparm(15).  
c  
c  
c ... kez = iparm(13)  
c  
c           a positive integer exponent of 2 used in  
defining the number  
c           of grid points in the z direction (see nz =  
iparm(16)).
```

```

c           kez .le. 50 is required.  for efficient
multigrid cycling,
c           kez should be chosen as large as possible and
kzr=iparm(10)
c           as small as possible within grid size
constraints when
c           defining nz = iparm(16).
c
c
c ... nx = iparm(14)
c
c           the number of equally spaced grid points in
the interval [xa,xb]
c           (including the boundaries).  nx must have the
form
c
c           nx = ixp* (2** (iex-1)) + 1
c
c           where ixp = iparm(8), iex = iparm(11).
c
c
c ... ny = iparm(15)
c
c           the number of equally spaced grid points in
the interval [yc,yd]
c           (including the boundaries).  ny must have the
form:
c
c           ny = jyq* (2** (jey-1)) + 1
c
c           where jyq = iparm(9), jey = iparm(12).
c
c
c ... nz = iparm(16)
c
c           the number of equally spaced grid points in
the interval [ze,zf]
c           (including the boundaries).  nz must have the
form
c
c           nz = kzr* (2** (kez-1)) + 1
c
c           where kzr = iparm(10), kez = iparm(13)

```

```

C
C *** example
C
C      suppose a solution is wanted on a 33 by 65 by
97 grid. then
C      ixp=2, jyq=4, kxr=6 and iex=jey=kez=5 could be
used. a better
C      choice would be ixp=jyq=2, kxr=3, and iex=5,
jey=kez=6.
C
C *** note
C
C      let G be the nx by ny by nz fine grid on which the
approximation is
C      generated and let n = max0(iex,jey,kez). in
mudpack, multigrid
C      cycling is implemented on the ascending chain of
grids
C
C      G(1) < ... < G(k) < ... < G(n) = G.
C
C      each g(k) (k=1,...,n) has mx(k) by my(k) by mz(k)
grid points
C      given by:
C
C      mx(k) = ixp*[2**max0(iex+k-n,1)-1] + 1
C
C      my(k) = jyq*[2**max0(jey+k-n,1)-1] + 1
C
C      mz(k) = kxr*[2**max0(kez+k-n,1)-1] + 1
C
C
C
C ... iguess=iparm(17)
C
C      = 0 if no initial guess to the pde is
provided
C      and/or full multigrid cycling beginning
at the
C      coarsest grid level is desired.
C
C      = 1 if an initial guess to the pde at the
finest grid
C      level is provided in phi (see below). in

```

```
this case
c           cycling beginning or restarting at the
finest grid
c           is initiated.
c
c *** comments on iguess = 0 or 1 . . .
c
c
c     setting iguess=0 forces full multigrid or "fmg"
c     cycling. phi
c     must be initialized at all grid points. it can be
set to zero at
c     non-Dirchlet grid points if nothing better is
available. the
c     values set in phi when iguess = 0 are passed and
down and serve
c     as an initial guess to the pde at the coarsest
grid level where
c     multigrid cycling commences.
c
c     if iguess = 1 then the values input in phi are an
initial guess to the
c     pde at the finest grid level where cycling begins.
this option should
c     be used only if a "very good" initial guess is
available (as, for
c     example, when restarting from a previous iguess=0
call).
c
c *** time dependent problems . . .
c
c     assume we are solving an elliptic pde every time
step in a
c     marching problem of the form:
c
c           l(p(t)) = r(t)
c
c     where the differential operator "l" has no time
dependence,
c     "p(t)" is the solution and "r(t)" is the right
hand side at
c     current time "t". let "dt" be the increment
between time steps.
c     then p(t) can be used as an initial guess to
```

```

p(t+dt) with
c      intl = 1 when solving
c
c      l(p(t+dt)) = r(t+dt).
c
c      after the first two time steps, rather than
continue, it would
c      be better to define the "correction" term:
c
c      e(t,dt) = p(t+dt) - p(t)
c
c      this clearly satisfies the equation
c
c      l(e(t,dt)) = r(t+dt) - r(t).
c
c      this should be solved with iguess = 0 and intl =
1. boundary
c      conditions for e(t,dt) are obtained from the
boundary conditions
c      for p(t) by subtracting given values at t from
given values at
c      t+dt. for example if
c
c      d(p(t))/dx = f(t), d(p(t+dt))/dx = f(t+dt)
c
c      at some x boundary then e(t,dt) satisfies the
derivative
c      boundary condition
c
c      d(e(t,dt))/dx = f(t+dt) - f(t).
c
c      e(t,dt) can be preset to 0.0 (at non-Dirchlet
points) or (if p(t-dt)
c      is saved) to p(t)-p(t-dt). with iguess = 0, these
values will serve
c      as an initial guess to e(t,dt) at the coarsest
grid level. this
c      approach has the advantage that a full sequence of
multigrid cycles,
c      beginning at the coarsest grid level, is invoked
every time step in
c      solving for e(t,dt). a few digits of accuracy in
e(t,dt), which is
c      ordinarily much smaller than p(t), will yield

```

```

several more digits of
c      accuracy in the final approximation:
c
c      p(t+dt) = p(t) + e(t,dt).
c
c      using this approach to integrate in time will give
more accuracy
c      then using p(t) as an initial guess to p(t+dt) for
all time steps.
c      it does require additional storage.
c
c      if the differential operator "l" has time
dependence (either thru
c      the coefficients in the pde or the coefficients in
the derivative
c      boundary conditions) then use p(t) as an initial
guess to p(t+dt)
c      when solving
c
c      l(t+dt) (p(t+dt)) = r(t+dt)
c
c      with intl = 0 for all time steps (the
discretization must be repeated
c      for each new "t"). either iguess = 0 (p(t) will
then be an initial
c      guess at the coarsest grid level where cycles will
commence) or
c      iguess = 1 (p(t) will then be an initial guess at
the finest grid
c      level where cycles will remain fixed) can be
tried.
c
c
c ... maxcy = iparm(18)
c
c      the exact number of cycles executed between
the finest
c      (nx by ny by nz) and the coarsest ((ixp+1) by
(jyq+1) by
c      (kzr+1)) grid levels when tolmax=fparm(7)=0.0
(no error
c      control). when tolmax=fparm(7).gt.0.0 is
input (error control)
c      then maxcy is a limit on the number of cycles

```

```
between the
c           finest and coarsest grid levels.  in any
case, at most
c           maxcy*(iprert+ipost) relaxation sweeps are
performed at the
c           finest grid level (see
iprert=mgopt(2),ipost=mgopt(3) below)
c           when multigrid iteration is working
"correctly" only a few
c           cycles are required for convergence.  large
values for maxcy
c           should not be required.
c
c
c ... method = iparm(19)
c
c
c           = 0 for gauss-seidel pointwise relaxation
with red/black ordering
c
c           This is the only relaxation method offered
with cud3sp.  Line
c           or planar relaxation would "lose" the
significant savings in
c           work space length defeating the purpose of
cud3sp.  If line
c           or planar relaxation is required then use
cud3.  method is
c           used as an argument only to focus attention
on the purpose
c           of cud3sp.
c
c ... length = iparm(20)
c
c           the length of the work space provided in
vector work.
c           This is considerably less then the work space
required by
c           the nonseparable solver cud3.
c
c           length = 7*(nx+2)*(ny+2)*(nz+2)/2
c
c           will usually but not always suffice.  The
exact minimal length
```

```
c           depends on the grid size arguments. It can
be predetermined
c ***      for the current input arguments by calling
cud3sp with iparm(20)
c           set equal to zero and printing iparm(21) or
(in f90) dynamically
c           allocating the work space using the value in
iparm(21) in a
c           subsequent cud3sp call.
c
c ... fparm
c
c           a floating point vector of length 8 used to
efficiently
c           pass floating point arguments. fparm is set
internally
c           in cud3sp and defined as follows . . .
c
c
c ... xa=fparm(1), xb=fparm(2)
c
c           the range of the x independent variable. xa
must
c           be less than xb
c
c
c ... yc=fparm(3), yd=fparm(4)
c
c           the range of the y independent variable. yc
must
c           be less than yd.
c
c
c ... ze=fparm(5), zf=fparm(6)
c
c           the range of the z independent variable. ze
must
c           be less than zf.
c
c
c ... tolmax = fparm(7)
c
c           when input positive, tolmax is a maximum
relative error tolerance
```

```

c           used to terminate the relaxation iterations.
assume phil(i,j,k)
c           and phi2(i,j,k) are the last two computed
approximations at all
c           grid points of the finest grid level. if we
define
c
c           phdif = max(abs(phi2(i,j,k)-phil(i,j,k)))
for all i,j,k
c
c           and
c
c           phmax = max(abs(phi2(i,j,k))) for all
i,j,k
c
c           then "convergence" is considered to have
occurred if and only if
c
c           phdif/phmax < tolmax.
c
c
c           if tolmax=fparm(7)=0.0 is input then there is
no error control
c           and maxcy cycles from the finest grid level
are executed. maxcy
c           is a limit which cannot be exceeded even with
error control.
c           *** calls with tolmax=0.0, when appropriate
because of known
c           convergence behavior, are more efficient than
calls with tolmax
c           positive (i.e., if possible DO NOT use error
control!).
c
c
c ... work
c
c           a complex array that must be provided for
work space.
c           see length = iparm(20). the values in work
must be preserved
c           if cud3sp is called again with
intl=iparm(1).ne.0 or if cud34sp
c           is called to improve accuracy.

```

```

C
C
C ... bndyc
C
C           a subroutine with arguments
(kbdy,xory,yorz,alfa,gbdy) .
C           which are used to input complex mixed
boundary conditions.
C           the boundaries are numbered one thru six and
the form of
C           conditions are described below.
C
C
C           (1) the kbdy=1 boundary
C
C           this is the (y,z) plane x=xa where nxa=iparm(2) =
2 flags
C           a mixed boundary condition of the form
C
C           dp/dx + alfxa*p(xa,y,z) = gbdxa(y,z)
C
C           in this case kbdy=1,xory=y,yorz=z will be input to
bndyc and
C           alfa,gbdy corresponding to alfxa,gbdxa(y,z) must
be returned.
C
C
C           (2) the kbdy=2 boundary
C
C           this is the (y,z) plane x=xb where nxb=iparm(3) =
2 flags
C           a mixed boundary condition of the form
C
C           dp/dx + alfxb*p(xb,y,z) = gbdxb(y,z)
C
C           in this case kbdy=2,xory=y,yorz=z will be input to
bndyc and
C           alfa,gbdy corresponding to alfxb,gbdxb(y,z) must
be returned.
C
C
C           (3) the kbdy=3 boundary
C
C           this is the (x,z) plane y=yc where nyc=iparm(4) =

```

```

2 flags
c      a mixed boundary condition of the form
c
c      dp/dy + alfy*c*p(x,yc,z) = gbdyc(x,z)
c
c      in this case kbdy=3,xory=x,yorz=z will be input to
bndyc and
c      alfa,gbdy corresponding to alfy*c,p(x,z) must
be returned.
c
c
c      (4) the kbdy=4 boundary
c
c      this is the (x,z) plane y=yd where nyd=iparm(5) =
2 flags
c      a mixed boundary condition of the form
c
c      dp/dy + alfyd*p(x,yd,z) = gbdyd(x,z)
c
c      in this case kbdy=4,xory=x,yorz=z will be input to
bndyc and
c      alfa,gbdy corresponding to alfyd,p(x,z) must
be returned.
c
c
c      (5) the kbdy=5 boundary
c
c      this is the (x,y) plane z=ze where nze=iparm(6) =
2 flags
c      a mixed boundary condition of the form
c
c      dp/dz + alfze*p(x,y,ze) = gbdze(x,y)
c
c      in this case kbdy=5,xory=x,yorz=y will be input to
bndyc and
c      alfa,gbdy corresponding to alfze,p(x,y) must
be returned.
c
c
c      (6) the kbdy=6 boundary
c
c      this is the (x,y) plane z=zf where nzf=iparm(7) =
2 flags
c      a mixed boundary condition of the form

```

```

C
C      dp/dz + alfzf*p(x,y,zf) = gbdzf(x,y)
C
C      in this case kbdy=6,xory=x,yorz=y will be input to
C      bndyc and
C      alfa,gbdy corresponding to alfzf,gbdzf(x,y) must
C      be returned.
C
C
C *** alfa,gbdy must be declared complex.  alfa is
C      constant.
C
C *** bndyc must provide the mixed boundary condition
C      values in correspondence with those flagged in
C      iparm(2)
C      thru iparm(7).  if all boundaries are specified
C      then
C      cud3sp will never call bndyc.  even then it must
C      be entered
C      as a dummy subroutine. bndyc must be declared
C      external in the routine calling cud3sp.  the
C      actual
C      name chosen may be different.
C
C
C
C ... cofx
C
C      a subroutine with arguments (x,cxx,cx,ce)
C      which provides the
C      known complex coefficients of the x derivative
C      terms for the pde
C      at any grid point x.  the name chosen in the
C      calling routine
C      may be different where the coefficient routine
C      must be declared
C      external.
C
C
C ... cofy
C
C      a subroutine with arguments (y,cyy,cy,cey)
C      which provides the
C      known complex coefficients of the y derivative
C      terms for the pde
C      at any grid point y.  the name chosen in the

```

```

calling routine
c      may be different where the coefficient routine
must be declared
c      external.

c
c ... cofz
c
c      a subroutine with arguments (z,czz,cz,cez)
which provides the
c      known complex coefficients of the z derivative
terms for the pde
c      at any grid point z.  the name chosen in the
calling routine
c      may be different where the coefficient routine
must be declared
c      external.

c
c ... rhs
c
c      an array dimensioned nx by ny by nz which
contains
c      the given right hand side values on the
uniform 3-d mesh.
c      rhs(i,j,k) = r(xi,yj,zk) for i=1,...,nx and
j=1,...,ny
c      and k=1,...,nz.

c
c ... phi
c
c      an array dimensioned nx by ny by nz .  on
input phi must
c      contain specified boundary values and an
initial guess
c      to the solution if flagged (see
iguess=iparm(17)=1).  for
c      example, if nyd=iparm(5)=1 then phi(i,ny,k)
must be set
c      equal to p(xi,yd,zk) for i=1,...,nx and
k=1,...,nz prior to
c      calling cud3sp.  the specified values are
preserved by cud3sp.
c
c ***      if no initial guess is given (iguess=0) then
phi must still

```

```
c      be initialized at non-Dirchlet grid points
(this is not
c      checked). these values are projected down and
serve as an initial
c      guess to the pde at the coarsest grid level.
set phi to 0.0 at
c      nonDirchlet grid points if nothing better is
available.
c
c
c ... mgopt
c
c      an integer vector of length 4 which allows
the user to select
c      among various multigrid options. if
mgopt(1)=0 is input then
c      a default set of multigrid arguments (chosen
for robustness)
c      will be internally selected and the
remaining values in mgopt
c      will be ignored. if mgopt(1) is nonzero
then the arguments
c      in mgopt are set internally and defined as
follows: (see the
c      basic coarse grid correction algorithm
below)
c
c
c      kcycle = mgopt(1)
c
c      = 0 if default multigrid options are to be
used
c
c      = 1 if v cycling is to be used (the least
expensive per cycle)
c
c      = 2 if w cycling is to be used (the
default)
c
c      > 2 if more general k cycling is to be used
c      *** warning--values larger than 2 increase
c      the execution time per cycle
considerably and
c      result in the nonfatal error ierror =
```

```

-5
c           which indicates inefficient multigrid
c cycling.
c
c     iprер = mgopt(2)
c
c           the number of "pre-relaxation" sweeps
c executed before the
c           residual is restricted and cycling is
c invoked at the next
c           coarser grid level (default value is 2
c whenever mgopt(1)=0)
c
c     ipost = mgopt(3)
c
c           the number of "post relaxation" sweeps
c executed after cycling
c           has been invoked at the next coarser grid
c level and the residual
c           correction has been transferred back
c (default value is 1
c           whenever mgopt(1)=0).
c
c *** if iprер, ipost, or (especially) kcycle is greater
c than 2
c           than inefficient multigrid cycling has probably
c been chosen and
c           the nonfatal error (see below) ierror = -5 will be
c set. note
c           this warning may be overridden by any other
c nonzero value
c           for ierror.
c
c     interpol = mgopt(4)
c
c           = 1 if multilinear prolongation
c (interpolation) is used to
c           transfer residual corrections and the pde
c approximation
c           from coarse to fine grids within full
c multigrid cycling.
c
c           = 3 if multicubic prolongation
c (interpolation) is used to

```

```

c           transfer residual corrections and the pde
approximation
c           from coarse to fine grids within full
multigrid cycling.
c           (this is the default value whenever
mgopt(1)=0).
c
c *** the default values (2,2,1,3) in the vector mgopt
were chosen for
c       robustness. in some cases v(2,1) cycles with
linear prolongation will
c       give good results with less computation
(especially in two-dimensions).
c       this was the default and only choice in an
earlier version of mudpack
c       (see [1]) and can be set with the integer vector
(1,2,1,1) in mgopt.
c
c *** the schedules for one full multigrid cycle
(iguess=0) using v(2,1)
c       cycles and w(2,1) cycles are depicted for a four
level grid below.
c       the number of relaxation sweeps when each grid is
visited are indicated.
c       the "*" stands for prolongation of the full
approximation and the "."
c       stands for transfer of residuals and residual
corrections within the
c       coarse grid correction algorithm (see below).
c
c       one fmg with v(2,1) cycles:
c
c
c       -----
c       -----2-----1-----
c       level 4
c               *
c               .
c               .
c               *
c               .
c               .
c       -----
c       -----2-----1-----2-----1-----
c       level 3
c               *
c               .
c               .
c               *
c               .
c               .
c               *
c               .
c               .
c       -----
c       -----2-----1-----2-----1-----2-----1-----
c       level 2
c               *
c               .
c               .

```

```

C          *   .   .   .   .
C          ---3---3-----3-----3-----
C          level 1
C
C
C      one fmg with w(2,1) cycles:
C
C          -----
C          -----2-----
--1--      level 4
C          *
C          .
C          -----
C          -----2-----1---2-----3-----
1----      level 3
C          *
C          .
C          -----
C          -----2---1---2---3---1-----2---3---1---2---3---1-
C          level 2
C          *
C          .
C          -----
C          --6---6-----6---6-----6---6-----6---6-----
C          level 1
C
C
C      the form of the "recursive" coarse grid correction
C      cycling used
C      when kcycle.ge.0 is input is described below in
C      pseudo-algorithmic
C      language. it is implemented non-recursively in
C      fortran in mudpack.
C
C      algorithm
cgc(k,l(k),u(k),r(k),kcycle,iprer,ipost,iresw,intpol)
C
C *** approximately solve l(k)*u(k) = r(k) using
C      multigrid iteration
C *** k is the current grid level
C *** l(k) is the discretized pde operator at level k
C *** u(k) is the initial guess at level k
C *** r(k) is the right hand side at level k
C *** i(k,k-1) is the restriction operator from level k
C      to level k-1
C *** (the form of i(k,k-1) depends on iresw)
C *** i(k-1,k) is the prolongation operator from level
C      k-1 to level k
C *** (the form of i(k-1,k) depends on intpol)
C

```

```

c      begin algorithm cgc
c
c ***   pre-relax at level k
c
c      . do (i=1,iprer)
c
c      . . relax(l(k),u(k),r(k))
c
c      . end do
c
c      . if (k > 1) then
c
c ***      restrict the residual from level k to level k-
c
c      . . r(k-1) = i(k,k-1) (r(k)-l(k)*u(k))
c
c      . . kount = 0
c
c      . . repeat
c
c ***      solve for the residual correction at level k-1
c in u(k-1)
c ***      using algorithm cgc "kcycle" times (this is
c the recursion)
c
c      . . . kount = kount+1
c
c      . . . invoke cgc(k-1,l(k-1),u(k-1),r(k-
c 1),kcycle,iprer,ipost,iresh)
c
c
c      . . until (kount.eq.kcycle)
c
c ***      transfer residual correction in u(k-1) to
c level k
c ***      with the prolongation operator and add to u(k)
c
c      . . u(k) = u(k) + i(k-1,k) (u(k-1))
c
c      . end if
c
c ***      post relax at level k
c

```

```

C      . do (i=1,ipost)
C
C      . . relax(l(k),u(k),r(k) )
C
C      . end do
C
C      . return
C
C
C      end algorithm cgc
C
C
C ****output
arguments*****
***

C ****
*****iparm(21)
C
C      on output iparm(21) contains the actual work
space length
C      required for the current grid sizes and
method. This value
C      will be computed and returned even if
iparm(20) is less than
C      iparm(21) (see ierror=9).
C
C
C ... iparm(22)
C
C      if error control is selected (tolmax =
fparm(7) .gt. 0.0) then
C      on output iparm(22) contains the actual
number of cycles executed
C      between the coarsest and finest grid levels
in obtaining the
C      approximation in phi. the quantity
(iprter+ipost)*iparm(22) is
C      the number of relaxation sweeps performed at
the finest grid level.

```

```

c
c
c ... fparm(8)
c
c           on output fparm(8) contains the final
computed maximum relative
c           difference between the last two iterates at
the finest grid level.
c           fparm(8) is computed only if there is error
control (tolmax.gt.0.)
c           assume phil(i,j,k) and phi2(i,j,k) are the
last two computed
c           values for phi(i,j,k) at all points of the
finest grid level.
c           if we define
c
c           phdif = max (abs(phi2(i,j,k)-phil(i,j,k)))
for all i,j,k
c
c           and
c
c           phmax = max (abs(phi2(i,j,k))) for all
i,j,k
c
c           then
c
c           fparm(8) = phdif/phmax
c
c           is returned whenever phmax.gt.0.0. in the
degenerate case
c           phmax = 0.0, fparm(8) = phdif is returned.
c
c
c
c ... work
c
c           on output work contains intermediate values
that must not be
c           destroyed if cud3sp is to be called again
with iparm(1)=1 or
c           if cud34sp is to be called to improve the
estimate to fourth
c           order.
c

```

```

c ... phi
c
c           on output phi(i,j,k) contains the
approximation to
c           p(xi,yj,zk) for all mesh points i=1,...,nx;
j=1,...,ny;
c           k=1,...,nz. the last computed iterate in phi
is returned
c           even if convergence is not obtained (ierror=-
1)
c
c ... ierror
c
c           For intl=iparm(1)=0 initialization calls,
ierror is an
c           error flag that indicates invalid input
arguments when
c           returned positive and nonfatal warnings when
returned
c           negative. Argument checking and
discretization
c           is bypassed for intl=1 calls which can only
return
c           ierror = -1 or 0 or 1.
c
c
c           non-fatal warnings * * *
c
c
c           ==5 if kcycle=mgopt(1) is greater than 2. values
larger than 2 results
c           in an algorithm which probably does far more
computation than
c           necessary. kcycle = 1 (v cycles) or kcycle=2
(w cycles) should
c           suffice for most problems. ierror = -5 is
also set if either
c           iprer = mgopt(2) or ipost=mgopt(3) is greater
than 2. the
c           ierror=-5 flag is overridden by any other
fatal or non-fatal
c           error.
c
c           ==4 if there are dominant nonzero first order

```

```

terms in the pde which
c           make it "hyperbolic" at the finest grid level.
numerically, this
c           happens if:
c
c           abs(cx)*dlx > 2.*abs(cxx)      (dlx = (xb-
xa) / (nx-1))
c
c                           (or)
c
c           abs(cy)*dly > 2.*abs(cyy)      (dly = (yd-
yc) / (ny-1))
c
c
c           at some fine grid point (xi,yj). if an
adjustment is not made the
c           condition can lead to a matrix coming from the
discretization
c           which is not diagonally dominant and
divergence is possible. since
c           the condition is "likely" at coarser grid
levels for pde's with
c           nonzero first order terms, the adjustments
(actualy first order
c           approximations to the pde)
c
c
c           cxx = amax1(cxx,0.5*abs(cx)*dx)
c
c                           (and)
c
c           cyy = amax1(cyy,0.5*abs(cy)*dy)
c
c
c           (here dx,dy are the x,y mesh sizes of the
subgrid)
c
c           are made to preserve convergence of multigrid
iteration. if made
c           at the finest grid level, it can lead to
convergence to an
c           erroneous solution (flagged by ierror = -4).
a possible remedy
c           is to increase resolution. the ierror = -4

```

```
flag overrides the
c           nonfatal ierror = -5 flag.
c
c
c     ==3  if the continuous elliptic pde is singular.
this means the
c           boundary conditions are periodic or pure
derivative at all
c           boundaries and ce(x,y) = 0.0 for all x,y.  a
solution is still
c           attempted but convergence may not occur due
to ill-conditioning
c           of the linear system coming from the
discretization.  the
c           ierror = -3 flag overrides the ierror=-4 and
ierror=-5 nonfatal
c           flags.
c
c
c     ==2  nonellipticity is not checked with cud3sp so
this flag is not set.
c           (compare with cud3.f)
c
c
c     ==1  if convergence to the tolerance specified in
tolmax=fparm(5)>0.
c           is not obtained in maxcy=iparm(13) multigrid
cycles between the
c           coarsest (ixp+1,jyq+1) and finest (nx,ny)
grid levels.
c           in this case the last computed iterate is
still returned.
c           the ierror = -1 flag overrides all other
nonfatal flags
c
c
c     no errors * * *
c
c     = 0
c
c     fatal argument errors * * *
c
c     = 1 if intl=iparm(1) is not 0 on initial call or
not 0 or 1
```

```

C          on subsequent calls
C
C      = 2 if any of the boundary condition flags
nxa,nxb,nyc,nyd,nze,nzf
C          in iparm(2) through iparm(7) is not 0,1 or 2 or
if
C          (nxa,nxb) or (nyc,nyd) or (nze,nzf) are not
pairwise zero.
C
C      = 3 if mino(ixp,jyq,kzr) < 2
(ixp=iparm(8),jyq=iparm(9),kzr=iparm(10))
C
C      = 4 if min0(iex,jey,kez) < 1
(iex=iparm(11),jey=iparm(12),kez=iparm(13))
C          or if max0(iex,jey,kez) > 50
C
C      = 5 if nx.ne.ixp*2** (iex-1)+1 or if
ny.ne.jyq*2** (jey-1)+1 or
C          if nz.ne.kzr*2** (kez-1)+1
(nx=iparm(14),ny=iparm(15),nz=iparm(16))
C
C      = 6 if iguess = iparm(17) is not equal to 0 or 1
C
C      = 7 if maxcy = iparm(18) < 1 (large values for
maxcy should not be used)
C
C      = 8 if method = iparm(19) is not equat to zero
C
C      = 9 if length = iparm(20) is too small (see
iparm(21) on output
C          for minimum required work space length)
C
C      =10 if xa >= xb or yc >= yd or ze >= zf
C
(xa=fparm(1),xb=fparm(2),yc=fparm(3),yd=fparm(4),ze=fpar
m(5),zf=fparm(6))
C
C      =11 if tolmax = fparm(7) < 0.0
C
C      errors in setting multigrid options * * * (see
also ierror=-5)
C
C      =12 if kcycle = mgopt(1) < 0 or
C          if iprer = mgopt(2) < 1 or

```

```
C           if ipost = mgopt(3) < 1 or
C           if interpol = mgopt(4) is not 1 or 3
C
C
C*****                                                 *****
*
C*****                                                 *****
*
C*****                                                 *****
*
C*****                                                 *****
*
C      end of cud3sp documentation
C
C
C*****                                                 *****
**
C*****                                                 *****
**
C*****                                                 *****
*
C*****                                                 *****
*
C*****                                                 *****

```

---

## CUH2

```
C
C      file cuh2.d
C
C      * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *                               copyright (c) 2008 by UCAR
*
C      *
*
C      *               University Corporation for Atmospheric
Research          *
C      *
*
```

C \* all rights reserved  
\*  
C \*  
\*  
C \* MUDPACK version 5.0.1  
\*  
C \*  
\*  
C \* A Fortran Package of Multigrid  
\*  
C \*  
\*  
C \* Subroutines and Example Programs  
\*  
C \*  
\*  
C \* Equations \* for Solving Elliptic Partial Differential  
C \*  
\*  
C \* by  
\*  
C \*  
\*  
C \* John Adams  
\*  
C \*  
\*  
C \* of  
\*  
C \*  
\*  
C \* the National Center for Atmospheric  
Research \*  
C \*  
\*  
C \* Boulder, Colorado (80307)  
U.S.A. \*  
C \*  
\*  
C \* which is sponsored by  
\*  
C \*  
\*

```
C      *          the National Science Foundation
*
C      *
*
C      * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C
C ... file cuh2.d
C
C      contains documentation for:
C      subroutine
cuh2(iparm,fparm,wk,iwk,coef,bndyc,rhs,phi,mgopt,ierror)
C      a sample fortran driver is file "tcuh2.f".
C
C ... required mudpack files
C
C      cudcom.f, cuhcom.f
C
C ... purpose
C
C      the "hybrid" multigrid/direct method code cuh2
approximates the
C      same 2-d nonseparable elliptic pde as the mudpack
solver cud2.
C      cuh2 combines the efficiency of multigrid
iteration with the certainty
C      a direct method. the basic algorithm is modified
by using banded
C      gaussian elimination in place of relaxation
whenever the coarsest
C      subgrid is encountered within multigrid cycling.
this provides
C      additional grid size flexibility by eliminating
the usual multigrid
C      constraint that the coarsest grid consist of "few"
points for effective
C      error reduction with multigrid cycling. In many
cases the hybrid method
C      provides more robust convergence characteristics
than multigrid cycling
C      alone.
C
C      The form of the pde solved is:
C
```

```

c
c           cxx(x,y)*pxx + cyy(x,y)*pyy + cx(x,y)*px +
cy(x,y)*py +
c
c           ce(x,y)*p(x,y) = r(x,y).
c
c
c     pxx,pyy,px,py are second and first partial
derivatives of the
c     unknown complex solution function p(x,y) with
respect to the
c     independent variables x,y. cxx,cyy,cx,cy,ce are
the known
c     complex coefficients of the elliptic pde and
r(x,y) is the known
c     complex right hand side of the equation.
Nonseparability means
c     some of the coefficients depend on both x and y.
if the pde
c     is separable subroutine cud2sp should be used
instead
c     of cud2 or cuh2.
c
c *** cuh2 becomes a full direct method if grid size
arguments are chosen
c     so that the coarsest and finest grids coincide.
choosing iex=jey=1
c     and ixp=nx-1, jyq=ny-1
(iex=iparm(6),jey=iparm(7),ixp=iparm(8),
c     jyq=iparm(9),nx=iparm(10),ny=iparm(11)) will set
gaussian elimination
c     on the nx by ny grid.
c
c
c ... argument differences with cud2.f
c
c     the input and output arguments of cuh2 are almost
identical to the
c     arguments of cud2 (see cud2.d) with the following
exceptions:
c
c     (1) the complex work space vector "wk" requires
c
c           (ixp+1)*(jyq+1)*(2*ixp+3)

```

```

c
c           additional words of storage (ixp = iparm(6),
jyq = iparm(7))
c           if periodic boundary conditions are not
flagged in the y direction
c           (nyc .ne. 0 where nyc = iparm(4)) or
c
c           (ixp+1)*[2*(ixp+1)*(2*jyq-1)+jyq+1]
c
c           additional words of storage if periodic
boundary conditions are
c           flagged in the y direction (nyc = 0). the
extra work space is
c           used for a direct solution with gaussian
elimination whenever the
c           coarsest grid is encountered within multigrid
cycling.
c
c     (2) An integer work space iwk of length at least
(ixp+1)*(jyq+1)
c           must be provided.
c
c     (3) jyq must be greater than 2 if periodic
boundary conditions
c           are flagged in the y direction and ixp must be
greater than
c           2 if periodic boundary conditions are flagged
in the x direction.
c           inputting jyq = 2 when nyc = 0 or inputting
ixp = 2 when nxa = 0
c           will set the fatal error flag ierror=3
c
c *** (4) it is no longer necessary that ixp and jyq be
"small" for
c           effective error reduction with multigrid
iteration. there
c           is no reduction in convergence rates when
larger values for
c           ixp or jyq are used . this provides
additional flexibility
c           in choosing grid size. in many cases cuh2
provides more
c           robust convergence than cud2. it can be used
in place of

```

```
c          cud2 for all nonsingular problems (see (5)
below) .

c
c      (5) iguess = iparm(11) = 1 (flagging an initial
guess) or
c          maxcy = iparm(14) > 1 (setting more than one
multigrid
c          cycle) are not allowed if cuh2 becomes a full
direct method
c          by choosing iex = jey = 1 (iex = iparm(8),jey
= iparm(9)).
c          this conflicting combination of input
arguments for multigrid
c          iteration and a full direct method set the
fatal error flag
c
c          ierror = 13
c
c          iguess = 0 and maxcy = 1 are required when
cuh2 becomes a
c          full direct method.

c
c      (6) if a "singular" pde is detected (see ierror=-3
description in cud2.d;
c          ce(x,y) = 0.0 for all x,y and the boundary
conditions are a combination
c          of periodic and/or pure derivatives) then cuh2
sets the fatal error
c          flag
c
c          ierror = 14
c
c          The direct method utilized by cuh2 would
likely cause a division
c          by zero in the singular case.  cud2 can be
tried for singular problems
c
c
c ... grid size considerations
c
c      (1) flexibility
c
c          cuh2 should be used in place of cud2 whenever
grid size
```

```
c      requirements do not allow choosing ixp and jyq
c      to be "small"
c      positive integers (typically less than 4).
c
c      example:
c
c      suppose we wish to solve an elliptic pde on a
c      one degree grid on
c      the full surface of a sphere. choosing ixp =
c      jyq = 45 and iex = 4
c      and jyq = 3 fits the required 361 by 181 grid
c      exactly. multigrid
c      cycling will be used on the sequence of
c      subgrid sizes:
c
c      46 x 46 < 91 x 46 < 181 x 91 < 361 x
c      181
c
c      the 46 x 46 coarsest subgrid has too much
c      resolution for effective
c      error reduction with relaxation only. cuh2
c      circumvents this
c      difficulty by generating an exact direct
c      solution (modulo roundoff
c      error) whenever the coarsest grid is
c      encountered.
c
c      (2) additional work space (see (1) under
c      "arguments differences") is
c      required by cuh2 to implement gaussian
c      elimination at the coarsest
c      grid level. this may limit the size of ixp
c      and jyq.
c
c      (3) operation counts
c
c      for simplicity, assume p = ixp = jyq and n =
c      nx = ny. banded
c      gaussian elimination requires  $\mathcal{O}(p^{**4})$ 
c      operations for solution
c      on the coarsest subgrid while multigrid
c      iteration is a  $\mathcal{O}(n^{**2})$ 
c      algorithm. these are approximately balanced
c      when
```

```

C
C      p**4 =: (n/ (2**k) ) **4 =: n**2
C
C      or
C
C      k =: log2(n)/2
C
C      grid levels are chosen with the hybrid method.
so if
C      p is approximately equal to
C
C      n/ (2** (log2(n)/2))
C
C      then the direct method and multigrid parts of
the hybrid algorithm
C      require roughly the same amount of computer
time. larger values
C      for p mean the direct method will dominate the
computation. smaller
C      values mean the hybrid method will cost only
marginally more than
C      multigrid iteration with coarse grid
relaxation.
C
C
C *** the remaining documentation is almost identical to
cud2.d
C      except for the modifications already indicated.
C
C ... mesh description . . .
C
C      the approximation is generated on a uniform nx by
ny grid. the grid
C      is superimposed on the rectangular solution region
C
C      [xa,xb] x [yc,yd].
C
C      let
C
C      dlx = (xb-xa) / (nx-1), dly = (yd-yc) / (ny-1)
C
C      be the uniform grid increments in the x,y
directions. then
C
```

```
c           xi=xa+(i-1)*dlx,   yj=yc+(j-1)*dly
c
c       for i=1,...,nx and j=1,...,ny  denote the x,y
uniform mesh points
c
c
c ... language
c
c       fortran90/fortran77
c
c
c ... portability
c
c       mudpack5.0.1 software has been compiled and tested
with fortran77
c       and fortran90 on a variety of platforms.
c
c ... methods
c
c       details of the methods employed by the solvers in
mudpack are given
c       in [1,9].  [1,2,9] contain performance
measurements on a variety of
c       elliptic pdes (see "references" in the file
"readme").  in summary:
c
c *** discretization and solution (second-order solvers)
(see [1])
c
c       the pde and boundary conditions are automatically
discretized at all
c       grid levels using second-order finite difference
formula.  diagonal
c       dominance at coarser grid levels is maintained in
the presence of
c       nonzero first-order terms by adjusting the second-
order coefficient
c       when necessary.  the resulting block tri-diagonal
linear system is
c       approximated using multigrid iteration
[10,11,13,15,16,18].  version
c       5.0.1 of mudpack uses only fully weighted residual
restriction.  defaults
c       include cubic prolongation and w(2,1) cycles.
```

these can be overridden  
c with selected multigrid options (see "mgopt").  
error control based on  
c maximum relative differences is available. full  
multigrid cycling (fmg)  
c or cycling beginning or restarting at the finest  
grid level can be  
c selected. a menu of relaxation methods including  
gauss-seidel point,  
c line relaxation(s) (in any combination of  
directions) and planar  
c relaxation (for three-dimensional anisotropic  
problems) are provided.  
c all methods use ordering based on alternating  
points (red/black),  
c lines, or planes for cray vectorization and  
improved convergence  
c rates [14].  
c  
c \*\*\* higher order solution (fourth-order solvers) (see  
[9,19,21])  
c  
c if the multigrid cycling results in a second-order  
estimate (i.e.,  
c discretization level error is reached) then this  
can be improved to a  
c fourth-order estimate using the technique of  
"deferred corrections."  
c the values in the solution array are used to  
generate a fourth-order  
c approximation to the truncation error. second-  
order finite difference  
c formula are used to approximate third and fourth  
partial derivatives  
c of the solution function [3]. the truncation  
error estimate is  
c transferred down to all grid levels using weighted  
averaging where  
c it serves as a new right hand side. the default  
multigrid options  
c are used to compute the fourth-order correction  
term which is added  
c to the original solution array.  
c

```
C
C ... references (partial)
C
C
C      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
C      Solution of Linear Elliptic Partial Differential
Equations,"
C      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
C
C      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
C      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
C      1989, pp.1-12.
C      .
C      .
C      .
C      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
Haidvogel, and V. Pizzo,
C      "Applications of Multigrid Software in the
Atmospheric Sciences,"
C      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-
1458.
C      .
C      .
C      .
C      [9] J. Adams, "Recent Enhancements in MUDPACK, a
Multigrid Software
C      package for Elliptic Partial Differential
Equations," Applied Math.
C      and Comp., 1991, vol. 43, May 1991, pp. 79-94.
C
C      [10]J. Adams, "MUDPACK-2: Multigrid Software for
Approximating
C      Elliptic Partial Differential Equations on Uniform
Grids with
C      any Resolution," Applied Math. and Comp., 1993,
vol. 53, February
C      1993, pp. 235-249
C      .
C      .
C      .
```

```
C
C ... argument description
C
C
C
*****
*****  

C *** input arguments
*****
C
*****
C
C ... iparm
C
C      an integer vector of length 17 used to pass
integer
C      arguments. iparm is set internally and
defined as
C      follows:
C
C
C ... intl=iparm(1)
C
C      an initialization argument. intl=0 must be
input
C      on an initial call. in this case input
arguments will
C      be checked for errors and the elliptic
partial differential
C      equation and boundary conditions will be
discretized using
C      second order finite difference formula.
C
C ***      an approximation is not generated after an
intl=0 call!
C      cuh2 should be called with intl=1 to
approximate the elliptic
C      pde discretized by the intl=0 call. intl=1
should also
C      be input if cuh2 has been called earlier and
only the
C      values in in rhs (see below) or gbdy (see
```

```
bndyc below)
c           or phi (see below) have changed.  this will
bypass
c           redundant pde discretization and argument
checking
c           and save computational time.  some examples
of when
c           intl=1 calls should be used are:
c
c           (0) after a intl=0 argument checking and
discretization call
c
c           (1) cuh2 is being recalled for additional
accuracy.  in
c           this case iguess=iparm(12)=1 should also
be used.
c
c           (2) cuh2 is being called every time step in a
time dependent
c           problem (see discussion below) where the
elliptic operator
c           does not depend on time.
c
c           (3) cuh2 is being used to solve the same
elliptic equation
c           for several different right hand sides
(igueess=0 should
c           probably be used for each new righthand
side).
c
c           intl = 0 must be input before calling with
intl = 1 when any
c           of the following conditions hold:
c
c           (a) the initial call to cuh2
c
c           (b) any of the integer arguments other than
igueess=iparm(12)
c           or maxcy=iparm(13) or mgopt have changed
since the previous
c           call.
c
c           (c) any of the floating point arguments other
than tolmax=
```

```
c           fparm(5) have changed since the previous
call
c
c           (d) any of the coefficients input by coef
(see below) have
c           changed since the previous call
c
c           (e) any of the "alfa" coefficients input by
bndyc (see below)
c           have changed since the previous call.
c
c           if any of (a) through (e) are true then the
elliptic pde
c           cust be discretized or rediscretized.  if
none of (a)
c           through (e) holds, calls can be made with
intl=1.
c           incorrect calls with intl=1 will produce
erroneous results.
c ***      the values set in the saved work space "work"
(see below) with
c           an intl=0 call must be preserved with
subsequent intl=1 calls.
c
c           MUDPACK software performance should be
monitored for intl=1
c           calls.  The intl=0 discretization call
performance depends
c           primarily on the efficiency or lack of
efficiency of the
c           user provided subroutines for pde
coefficients and
c           boundary conditions.
c
c
c ... nxa=iparm(2)
c
c           flags boundary conditions on the edge x=xa
c
c           = 0 if p(x,y) is periodic in x on [xa,xb]
c           (i.e., p(x+xb-xa,y) = p(x,y) for all x,y
c           (if nxa=0 then nxb=0 is required, see
ierror = 2)
c
```

```

c      = 1 if p(xa,y) is specified (this must be input
thru phi(1,j))
c
c      = 2 if there are mixed derivative boundary
conditions at x=xa
c          (see bndyc)
c
c
c ... nxb=iparm(3)
c
c          flags boundary conditions on the edge x=xb
c
c      = 0 if p(x,y) is periodic in x on [xa,xb]
c          (i.e., p(x+xb-xa,y) = p(x,y) for all x,y)
c          (if nxb=0 then nxa=0 is required, see
ierror = 2)
c
c      = 1 if p(xb,y) is specified (this must be input
thru phi(nx,j))
c
c      = 2 if there are mixed derivative boundary
conditions at x=xb
c          (see bndyc)
c
c
c ... nyc=iparm(4)
c
c          flags boundary conditions on the edge y=yc
c
c      = 0 if p(x,y) is periodic in y on [yc,yd]
c          (i.e., p(x,y+yd-yc) = p(x,y) for all x,y)
c          (if nyc=0 then nyd=0 is required, see
ierror = 2)
c
c      = 1 if p(x,yc) is specified (this must be input
thru phi(i,1))
c
c      = 2 if there are mixed derivative boundary
conditions at y=yc
c          (see bndyc)
c
c
c ... nyd=iparm(5)
c

```

```

c           flags boundary conditions on the edge y=yd
c
c           = 0 if p(x,y) is periodic in y on [yc,yd]
c                   (i.e., p(x,y+yd-yc) = p(x,y) for all x,y
c                   (if nyd=0 then nyc=0 is required, see
ierror = 2)
c
c           = 1 if p(x,yd) is specified (this must be input
thru phi(i,ny))
c
c           = 2 if there are mixed derivative boundary
conditions at y=yd
c                   (see bndyc)
c
c
c *** grid size arguments
c
c
c ... ixp = iparm(6)
c
c           an integer greater than one which is used in
defining the number
c           of grid points in the x direction (see nx =
iparm(10)). "ixp+1"
c           is the number of points on the coarsest x
grid visited during
c           multigrid cycling. ixp should be chosen as
small as possible.
c           recommended values are the small primes 2 or
3.
c           larger values can reduce multigrid
convergence rates considerably,
c           especially if line relaxation in the x
direction is not used.
c           if ixp > 2 then it should be 2 or a small odd
value since a power
c           of 2 factor of ixp can be removed by
increasing iex = iparm(8)
c           without changing nx = iparm(10).
c
c
c ... jyq = iparm(7)
c
c           an integer greater than one which is used in

```

```

defining the number
c          of grid points in the y direction (see ny =
iparm(11)). "jyq+1"
c          is the number of points on the coarsest y
grid visited during
c          multigrid cycling. jyq should be chosen as
small as possible.
c          recommended values are the small primes 2 or
3.
c          larger values can reduce multigrid
convergence rates considerably,
c          especially if line relaxation in the y
direction is not used.
c          if jyq > 2 then it should be 2 or a small odd
value since a power
c          of 2 factor of jyq can be removed by
increasing jey = iparm(9)
c          without changing ny = iparm(11).

c
c
c ... iex = iparm(8)
c
c          a positive integer exponent of 2 used in
defining the number
c          of grid points in the x direction (see nx =
iparm(10)).
c          iex .le. 50 is required. for efficient
multigrid cycling,
c          iex should be chosen as large as possible and
ixp=iparm(8)
c          as small as possible within grid size
constraints when
c          defining nx.

c
c
c ... jey = iparm(9)
c
c          a positive integer exponent of 2 used in
defining the number
c          of grid points in the y direction (see ny =
iparm(11)).
c          jey .le. 50 is required. for efficient
multigrid cycling,
c          jey should be chosen as large as possible and

```

```

jyq=iparm(7)
c           as small as possible within grid size
constraints when
c           defining ny.
c
c
c
c ... nx = iparm(10)
c
c           the number of equally spaced grid points in
the interval [xa,xb]
c           (including the boundaries).  nx must have the
form
c
c           nx = ixp*(2** (iex-1)) + 1
c
c           where ixp = iparm(6), iex = iparm(8).
c
c
c ... ny = iparm(11)
c
c           the number of equally spaced grid points in
the interval [yc,yd]
c           (including the boundaries).  ny must have the
form:
c
c           ny = jyq*(2** (jey-1)) + 1
c
c           where jyq = iparm(7), jey = iparm(9).
c
c
c *** example
c
c           suppose a solution is wanted on a 33 by 97
grid.  then
c           ixp=2, jyq=6 and iex=jey=5 could be used.  a
better
c           choice would be ixp=2, jyq=3, and iex=5,
jey=6.
c
c *** grid size flexibility considerations:
c
c           the hybrid multigrid/direct method code cuh2
provides more grid size

```

```

c      flexibility than cud2 by removing the constraint
that ixp and jyq are
c      2 or 3. this is accomplished by using a direct
method whenever the
c      coarsest (ixp+1) x (jyq+1) grid is encountered in
multigrid cycling.
c      if nx = ixp+1 and ny = jyq+1 then cuh2 becomes a
full direct method.
c      cuh2 is roughly equivalent to cud2 in efficiency
as long as ixp and
c      jyq remain "small". if the problem to be
approximated requires
c      a grid neither cud2 or cuh2 can exactly fit then
another option
c      is to generate an approximation on a "close grid"
using cud2 or cuh2.
c      then transfer the result to the required grid
using cubic interpolation
c      via the package "regridpack" (contact john adams
about this software)
c
c *** note
c
c      let G be the nx by ny fine grid on which the
approximation is
c      generated and let n = max0(iex,jey). in mudpack,
multigrid
c      cycling is implemented on the ascending chain of
grids
c
c          G(1) < ... < G(k) < ... < G(n) = g.
c
c      each G(k) (k=1,...,n) has mx(k) by my(k) grid
points
c      given by:
c
c          mx(k) = ixp*[2** (max0(iex+k-n,1)-1)] + 1
c
c          my(k) = jyq*[2** (max0(jey+k-n,1)-1)] + 1
c
c      If iex = jey = 1 then G(1) = G(n) and cuh2 solves
the problem
c      directly with block banded Gaussian elimination.
Otherwise

```

```
c      cuh2 replaces relaxation with a direct method on
G(1).
c
c ... iguess=iparm(12)
c
c           = 0 if no initial guess to the pde is
provided
c
c           = 1 if an initial guess to the pde is at the
finest grid
c           level is provided in phi (see below)
c
c   comments on iguess = 0 or 1 . . .
c
c   even if iguess = 0, phi must be initialized at all
grid points (this
c   is not checked). phi can be set to 0.0 at non-
dirchlet grid points
c   if nothing better is available. the values set in
phi when iguess = 0
c   are passed down and serve as an initial guess to
the pde at the coarsest
c   grid level where cycling commences. in this
sense, values input in
c   phi always serve as an initial guess. setting
iguess = 0 forces full
c   multigrid cycling beginning at the coarsest and
finishing at the finest
c   grid level.
c
c   if iguess = 1 then the values input in phi are an
initial guess to the
c   pde at the finest grid level where cycling begins.
this option should
c   be used only if a "very good" initial guess is
available (as, for
c   example, when restarting from a previous iguess=0
call).
c
c   time dependent problems . . .
c
c *** assume we are solving an elliptic pde every time
step in a
c   marching problem of the form:
```

```

c
c           l(p(t)) = r(t)
c
c   where the differential operator "l" has no time
dependence,
c   "p(t)" is the solution and "r(t)" is the right
hand side at
c   current time "t". let "dt" be the increment
between time steps.
c   then p(t) can be used as an initial guess to
p(t+dt) with
c   intl = 1 when solving
c
c           l(p(t+dt)) = r(t+dt).
c
c   after the first two time steps, rather than
continue, it would
c   be better to define the "correction" term:
c
c           e(t,dt) = p(t+dt) - p(t)
c
c   this clearly satisfies the equation
c
c           l(e(t,dt)) = r(t+dt) - r(t).
c
c   this should be solved with iguess = 0 and intl =
1. boundary
c   conditions for e(t,dt) are obtained from the
boundary conditions
c   for p(t) by subtracting given values at t from
given values at
c   t+dt. for example if
c
c           d(p(t))/dx = f(t), d(p(t+dt))/dx = f(t+dt)
c
c   at some x boundary then e(t,dt) satisfies the
derivative
c   boundary condition
c
c           d(e(t,dt))/dx = f(t+dt) - f(t).
c
c   e(t,dt) can be preset to 0.0 (at nondirichlet
points) or (if p(t-dt)
c   is saved) to p(t)-p(t-dt). with iguess = 0, these

```

```
values will serve
c      as an initial guess to e(t,dt) at the coarsest
grid level. this
c      approach has the advantage that a full sequence of
multigrid cycles,
c      beginning at the coarsest grid level, is invoked
every time step in
c      solving for e(t,dt). a few digits of accuracy in
e(t,dt), which is
c      ordinarily much smaller than p(t), will yield
several more digits of
c      accuracy in the final approximation:
c
c          p(t+dt) = p(t) + e(t,dt).
c
c      using this approach to integrate in time will give
more accuracy
c      then using p(t) as an initial guess to p(t+dt) for
all time steps.
c      it does require additional storage.
c
c      if the differential operator "l" has time
dependence (either thru
c      the coefficients in the pde or the coefficients in
the derivative
c      boundary conditions) then use p(t) as an initial
guess to p(t+dt)
c      when solving
c
c          l(t+dt) (p(t+dt)) = r(t+dt)
c
c      with intl = 0 for all time steps (the
discretization must be repeated
c      for each new "t"). either iguess = 0 (p(t) will
then be an initial
c      guess at the coarsest grid level where cycles will
commence) or
c      iguess = 1 (p(t) will then be an initial guess at
the finest grid
c      level where cycles will remain fixed) can be
tried.
c
c
c ... maxcy = iparm(13)
```

```

c
c           the exact number of cycles executed between
the finest (nx by
c           ny) and the coarsest ((ixp+1) by (jyq+1))
grid levels when
c           tolmax=fparm(5)=0.0 (no error control). when
tolmax > 0.0
c           is input (error control) then maxcy is a
limit on the number
c           of cycles between the finest and coarsest
grid levels. in
c           any case, at most maxcy*(iprert+ipost)
relaxation sweeps are
c           are performed at the finest grid level (see
iprert=mgopt(2),
c           ipost=mgopt(3) below). when multigrid
iteration is working
c           "correctly" only a few are required for
convergence. large
c           values for maxcy should not be necessary.
c
c
c ... method = iparm(14) determines the method of
relaxation
c           (gauss-seidel based on alternating points
or lines)
c
c           = 0 for point relaxation
c
c           = 1 for line relaxation in the x direction
c
c           = 2 for line relaxation in the y direction
c
c           = 3 for line relaxation in both the x and y
direction
c
c
c *** choice of method. . .
c
c       let fx represent the quantity
cabs(cxx(x,y))/dlx**2 over the solution region.
c
c       let fy represent the quantity
cabs(cyy(x,y))/dly**2 over the solution region

```

```

c
c      if fx,fy are roughly the same size and do not vary
too much over
c      the solution region choose method = 0.  if this
fails try method=3.
c
c      if fx is much greater than fy choose method = 1.
c
c      if fy is much greater than fx choose method = 2
c
c      if neither fx or fy dominates over the solution
region and they
c          both vary considerably choose method = 3.
c
c
c ... length = iparm(15)
c
c          the length of the work space provided in
vector work (see below).
c          let isx = 0 if method = 0 or method = 2
c          let isx = 3 if method = 1 or method = 3 and
nxa.ne.0
c          let isx = 5 if method = 1 or method = 3 and
nxa.eq.0
c          let jsy = 0 if method = 0 or method = 1
c          let jsy = 3 if method = 2 or method = 3 and
nyc.ne.0
c          let jsy = 5 if method = 2 or method = 3 and
nyc.eq.0
c
c          let ldir = (ixp+1)*(jyq+1)*(2*ixp+3) if
nyc.ne.0 or
c          let ldir = (ixp+1)*[2*(ixp+1)*(2*jyq-
1)+jyq+1] if nyc=0
c
c          then . . .
c
c          length =
4*[nx*ny*(10+isx+jsy)+8*(nx+ny+2)]/3 + ldir
c
c          will suffice in most cases.  the exact
minimal work space
c          length required for the current nx,ny and
method is output

```

```
c           in iparm(16) (even if iparm(15) is too
small).  this will be
c           less then the value given by the simplified
formula above
c           in most cases.
c
c
c ... fparm
c
c           a floating point vector of length 6 used to
efficiently
c           pass floating point arguments.  fparm is set
internally
c           in cuh2 and defined as follows . . .
c
c
c ... xa=fparm(1), xb=fparm(2)
c
c           the range of the x independent variable.  xa
must
c           be less than xb
c
c
c ... yc=fparm(3), yd=fparm(4)
c
c           the range of the y independent variable.  yc
must
c           be less than yd.
c
c
c ... tolmax = fparm(5)
c
c           when input positive, tolmax is a maximum
relative error tolerance
c           used to terminate the relaxation iterations.
assume phil(i,j)
c           and phi2(i,j) are the last two computed
approximations at all
c           grid points of the finest grid level. if we
define
c
c           phdif = max(cabs(phi2(i,j)-phil(i,j)))
for all i,j
c
```

```
c           and
c
c           phmax = max(cabs(phi2(i,j))) for all i,j
c
c           then "convergence" is considered to have
occurred if and only if
c
c           phdif/phmax < tolmax.
c
c
c           if tolmax=iparm(5)=0.0 is input then there is
no error control
c           and maxcy cycles from the finest grid level
are executed. maxcy
c           is a limit which cannot be exceeded even with
error control.
c           *** calls with tolmax=0.0, when appropriate
because of known
c           convergence behavior, are more efficient than
calls with tolmax
c           positive (i.e., if possible DO NOT error
control!).
c
c ... wk
c
c           a one dimensional complex saved work space
(see iparm(15) for
c           length) which must be preserved from the
previous call when
c           calling with intl=iparm(1)=1.
c
c ... iwk
c
c           an integer vector dimensioned of length at
least (ixp+1)*(jyq+1)
c           (ixp = iparm(6), jyq=iparm(7)) in the routine
calling cuh2.
c           The length of iwk is not checked! If iwk has
length less than
c           (ixp+1)*(jyq+1) then undetectable errors will
result.
c
c ... bndyc
```

```

c           a subroutine with arguments
(kbdy,xory,alfa,gbdy) which
c           are used to input mixed boundary conditions
to cuh2. bndyc
c           must be declared "external" in the program
calling cuh2.
c           kbdy is type integer, xory type real,
alfa,gbdy type complex
c           the boundaries are numbered one thru four and
the mixed
c           derivative boundary conditions are described
below (see the
c           sample driver code "tcuh2.f" for an example
of how bndyc is
c           can beset up).

c
c           * * * * * * * * * * * * * * * y=yd
c           *         kbdy=4      *
c           *
c           *
c           *
c           *
c           * kbdy=1      kbdy=2 *
c           *
c           *
c           *
c           *
c           *         kbdy=3      *
c           * * * * * * * * * * * * * * * y=yc
c
c           x=xa          x=xb
c
c
c           (1) the kbdy=1 boundary
c
c           this is the edge x=xa where nx=iparm(2)=2
flags
c           a mixed boundary condition of the form
c
c           dp/dx + alfx(x,y)*p(xa,y) = gbdxa(y)
c
c           in this case kbdy=1,xory=y will be input to
bndyc and
c           alfa,gbdy corresponding to alfx(x,y),gbdxa(y)
must be returned.
c

```

```

C
C          (2) the kbdy=2 boundary
C
C          this is the edge x=xb where nxb=iparm(3)=2
flags      a mixed boundary condition of the form
C
C           $\frac{dp}{dx} + \text{alfxb}(y) * p(xb, y) = \text{gbdxb}(y)$ 
C
C          in this case kbdy=2, xory=y, will be input to
bndyc and
C          alfa, gbdy corresponding to alfxb(y), gbdxb(y)
must be returned.

C
C
C          (3) the kbdy=3 boundary
C
C          this is the edge y=yc where nyc=iparm(4)=2
flags      a mixed boundary condition of the form
C
C           $\frac{dp}{dy} + \text{alfyc}(x) * p(x, yc) = \text{gbdyc}(x)$ 
C
C          in this case kbdy=3, xory=x will be input to
bndyc and
C          alfa, gbdy corresponding to alfy(c)(x), gbdyc(x)
must be returned.

C
C
C          (4) the kbdy=4 boundary
C
C          this is the edge y=yd where nyd=iparm(5)=2
flags      a mixed boundary condition of the form
C
C           $\frac{dp}{dy} + \text{alfyd}(x) * p(x, yd) = \text{gbdyd}(x)$ 
C
C          in this case kbdy=4, xory=x will be input to
bndyc and
C          alfa, gbdy corresponding to alfyd(x), gbdyd(x)
must be returned.

C
C
C ***      bndyc must provide the mixed boundary

```

```

condition values
c           in correspondence with those flagged in
iparm(2) thru
c           iparm(5). if all boundaries are specified or
periodic
c           cuh2 will never call bndyc. even then it
must be entered
c           as a dummy subroutine. bndyc must be declared
"external"
c           in the routine calling cuh2. the actual name
chosen may
c           be different.

c
c
c ... coef
c
c           a subroutine with arguments
(x,y,cxx,cyy,cx,cy,ce) which
c           provides the known complex coefficients for
the elliptic pde at
c           any grid point (x,y). the name chosen in the
calling routine
c           may be different where the coefficient routine
must be declared
c           "external."
c
c ... rhs
c
c           an array dimensioned nx by ny which contains
the given
c           right hand side values on the uniform 2-d
mesh.

c
c           rhs(i,j) = r(xi,yj) for i=1,...,nx and
j=1,...,ny
c
c ... phi
c
c           an array dimensioned nx by ny. on input phi
must contain
c           specified boundary values. for example, if
nyd=iparm(5)=1
c           then phi(i,ny) must be set equal to p(xi,yd)
for i=1,...nx

```

```
c           prior to calling cuh2.  these values are
preserved by cuh2.
c           if an initial guess is provided
(iguess=iparm(11)=1) it must
c           be input thru phi.
c
c
c ***      if no initial guess is given (iguess=0) then
phi must still
c           be initialized at all grid points (this is
not checked).  these
c           values will serve as an initial guess to the
pde at the coarsest
c           grid level after a transfer from the fine
solution grid.  set phi
c           equal to to 0.0 at all internal and non-
specified boundaries
c           grid points if nothing better is available.
c
c
c ... mgopt
c
c           an integer vector of length 4 which allows
the user to select
c           among various multigrid options.  if
mgopt(1)=0 is input then
c           a default set of multigrid parameters
(chosen for robustness)
c           will be internally selected and the
remaining values in mgopt
c           will be ignored.  if mgopt(1) is nonzero
then the parameters
c           in mgopt are set internally and defined as
follows: (see the
c           basic coarse grid correction algorithm
below)
c
c
c       kcycle = mgopt(1)
c
c           = 0 if default multigrid options are to be
used
c
c           = 1 if v cycling is to be used (the least
```

```
expensive per cycle)
c
c           = 2 if w cycling is to be used (the
default)
c
c           > 2 if more general k cycling is to be used
c           *** warning--values larger than 2 increase
c           the execution time per cycle
considerably and
c           result in the nonfatal error ierror =
-5
c           which indicates inefficient multigrid
cycling.
c
c     iprер = mgopt(2)
c
c           the number of "pre-relaxation" sweeps
executed before the
c           residual is restricted and cycling is
invoked at the next
c           coarser grid level (default value is 2
whenever mgopt(1)=0)
c
c     ipost = mgopt(3)
c
c           the number of "post relaxation" sweeps
executed after cycling
c           has been invoked at the next coarser grid
level and the residual
c           correction has been transferred back
(default value is 1
c           whenever mgopt(1)=0).
c
c *** if iprер, ipost, or (especially) kcycle is greater
than 2
c           than inefficient multigrid cycling has probably
been chosen and
c           the nonfatal error (see below) ierror = -5 will be
set. note
c           this warning may be overridden by any other
nonzero value
c           for ierror.
c
c     interpol = mgopt(4)
```

```
c
c           = 1 if multilinear prolongation
(interpolation) is used to
c           transfer residual corrections and the pde
approximation
c           from coarse to fine grids within full
multigrid cycling.

c
c           = 3 if multicubic prolongation
(interpolation) is used to
c           transfer residual corrections and the pde
approximation
c           from coarse to fine grids within full
multigrid cycling.

c           (this is the default value whenever
mgopt(1)=0).

c
c *** the default values (2,2,1,3) in the vector mgopt
were chosen for
c   robustness. in some cases v(2,1) cycles with
linear prolongation will
c   give good results with less computation
(especially in two-dimensions).
c   this was the default and only choice in an
earlier version of mudpack
c   (see [1]) and can be set with the integer vector
(1,2,1,1) in mgopt.

c
c *** the schedules for one full multigrid cycle
(iguess=0) using v(2,1)
c   cycles and w(2,1) cycles are depicted for a four
level grid below.
c   the number of relaxation sweeps when each grid is
visited are indicated.
c   the "*" stands for prolongation of the full
approximation and the "."
c   stands for transfer of residuals and residual
corrections within the
c   coarse grid correction algorithm (see below). all
version 5.0.1
c   mudpack solvers use only fully weighted residual
restriction. The
c   "D" at grid level 1 indicates a direct method is
used.
```

```

C
C      one fmg with v(2,1) cycles:
C
C
C      -----
C      -----2-----1-
C      level 4
C          *
C          .
C          *
C          .
C      -----2-----1-----2-----1-----
C      level 3
C          *
C          .
C          *
C          .
C      -----2-----1-----2-----1-----2-----1-----
C      level 2
C          *
C          .
C          *
C          .
C      ---D---D-----D-----D-----
C      level 1
C
C
C
C      one fmg with w(2,1) cycles:
C
C
C      -----
C      -----2-----
C      --1-- level 4
C          *
C          .
C          .
C      -----2-----1-----2-----3-----
C      1---- level 3
C          *
C          .
C          .
C          .
C          .
C      -----2-----1-----2-----3-----1-----
C      -----2-----1-----2-----3-----1-----2-----3-----1-----
C      level 2
C          *
C          .
C          .
C          .
C          .
C          .
C          .
C          .
C          .
C          .
C          .
C          .
C      ---D---D-----D-----D-----D-----D-----D-----
C      level 1
C
C
C      the form of the "recursive" coarse grid correction
C      cycling used
C      when kcycle.ge.0 is input is described below in
C      pseudo-algorithmic
C      language. it is implemented non-recursively in
C      fortran in mudpack.
C *** this algorithm is modified with the hybrid

```

```

solvers which use
c      a direct method whenever grid level 1 is
encountered.
c
c      algorithm
cgc(k,l(k),u(k),r(k),kcycle,iprer,ipost,ireshape,intpol)
c
c *** approximately solve l(k)*u(k) = r(k) using
multigrid iteration
c *** k is the current grid level
c *** l(k) is the discretized pde operator at level k
c *** u(k) is the initial guess at level k
c *** r(k) is the right hand side at level k
c *** i(k,k-1) is the restriction operator from level k
to level k-1
c *** (the form of i(k,k-1) depends on ireshape)
c *** i(k-1,k) is the prolongation operator from level
k-1 to level k
c *** (the form of i(k-1,k) depends on intpol)
c
c      begin algorithm cgc
c
c ***     pre-relax at level k
c
c      . do (i=1,iprer)
c
c      . . relax(l(k),u(k),r(k))
c
c      . end do
c
c      . if (k > 1) then
c
c ***      restrict the residual from level k to level k-
1
c
c      . . r(k-1) = i(k,k-1) (r(k)-l(k)*u(k))
c
c      . . kount = 0
c
c      . . repeat
c
c ***      solve for the residual correction at level k-1
in u(k-1)
c ***      using algorithm cgc "kcycle" times (this is

```

```

the recursion)
C
C      . . . kount = kount+1
C
C      . . . invoke cgc(k-1,l(k-1),u(k-1),r(k-
1),kcycle,iprer,ipost,iresw)
C
C
C      . . until (kount.eq.kcycle)
C
C ***      transfer residual correction in u(k-1) to
level k
C ***      with the prolongation operator and add to u(k)
C
C      . . u(k) = u(k) + i(k-1,k) (u(k-1))
C
C      . end if
C
C ***      post relax at level k
C
C      . do (i=1,ipost)
C
C      . . relax(l(k),u(k),r(k))
C
C      . end do
C
C      . return
C
C      end algorithm cgc
C
C
C
*****
*****  

C *** output arguments
*****  

C
*****  

C
C ... iparm(16) *** set for intl=0 calls only
C
C          on output iparm(16) contains the actual work

```

```

space length
c           required. this will usually be less than
that given by the
c           simplified formula for length=iparm(15) (see
as input argument)
c
c
c ... iparm(17) *** set for intl=1 calls only
c
c           on output iparm(17) contains the actual
number of multigrid cycles
c           between the finest and coarsest grid levels
used to obtain the
c           approximation when error control (tolmax >
0.0) is set.
c
c
c ... fparm(6) *** set for intl=1 calls with fparm(5)
> 0. only
c
c           on output fparm(6) contains the final
computed maximum relative
c           difference between the last two iterates at
the finest grid level.
c           fparm(6) is computed only if there is error
control (tolmax > 0.0)
c           assume phi1(i,j,k) and phi2(i,j,k) are the
last two computed
c           values for phi(i,j,k) at all points of the
finest grid level.
c           if we define
c
c           phdif = max(cabs(phi2(i,j)-phi1(i,j)))
over all i,j
c
c           and
c
c           phmax = max(cabs(phi2(i,j))) over all i,j
c
c           then
c
c           fparm(6) = phdif/phmax
c
c           is returned whenever phmax > 0.0. in the

```

```
degenerate case
c           phmax = 0.0, fparm(6) = phdif is returned.
c
c
c ... work
c
c           on output work contains intermediate values
that must not
c           be destroyed if cuh2 is to be called again
with intl=1
c
c
c ... phi    *** for intl=1 calls only
c
c           on output phi(i,j) contains the approximation
to p(xi,yj)
c           for all mesh points i = 1,...,nx and
j=1,...,ny. the last
c           computed iterate in phi is returned even if
convergence is
c           not obtained
c
c
c ... ierror
c
c           for intl=iparm(1)=0 initialization calls,
ierror is an
c           error flag that indicates invalid input
arguments when
c           returned positive and nonfatal warnings when
returned
c           negative. discretization is bypassed for
intl=1 calls
c           which can only return ierror = -1 or 0 or 1.
c
c
c     non-fatal warnings * * *
c
c
c     ==5 if kcycle=mgopt(1) is greater than 2. values
larger than 2 results
c           in an algorithm which probably does far more
computation than
c           necessary. kcycle = 1 (v cycles) or kcycle=2
```

```

(w cycles) should
c           suffice for most problems. ierror = -5 is
also set if either
c           iprер = mgopt(2) or ipost=mgopt(3) is greater
than 2. the
c           ierror=-5 flag is overridden by any other
fatal or non-fatal
c           error.

c
c     =-4 if there are dominant nonzero first order
terms in the pde which
c           make it "hyperbolic" at the finest grid level.
numerically, this
c           happens if:
c
c           cabs(cx)*dlx > 2.*cabs(cxx)      (dlx = (xb-
xa) / (nx-1))
c
c                           (or)
c
c           cabs(cy)*dly > 2.*cabs(cyy)      (dly = (yd-
yc) / (ny-1))
c
c
c           at some fine grid point (xi,yj). if an
adjustment is not made the
c           condition can lead to a matrix coming from the
discretization
c           which is not diagonally dominant and
divergence is possible. since
c           the condition is "likely" at coarser grid
levels for pde's with
c           nonzero first order terms, the adjustments
(actually first order
c           approximations to the pde)

c
c
c           cxx = cmplx(0.5*cabs(cx)*dx,0.0)
c
c           cyy = cmplx(0.5*cabs(cy)*dy,0.0)
c
c
c           (here dx,dy are the x,y mesh sizes of the
subgrid)

```

```

c
c      are made when necessary to preserve
convergence. if made
c      at the finest grid level, it can lead to
convergence to an
c      erroneous solution (flagged by ierror = -4).
a possible remedy
c      is to increase resolution. the ierror = -4
flag overrides the
c      nonfatal ierror = -5 flag.

c
c
c      ==2  if the pde is not elliptic
c
c      real(cxx)*real(cyy).le.0.0 or
c      aimag(cxx)*aimag(cyy).le.0.0
c
c      in this case a solution is still attempted
although convergence
c      may not occur due to ill-conditioning of the
linear system.
c      the ierror = -2 flag overrides the ierror=-
5,-4 nonfatal
c      flags.

c
c
c      ==1  if convergence to the tolerance specified in
tolmax=fparm(5)>0.
c      is not obtained in maxcy=iparm(13) multigrid
cycles between the
c      coarsest (ixp+1,jyq+1) and finest (nx,ny)
grid levels.
c      in this case the last computed iterate is
still returned.
c      the ierror = -1 flag overrides all other
nonfatal flags

c
c
c      no errors * * *
c
c      = 0
c
c      fatal argument errors * * *
c

```

```

c      = 1 if intl=iparm(1) is not 0 on initial call or
not 0 or 1
c          on subsequent calls
c
c      = 2 if any of the boundary condition flags
nxa,nxb,nyc,nyd
c          in iparm(2),iparm(3),iparm(4),iparm(5) are not
0,1 or 2
c          or if nxa,nxb or nyc,nyd are not pairwise
zero.
c
c      = 3 if min0(ixp,jyq) < 2 (ixp = iparm(6), jyq =
iparm(7))
c          of if ixp < 3 when nxa=0 or if jyq < 3 when
nyc=0.
c
c      = 4 if min0(iex,jey) < 1 (iex = iparm(8), jey =
iparm(9)) or
c          if max0(iex,jey) > 50
c
c      = 5 if nx.ne.ixp*2** (iex-1)+1 or
ny.ne.jyq*2** (jey-1)+1
c          (nx = iparm(10), ny = iparm(11))
c
c      = 6 if iguess = iparm(12) is not equal to 0 or 1
c
c      = 7 if maxcy = iparm(13) < 1
c
c      = 8 if method = iparm(14) is not 0,1,2, or 3
c
c      = 9 if length = iparm(15) is too small (see
iparm(16) on output
c          for minimum required work space length)
c
c      =10 if xa >= xb or yc >= yd
c
(xa=fparm(1),xb=fparm(2),yc=fparm(3),yd=fparm(4))
c
c      =11 if tolmax = fparm(5) < 0.0
c
c      errors in setting multigrid options * * * (see
also ierror=-5)
c
c      =12 if kcyle = mgopt(1) < 0 or

```

```
C           if iprer = mgopt(1) < 1 or
C           if ipost = mgopt(3) < 1 or
C           if interpol = mgopt(4) is not 1 or 3
C
C           =13 if iex=jey=1 (full direct method) and iguess=1
C or maxcy > 1
C
C           =14 if the elliptic pde is singular (see ierror=-3
C in cud2.d)
C
C
C ****
*
C
C ****
*
C
C ****
*
C
C ****
*
C
C ****
**
C
C ****
**
C
C ****
*
C
C ****
```

---

## CUH24

```
C
C     file cuh24.d
C
C     * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C     *
C     *
C     *               copyright (c) 2008 by UCAR
```

\*  
C \*  
\*  
C \* University Corporation for Atmospheric  
Research \*  
C \*  
\*  
C \* all rights reserved  
\*  
C \*  
\*  
C \* MUDPACK version 5.0.1  
\*  
C \*  
\*  
C \* A Fortran Package of Multigrid  
\*  
C \*  
\*  
C \* Subroutines and Example Programs  
\*  
C \*  
\*  
C \* for Solving Elliptic Partial Differential  
Equations \*  
C \*  
\*  
C \* by  
\*  
C \*  
\*  
C \* John Adams  
\*  
C \*  
\*  
C \* of  
\*  
C \*  
\*  
C \* the National Center for Atmospheric  
Research \*  
C \*  
\*  
C \* Boulder, Colorado (80307)

```
U.S.A.          *
C      *
*
C      *           which is sponsored by
*
C      *
*
C      *           the National Science Foundation
*
C      *
*
C      *           ****
* * * * * * *
C
C ... file cuh24.d
C
C     contains documentation for:
C     subroutine cuh24(wk,iwk,phi,ierror)
C     A sample fortran driver is file "tcuh24.f".
C
C ... required MUDPACK files
C
C     cuh2.f, cudcom.f
C
C ... purpose
C
C     cuh24 attempts to improve the estimate in phi,
obtained by calling
C     cuh2, from second to fourth order accuracy. see
the file "cuh2.d"
C     for a detailed discussion of the elliptic pde
approximated and
C     arguments "wk,iwk,phi" which are also part of the
argument list for
C     cuh2.
C
C ... assumptions
C
C     * phi contains a second-order approximation from
an earlier cuh2 call
C
C     * arguments "wk,iwk,phi" are the same used in
calling cuh2
C
```

```
c      * "wk,iwk,phi" have not changed since the last
call to cuh2
c
c      * the finest grid level contains at least 6
points in each direction
c
c
c *** warning
c
c      if the first assumption is not true then a fourth
order approximation
c      cannot be produced in phi. the last assumption
(adequate grid size)
c      is the only one checked. failure in any of the
others can result in
c      in an undetectable error.
c
c ... language
c
c      fortran90/fortran77
c
c ... portability
c
c      mudpack5.0.1 software has been compiled and tested
with fortran77
c      and fortran90 on a variety of platforms.
c
c ... methods
c
c      details of the methods employed by the solvers in
mudpack are given
c      in [1,9]. [1,2,9] contain performance
measurements on a variety of
c      elliptic pdes (see "references" in the file
"readme"). in summary:
c
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
estimate (i.e.,
c      discretization level error is reached) then this
can be improved to a
```

```
c      fourth-order estimate using the technique of
"deferred corrections"
c      the values in the solution array are used to
generate a fourth-order
c      approximation to the truncation error. second-
order finite difference
c      formula are used to approximate third and fourth
partial derivatives
c      of the solution function [3]. the truncation
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side. the default
multigrid options
c      are used to compute the fourth-order correction
term which is added
c      to the original solution array.
c
c
c ... references (partial)
c
c
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
c      Solution of Linear Elliptic Partial Differential
Equations,"
c      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
c
c      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
c      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
c      1989, pp.1-12.
c      .
c      .
c      .
c      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
Haidvogel, and V. Pizzo,
c      "Applications of Multigrid Software in the
Atmospheric Sciences,"
c      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-
1458.
c      .
```

```
C .
C .
C      [9] J. Adams, "Recent Enhancements in MUDPACK, a
Multigrid Software
C      package for Elliptic Partial Differential
Equations," Applied Math.
C      and Comp., 1991, vol. 43, May 1991, pp. 79-94.
C
C      [10]J. Adams, "MUDPACK-2: Multigrid Software for
Approximating
C      Elliptic Partial Differential Equations on Uniform
Grids with
C      any Resolution," Applied Math. and Comp., 1993,
vol. 53, February
C      1993, pp. 235-249
C .
C .
C .
C
C ... error parameter
C
C      = 0 if no error is detected
C
C      = 30 if min0(nx,ny) < 6 where nx,ny are the fine
grid sizes
C          in the x,y directions.
C
C
C
*****
*****
```

\*\*\*\*\*

```
C
*****
*****
```

\*\*\*\*\*

```
C
*****
*****
```

\*\*\*\*\*

```
C
C      end of cuh24 documentation
C
C
*****
*****
```

\*\*\*\*\*

```
C
*****
*****
```

```
*****
```

```
C
```

---

## CUH24CR

```
C  
C      file cuh24cr.d  
C  
C      * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C      *  
*  
C      *          copyright (c) 2008 by UCAR  
*  
C      *  
*  
C      *          University Corporation for Atmospheric  
Research           *  
C      *  
*  
C      *          all rights reserved  
*  
C      *  
*  
C      *          MUDPACK version 5.0.1  
*  
C      *  
*  
C      *          A Fortran Package of Multigrid  
*  
C      *  
*  
C      *          Subroutines and Example Programs  
*  
C      *  
*  
C      *          for Solving Elliptic Partial Differential  
Equations       *
```

```
C      *
*
C      *                                by
*
C      *
*
C      *                                John Adams
*
C      *
*
C      *                                of
*
C      *
*
C      *                                the National Center for Atmospheric
Research          *
C      *
*
C      *                                Boulder, Colorado (80307)
U.S.A.          *
C      *
*
C      *                                which is sponsored by
*
C      *
*
C      *                                the National Science Foundation
*
C      *
*
C      *      * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C
C ... file cuh24cr.d
C
C     contains documentation for:
C     subroutine cuh24cr(wk,iwk,coef,bndyc,phi,ierror)
C     A sample fortran driver is file "tcuh24cr.f".
C
C ... required MUDPACK files
C
C     cuh2cr.f, cudcom.f
C
C ... purpose
```

```
c
c      cuh24cr attempts to improve the estimate in phi,
obtained by calling
c      cuh2cr, from second to fourth order accuracy.
see the file "cuh2cr.d"
c      for a detailed discussion of the elliptic pde
approximated and
c      arguments "wk,iwk,coef,bndyc,phi" which are also
part of the argument
c      list for cuh2cr.
c
c ... assumptions
c
c      * phi contains a second-order approximation from
an earlier cuh2cr call
c
c      * arguments "wk,iwk,coef,bndyc,phi" are the same
used in calling cuh2cr
c
c      * "wk,iwk,coef,bndyc,phi" have not changed since
the last call to cuh2cr
c
c      * the finest grid level contains at least 6
points in each direction
c
c
c *** warning
c
c      if the first assumption is not true then a fourth
order approximation
c      cannot be produced in phi. the last assumption
(adequate grid size)
c      is the only one checked. failure in any of the
others can result in
c      in an undetectable error.
c
c ... language
c
c      fortran90/fortran77
c
c ... portability
c
c      mudpack5.0.1 software has been compiled and tested
with fortran77
```

```
c      and fortran90 on a variety of platforms.  
c  
c ... methods  
c  
c      details of the methods employed by the solvers in  
mudpack are given  
c      in [1,9]. [1,2,9] contain performance  
measurements on a variety of  
c      elliptic pdes (see "references" in the file  
"readme"). in summary:  
c  
c  
c *** higher order solution (fourth-order solvers) (see  
[9,19,21])  
c  
c      if the multigrid cycling results in a second-order  
estimate (i.e.,  
c      discretization level error is reached) then this  
can be improved to a  
c      fourth-order estimate using the technique of  
"deferred corrections"  
c      the values in the solution array are used to  
generate a fourth-order  
c      approximation to the truncation error. second-  
order finite difference  
c      formula are used to approximate third and fourth  
partial derivatives  
c      of the solution function [3]. the truncation  
error estimate is  
c      transferred down to all grid levels using weighted  
averaging where  
c      it serves as a new right hand side. the default  
multigrid options  
c      are used to compute the fourth-order correction  
term which is added  
c      to the original solution array.  
c  
c  
c ... references (partial)  
c  
c  
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software  
for the Efficient  
c      Solution of Linear Elliptic Partial Differential
```

Equations,"

c        Applied Math. and Comput. vol.34, Nov 1989,  
pp.113-146.

c

c        [2] J. Adams, "FMG Results with the Multigrid  
Software Package MUDPACK,"

c        proceedings of the fourth Copper Mountain  
Conference on Multigrid, SIAM,

c        1989, pp.1-12.

c        .

c        .

c        .

c        [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.  
Haidvogel, and V. Pizzo,

c        "Applications of Multigrid Software in the  
Atmospheric Sciences,"

c        Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-  
1458.

c        .

c        .

c        .

c        [9] J. Adams, "Recent Enhancements in MUDPACK, a  
Multigrid Software

c        package for Elliptic Partial Differential  
Equations," Applied Math.

c        and Comp., 1991, vol. 43, May 1991, pp. 79-94.

c

c        [10]J. Adams, "MUDPACK-2: Multigrid Software for  
Approximating

c        Elliptic Partial Differential Equations on Uniform  
Grids with

c        any Resolution," Applied Math. and Comp., 1993,  
vol. 53, February

c        1993, pp. 235-249

c        .

c        .

c        .

c

c ... error parameter

c

c        = 0 if no error is detected

c

c        = 30 if min0(nx,ny) < 6 where nx,ny are the fine  
grid sizes

```
C      in the x,y directions.  
C  
C  
C  
*****  
*****  
C  
*****  
*****  
C  
*****  
*****  
C  
      end of cuh24cr documentation  
C  
C  
*****  
*****  
C  
*****  
*****  
C
```

---

## CUH2CR

```
C  
C      file cuh2cr.d  
C  
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * * *  
C      *  
*  
C      *          copyright (c) 2008 by UCAR  
*  
C      *  
*  
C      *          University Corporation for Atmospheric  
Research           *  
C      *  
*  
C      *          all rights reserved
```

\*  
C \*  
\*  
C \* MUDPACK version 5.0.1  
\*  
C \*  
\*  
C \* A Fortran Package of Multigrid  
\*  
C \*  
\*  
C \* Subroutines and Example Programs  
\*  
C \*  
\*  
C \* for Solving Elliptic Partial Differential  
Equations \*  
C \*  
\*  
C \* by  
\*  
C \*  
\*  
C \* John Adams  
\*  
C \*  
\*  
C \* of  
\*  
C \*  
\*  
C \* the National Center for Atmospheric  
Research \*  
C \*  
\*  
C \* Boulder, Colorado (80307)  
U.S.A. \*  
C \*  
\*  
C \* which is sponsored by  
\*  
C \*  
\*  
C \* the National Science Foundation

```
*  
C      *  
*  
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C  
c ... file cuh2cr.d  
C  
c      contains documentation for the complex mudpack  
solver:  
c      subroutine  
cuh2cr(iparm,fparm,work,iw,coef,bndyc,rhs,phi,mgopt,ierr  
or)  
c      a sample fortran driver is file "tcuh2cr.f".  
C  
c ... required mudpack files  
C  
c      cudcom.f  
C  
c ... purpose  
C  
c      the "hybrid" multigrid/direct method code cuh2cr  
approximates the  
c      same 2-d nonseparable elliptic pde as the mudpack  
solver cud2cr.  
c      cuh2cr combines the efficiency of multigrid  
iteration with the certainty  
c      a direct method. the basic algorithm is modified  
by using banded  
c      gaussian elimination in place of relaxation  
whenever the coarsest  
c      subgrid is encountered within multigrid cycling.  
this provides  
c      additional grid size flexibility by eliminating  
the usual multigrid  
c      constraint that the coarsest grid consist of "few"  
points for effective  
c      error reduction with multigrid cycling. In many  
cases the hybrid method  
c      provides more robust convergence characteristics  
than multigrid cycling  
c      alone.  
  
c      The pde approximated is:
```

```

C
C
C           cxx(x,y)*pxx + cxy(x,y)*pxy + cyy(x,y)*pyy +
cx(x,y)*px +
C
C           cy(x,y)*py + ce(x,y)*p(x,y) = r(x,y) .
C
C
C           pxx,pxy,pyy,px,py are second and first partial
derivatives of the
C           unknown complex solution function p(x,y) with
respect to the
C           independent variables x,y.   cxx,cxy,cyy,cx,cy,ce
are the known
C           complex coefficients of the elliptic pde and
r(x,y) is the known
C           complex right hand side of the equation.  The real
parts of cxx,cyy
C           or the imaginary parts of cxx,cyy should be
positive for all x,y
C           in the solution region (see ierror==2) .
Nonseparability means some
C           of the coefficients depend on both x and y.  if
the PDE is separable
C           and cxy = 0 then subroutine cud2sp should be used.
C
C *** cuh2cr becomes a full direct method if grid size
arguments are chosen
C           so that the coarsest and finest grids coincide.
choosing iex=jey=1
C           and ixp=nx-1, jyq=ny-1
(iex=iparm(6),jey=iparm(7),ixp=iparm(8),
C           jyq=iparm(9),nx=iparm(10),ny=iparm(11)) will set
gaussian elimination
C           on the nx by ny grid.
C
C ... argument differences with cud2cr.f
C
C           the input and output arguments of cuh2cr are
almost identical to the
C           arguments of cud2cr (see cud2cr.d) with the
following exceptions:
C
C           (1) the complex work space vector "wk" requires

```

```

c
c           (ixp+1)*(jyq+1)*(2*ixp+3)
c
c           additional words of storage (ixp = iparm(6),
jyq = iparm(7))
c           if periodic boundary conditions are not
flagged in the y direction
c           (nyc .ne. 0 where nyc = iparm(4)) or
c
c           (ixp+1)*[2*(ixp+1)*(2*jyq-1)+jyq+1]
c
c           additional words of storage if periodic
boundary conditions are
c           flagged in the y direction (nyc = 0).  the
extra work space is
c           used for a direct solution with gaussian
elimination whenever the
c           coarsest grid is encountered within multigrid
cycling.
c
c           (2) An integer work space iwk of length at least
(ixp+1)*(jyq+1)
c           must be provided.
c
c           (3) jyq must be greater than 2 if periodic
boundary conditions
c           are flagged in the y direction and ixp must be
greater than
c           2 if periodic boundary conditions are flagged
in the x direction.
c           inputting jyq = 2 when nyc = 0 or inputting
ixp = 2 when nxa = 0
c           will set the fatal error flag ierror=3
c
c *** (4) it is no longer necessary that ixp and jyq be
"small" for
c           effective error reduction with multigrid
iteration.  there
c           is no reduction in convergence rates when
larger values for
c           ixp or jyq are used .  this provides
additional flexibility
c           in choosing grid size.  in many cases cuh2
provides more

```

```
c      robust convergence than cud2. it can be used
in place of
c      cud2 for all nonsingular problems (see (5)
below).
c
c      (5) iguess = iparm(11) = 1 (flagging an initial
guess) or
c      maxcy = iparm(14) > 1 (setting more than one
multigrid
c      cycle) are not allowed if cuh2 becomes a full
direct method
c      by choosing iex = jey = 1 (iex = iparm(8),jey
= iparm(9)).
c      this conflicting combination of input
arguments for multigrid
c      iteration and a full direct method set the
fatal error flag
c
c      ierror = 13
c
c      iguess = 0 and maxcy = 1 are required when
cuh2 becomes a
c      full direct method.
c
c      (6) if a "singular" pde is detected (see ierror=-3
description in cud2.d;
c      ce(x,y) = 0.0 for all x,y and the boundary
conditions are a combination
c      of periodic and/or pure derivatives) then cuh2
sets the fatal error
c      flag
c
c      ierror = 14
c
c      The direct method utilized by cuh2 would
likely cause a division
c      by zero in the singular case. cud2 can be
tried for singular problems
c
c
c ... grid size considerations
c
c      (1) flexibility
```

```
c cuh2 should be used in place of cud2 whenever
grid size
c requirements do not allow choosing ixp and jyq
to be "small"
c positive integers (typically less than 4).
c
c example:
c
c suppose we wish to solve an elliptic pde on a
one degree grid on
c the full surface of a sphere. choosing ixp =
jyq = 45 and iex = 4
c and jyq = 3 fits the required 361 by 181 grid
exactly. multigrid
c cycling will be used on the sequence of
subgrid sizes:
c
c 46 x 46 < 91 x 46 < 181 x 91 < 361 x
181
c
c the 46 x 46 coarsest subgrid has too much
resolution for effective
c error reduction with relaxation only. cuh2
circumvents this
c difficulty by generating an exact direct
solution (modulo roundoff
c error) whenever the coarsest grid is
encountered.
c
c (2) additional work space (see (1) under
"arguments differences") is
c required by cuh2 to implement gaussian
elimination at the coarsest
c grid level. this may limit the size of ixp
and jyq.
c
c (3) operation counts
c
c for simplicity, assume p = ixp = jyq and n =
nx = ny. banded
c gaussian elimination requires  $O(p^{**}4)$ 
operations for solution
c on the coarsest subgrid while multigrid
iteration is a  $O(n^{**}2)$ 
```

```

c      algorithm. these are approximately balanced
when
c
c      p**4 =: (n/ (2**k) )**4 =: n**2
c
c      or
c
c      k =: log2(n) /2
c
c      grid levels are chosen with the hybrid method.
so if
c      p is approximately equal to
c
c      n/ (2** (log2(n) /2) )
c
c      then the direct method and multigrid parts of
the hybrid algorithm
c      require roughly the same amount of computer
time. larger values
c      for p mean the direct method will dominate the
computation. smaller
c      values mean the hybrid method will cost only
marginally more than
c      multigrid iteration with coarse grid
relaxation.
c
c
c *** the remaining documentation is almost identical to
cud2.d
c      except for the modifications already indicated.
c
c ... mesh description . . .
c
c      the approximation is generated on a uniform nx by
ny grid. the grid
c      is superimposed on the rectangular solution region
c
c      [xa,xb] x [yc,yd].
c
c      let
c
c      dlx = (xb-xa) / (nx-1), dly = (yd-yc) / (ny-1)
c
c      be the uniform grid increments in the x,y

```

```
directions. then
c
c           xi=xa+(i-1)*dlx,   yj=yc+(j-1)*dly
c
c       for i=1,...,nx and j=1,...,ny  denote the x,y
uniform mesh points
c
c
c ... language
c
c       fortran90/fortran77
c
c
c ... portability
c
c       mudpack5.0.1 software has been compiled and tested
with fortran77
c       and fortran90 on a variety of platforms.
c
c ... methods
c
c       details of the methods employed by the solvers in
mudpack are given
c       in [1,9]. [1,2,9] contain performance
measurements on a variety of
c       elliptic pdes (see "references" in the file
"readme"). in summary:
c
c *** discretization and solution (second-order solvers)
(see [1])
c
c       the pde and boundary conditions are automatically
discretized at all
c       grid levels using second-order finite difference
formula. diagonal
c       dominance at coarser grid levels is maintained in
the presence of
c       nonzero first-order terms by adjusting the second-
order coefficient
c       when necessary. the resulting block tri-diagonal
linear system is
c       approximated using multigrid iteration
[10,11,13,15,16,18]. version
c       5.0.1 of mudpack uses only fully weighted residual
```

```
restriction. defaults
c      include cubic prolongation and w(2,1) cycles.
these can be overridden
c      with selected multigrid options (see "mgopt").
error control based on
c      maximum relative differences is available. full
multigrid cycling (fmg)
c      or cycling beginning or restarting at the finest
grid level can be
c      selected. a menu of relaxation methods including
gauss-seidel point,
c      line relaxation(s) (in any combination of
directions) and planar
c      relaxation (for three-dimensional anisotropic
problems) are provided.
c      all methods use ordering based on alternating
points (red/black),
c      lines, or planes for cray vectorization and
improved convergence
c      rates [14].
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
estimate (i.e.,
c      discretization level error is reached) then this
can be improved to a
c      fourth-order estimate using the technique of
"deferred corrections."
c      the values in the solution array are used to
generate a fourth-order
c      approximation to the truncation error. second-
order finite difference
c      formula are used to approximate third and fourth
partial derivatives
c      of the solution function [3]. the truncation
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side. the default
multigrid options
c      are used to compute the fourth-order correction
term which is added
```

```
c      to the original solution array.  
c  
c  
c ... references (partial)  
c  
c  
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software  
for the Efficient  
c      Solution of Linear Elliptic Partial Differential  
Equations,"  
c      Applied Math. and Comput. vol.34, Nov 1989,  
pp.113-146.  
c  
c      [2] J. Adams, "FMG Results with the Multigrid  
Software Package MUDPACK,"  
c      proceedings of the fourth Copper Mountain  
Conference on Multigrid, SIAM,  
c      1989, pp.1-12.  
c      .  
c      .  
c      .  
c      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.  
Haidvogel, and V. Pizzo,  
c      "Applications of Multigrid Software in the  
Atmospheric Sciences,"  
c      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-  
1458.  
c      .  
c      .  
c      .  
c      [9] J. Adams, "Recent Enhancements in MUDPACK, a  
Multigrid Software  
c      package for Elliptic Partial Differential  
Equations," Applied Math.  
c      and Comp., 1991, vol. 43, May 1991, pp. 79-94.  
c  
c      [10]J. Adams, "MUDPACK-2: Multigrid Software for  
Approximating  
c      Elliptic Partial Differential Equations on Uniform  
Grids with  
c      any Resolution," Applied Math. and Comp., 1993,  
vol. 53, February  
c      1993, pp. 235-249  
c      .
```

```
C .
C .
C
C ... argument description
C
C
C
*****
C *** input arguments
*****
C
*****
C ... iparm
C
C      an integer vector of length 17 used to pass
integer
C      arguments. iparm is set internally and
defined as
C      follows:
C
C
C ... intl=iparm(1)
C
C      an initialization argument. intl=0 must be
input
C      on an initial call. in this case input
arguments will
C      be checked for errors and the elliptic
partial differential
C      equation and boundary conditions will be
discretized using
C      second order finite difference formula.
C
C ***      an approximation is not generated after an
intl=0 call!
C      cuh2cr should be called with intl=1 to
approximate the elliptic
C      pde discretized by the intl=0 call. intl=1
should also
C      be input if cuh2cr has been called earlier
```

```
and only the
c           values in in rhs (see below) or gbdy (see
bndyc below)
c           or phi (see below) have changed.  this will
bypass
c           redundant pde discretization and argument
checking
c           and save computational time.  some examples
of when
c           intl=1 calls should be used are:
c
c           (0) after a intl=0 argument checking and
discretization call
c
c           (1) cuh2cr is being recalled for additional
accuracy.  in
c           this case iguess=iparm(12)=1 should also
be used.
c
c           (2) cuh2cr is being called every time step in
a time dependent
c           problem (see discussion below) where the
elliptic operator
c           does not depend on time.
c
c           (3) cuh2cr is being used to solve the same
elliptic equation
c           for several different right hand sides
(igueess=0 should
c           probably be used for each new righthand
side) .
c
c           intl = 0 must be input before calling with
intl = 1 when any
c           of the following conditions hold:
c
c           (a) the initial call to cuh2cr
c
c           (b) any of the integer arguments other than
igueess=iparm(12)
c           or maxcy=iparm(13) or mgopt have changed
since the previous
c           call.
c
```

```
c           (c) any of the floating point arguments other
than tolmax=
c           fparm(5) have changed since the previous
call
c
c           (d) any of the coefficients input by coef
(see below) have
c           changed since the previous call
c
c           (e) any of the "alfa" coefficients input by
bndyc (see below)
c           have changed since the previous call.
c
c           if any of (a) through (e) are true then the
elliptic pde
c           must be discretized or rediscretized. if
none of (a)
c           through (e) holds, calls can be made with
intl=1.
c           incorrect calls with intl=1 will produce
erroneous results.
c ***      the values set in the saved work space "work"
(see below) with
c           an intl=0 call must be preserved with
subsequent intl=1 calls.
c
c           MUDPACK software performance should be
monitored for intl=1
c           calls. The intl=0 discretization call
performance depends
c           primarily on the efficiency or lack of
efficiency of the
c           user provided subroutines for pde
coefficients and
c           boundary conditions.
c
c
c ... nxa=iparm(2)
c
c           flags boundary conditions on the edge x=xa
c
c           = 0 if p(x,y) is periodic in x on [xa,xb]
c           (i.e., p(x+xb-xa,y) = p(x,y) for all x,y
c           (if nxa=0 then nxb=0 is required, see
```

```

ierror = 2)
c
c      = 1 if p(xa,y) is specified (this must be input
thru phi(1,j))
c
c      = 2 if there are mixed derivative boundary
conditions at x=xa
c          (see bndyc)
c
c
c ... nxb=iparm(3)
c
c      flags boundary conditions on the edge x=xb
c
c      = 0 if p(x,y) is periodic in x on [xa,xb]
c          (i.e., p(x+xb-xa,y) = p(x,y) for all x,y)
c          (if nxb=0 then nxa=0 is required, see
ierror = 2)
c
c      = 1 if p(xb,y) is specified (this must be input
thru phi(nx,j))
c
c      = 2 if there are mixed derivative boundary
conditions at x=xb
c          (see bndyc)
c
c
c ... nyc=iparm(4)
c
c      flags boundary conditions on the edge y=yc
c
c      = 0 if p(x,y) is periodic in y on [yc,yd]
c          (i.e., p(x,y+yd-yc) = p(x,y) for all x,y)
c          (if nyc=0 then nyd=0 is required, see
ierror = 2)
c
c      = 1 if p(x,yc) is specified (this must be input
thru phi(i,1))
c
c      = 2 if there are mixed derivative boundary
conditions at y=yc
c          (see bndyc)
c
c

```

```

c ... nyd=iparm(5)
c
c           flags boundary conditions on the edge y=yd
c
c           = 0 if p(x,y) is periodic in y on [yc,yd]
c           (i.e., p(x,y+yd-yc) = p(x,y) for all x,y
c           (if nyd=0 then nyc=0 is required, see
ierror = 2)
c
c           = 1 if p(x,yd) is specified (this must be input
thru phi(i,ny))
c
c           = 2 if there are mixed derivative boundary
conditions at y=yd
c           (see bndyc)
c
c
c *** grid size arguments
c
c
c ... ixp = iparm(6)
c
c           an integer greater than one which is used in
defining the number
c           of grid points in the x direction (see nx =
iparm(10)). "ixp+1"
c           is the number of points on the coarsest x
grid visited during
c           multigrid cycling. ixp should be chosen as
small as possible.
c           recommended values are the small primes 2 or
3.
c           larger values can reduce multigrid
convergence rates considerably,
c           especially if line relaxation in the x
direction is not used.
c           if ixp > 2 then it should be 2 or a small odd
value since a power
c           of 2 factor of ixp can be removed by
increasing iex = iparm(8)
c           without changing nx = iparm(10).
c
c
c ... jyq = iparm(7)

```

```
c
c      an integer greater than one which is used in
defining the number
c      of grid points in the y direction (see ny =
iparm(11)). "jyq+1"
c      is the number of points on the coarsest y
grid visited during
c      multigrid cycling. jyq should be chosen as
small as possible.
c      recommended values are the small primes 2 or
3.
c      larger values can reduce multigrid
convergence rates considerably,
c      especially if line relaxation in the y
direction is not used.
c      if jyq > 2 then it should be 2 or a small odd
value since a power
c      of 2 factor of jyq can be removed by
increasing jey = iparm(9)
c      without changing ny = iparm(11).
c
c
c ... iex = iparm(8)
c
c      a positive integer exponent of 2 used in
defining the number
c      of grid points in the x direction (see nx =
iparm(10)).
c      iex .le. 50 is required. for efficient
multigrid cycling,
c      iex should be chosen as large as possible and
ixp=iparm(8)
c      as small as possible within grid size
constraints when
c      defining nx.
c
c
c ... jey = iparm(9)
c
c      a positive integer exponent of 2 used in
defining the number
c      of grid points in the y direction (see ny =
iparm(11)).
c      jey .le. 50 is required. for efficient
```

```

multigrid cycling,
c           jey should be chosen as large as possible and
jyq=iparm(7)
c           as small as possible within grid size
constraints when
c           defining ny.
c
c
c
c ... nx = iparm(10)
c
c           the number of equally spaced grid points in
the interval [xa,xb]
c           (including the boundaries). nx must have the
form
c
c           nx = ixp*(2** (iex-1)) + 1
c
c           where ixp = iparm(6), iex = iparm(8).
c
c
c ... ny = iparm(11)
c
c           the number of equally spaced grid points in
the interval [yc,yd]
c           (including the boundaries). ny must have the
form:
c
c           ny = jyq*(2** (jey-1)) + 1
c
c           where jyq = iparm(7), jey = iparm(9).
c
c
c *** example
c
c           suppose a solution is wanted on a 33 by 97
grid. then
c           ixp=2, jyq=6 and iex=jey=5 could be used. a
better
c           choice would be ixp=2, jyq=3, and iex=5,
jey=6.
c
c *** note
c

```

```

c      let g be the nx by ny fine grid on which the
approximation is
c      generated and let n = max0(iex,jey).  in mudpack,
multigrid
c      cycling is implemented on the ascending chain of
grids
c
c          g(1) < ... < g(k) < ... < g(n) = g.
c
c      each g(k) (k=1,...,n) has mx(k) by my(k) grid
points
c      given by:
c
c          mx(k) = ixp*[2**max0(iex+k-n,1)-1] + 1
c
c          my(k) = jyq*[2**max0(jey+k-n,1)-1] + 1
c
c
c
c ... iguess=iparm(12)
c
c          = 0 if no initial guess to the pde is
provided
c
c          = 1 if an initial guess to the pde is at the
finest grid
c              level is provided in phi (see below)
c
c      comments on iguess = 0 or 1 . . .
c
c      even if iguess = 0, phi must be initialized at all
grid points (this
c      is not checked).  phi can be set to 0.0 at non-
dirchlet grid points
c      if nothing better is available.  the values set in
phi when iguess = 0
c      are passed down and serve as an initial guess to
the pde at the coarsest
c      grid level where cycling commences.  in this
sense, values input in
c      phi always serve as an initial guess.  setting
iguess = 0 forces full
c      multigrid cycling beginning at the coarsest and
finishing at the finest

```

```

c      grid level.
c
c      if iguess = 1 then the values input in phi are an
initial guess to the
c      pde at the finest grid level where cycling begins.
this option should
c      be used only if a "very good" initial guess is
available (as, for
c      example, when restarting from a previous iguess=0
call).
c
c      time dependent problems . . .
c
c *** assume we are solving an elliptic pde every time
step in a
c      marching problem of the form:
c
c          l(p(t)) = r(t)
c
c      where the differential operator "l" has no time
dependence,
c      "p(t)" is the solution and "r(t)" is the right
hand side at
c      current time "t". let "dt" be the increment
between time steps.
c      then p(t) can be used as an initial guess to
p(t+dt) with
c      intl = 1 when solving
c
c          l(p(t+dt)) = r(t+dt).
c
c      after the first two time steps, rather than
continue, it would
c      be better to define the "correction" term:
c
c          e(t,dt) = p(t+dt) - p(t)
c
c      this clearly satisfies the equation
c
c          l(e(t,dt)) = r(t+dt) - r(t).
c
c      this should be solved with iguess = 0 and intl =
1. boundary
c      conditions for e(t,dt) are obtained from the

```

```

boundary conditions
c      for p(t) by subtracting given values at t from
given values at
c      t+dt. for example if
c
c           $d(p(t))/dx = f(t)$ ,  $d(p(t+dt))/dx = f(t+dt)$ 
c
c      at some x boundary then e(t,dt) satisfies the
derivative
c      boundary condition
c
c           $d(e(t,dt))/dx = f(t+dt) - f(t)$ .
c
c      e(t,dt) can be preset to 0.0 (at nondirichlet
points) or (if p(t-dt)
c      is saved) to p(t)-p(t-dt). with iguess = 0, these
values will serve
c      as an initial guess to e(t,dt) at the coarsest
grid level. this
c      approach has the advantage that a full sequence of
multigrid cycles,
c      beginning at the coarsest grid level, is invoked
every time step in
c      solving for e(t,dt). a few digits of accuracy in
e(t,dt), which is
c      ordinarily much smaller than p(t), will yield
several more digits of
c      accuracy in the final approximation:
c
c           $p(t+dt) = p(t) + e(t,dt)$ .
c
c      using this approach to integrate in time will give
more accuracy
c      then using p(t) as an initial guess to p(t+dt) for
all time steps.
c      it does require additional storage.
c
c      if the differential operator "l" has time
dependence (either thru
c      the coefficients in the pde or the coefficients in
the derivative
c      boundary conditions) then use p(t) as an initial
guess to p(t+dt)
c      when solving

```

```

c
c           l(t+dt) (p(t+dt)) = r(t+dt)
c
c       with intl = 0 for all time steps (the
discretization must be repeated
c       for each new "t"). either iguess = 0 (p(t) will
then be an initial
c       guess at the coarsest grid level where cycles will
commence) or
c       iguess = 1 (p(t) will then be an initial guess at
the finest grid
c       level where cycles will remain fixed) can be
tried.
c
c
c ... maxcy = iparm(13)
c
c       the exact number of cycles executed between
the finest (nx by
c               ny) and the coarsest ((ixp+1) by (jyq+1))
grid levels when
c       tolmax=fparm(5)=0.0 (no error control). when
tolmax > 0.0
c       is input (error control) then maxcy is a
limit on the number
c       of cycles between the finest and coarsest
grid levels. in
c       any case, at most maxcy*(iprert+ipost)
relaxation sweeps are
c       are performed at the finest grid level (see
iprer=mgopt(2),
c               ipost=mgopt(3) below). when multigrid
iteration is working
c       "correctly" only a few are required for
convergence. large
c       values for maxcy should not be necessary.
c
c
c ... method = iparm(14) determines the method of
relaxation
c               (gauss-seidel based on alternating points
or lines)
c
c               = 0 for point relaxation

```

```

c
c      = 1 for line relaxation in the x direction
c
c      = 2 for line relaxation in the y direction
c
c      = 3 for line relaxation in both the x and y
direction
c
c
c *** choice of method. . .
c
c      let fx represent the quantity cxx(x,y)/dlx**2 over
the solution region.
c
c      let fy represent the quantity cyy(x,y)/dly**2 over
the solution region
c
c      if fx,fy are roughly the same size and do not vary
too much over
c      the solution region choose method = 0. if this
fails try method=3.
c
c      if fx is much greater than fy choose method = 1.
c
c      if fy is much greater than fx choose method = 2
c
c      if neither fx or fy dominates over the solution
region and they
c      both vary considerably choose method = 3.
c
c
c ... length = iparm(15)
c
c      the length of the work space provided in
complex work
c      space "work")
c
c      let isx = 0 if method = 0 or method = 2
c      let isx = 3 if method = 1 or method = 3 and
nxa.ne.0
c      let isx = 5 if method = 1 or method = 3 and
nxa.eq.0
c      let jsy = 0 if method = 0 or method = 1
c      let jsy = 3 if method = 2 or method = 3 and

```

```
nyc.ne.0
c           let jsy = 5 if method = 2 or method = 3 and
nyc.eq.0
c           then . . .
c
c           length =
[7* (nx+2) * (ny+2)+4* (11+isxt+jsy) *nx*ny]/3
c
c           will suffice in most cases. the exact
minimal work space
c           length required for the current nx,ny and
method is output
c           in iparm(16) (even if iparm(15) is too
small). this will be
c           less then the value given by the simplified
formula above
c           in most cases.
c
c
c ... fparm
c
c           a floating point vector of length 6 used to
efficiently
c           pass floating point arguments. fparm is set
internally
c           in cuh2cr and defined as follows . . .
c
c
c ... xa=fparm(1), xb=fparm(2)
c
c           the range of the x independent variable. xa
must
c           be less than xb
c
c
c ... yc=fparm(3), yd=fparm(4)
c
c           the range of the y independent variable. yc
must
c           be less than yd.
c
c
c ... tolmax = fparm(5)
c
```

```

c      when input positive, tolmax is a maximum
relative error tolerance
c      used to terminate the relaxation iterations.
assume phi1(i,j)
c      and phi2(i,j) are the last two computed
approximations at all
c      grid points of the finest grid level. if we
define
c
c      phdif = max(abs(phi2(i,j)-phi1(i,j))) for
all i,j
c
c      and
c
c      phmax = max(abs(phi2(i,j))) for all i,j
c
c      then "convergence" is considered to have
occurred if and only if
c
c      phdif/phmax < tolmax.
c
c
c      if tolmax=fparm(5)=0.0 is input then there is
no error control
c      and maxcy cycles from the finest grid level
are executed. maxcy
c      is a limit which cannot be exceeded even with
error control.
c      *** calls with tolmax=0.0, when appropriate
because of known
c      convergence behavior, are more efficient than
calls with tolmax
c      positive (i.e., if possible do not use error
control!).
c
c ... work
c
c      a complex saved work space (see iparm(15) for
size) which
c      must be preserved from the previous call when
calling with
c      intl=iparm(1)=1.
c
c ... bndyc

```

```

c
c      a subroutine with arguments
(kbdy,xory,alfa,beta,gama,gbdy) which
c      are used to input mixed boundary conditions
to cuh2cr. bndyc
c      must be declared "external" in the program
calling cuh2cr. kbdf
c      is type integer, xory real, and
alfa,beta,gama,gbdy type complex.
c      the boundaries are numbered one thru four and
the mixed
c      derivative boundary conditions are described
below (see the
c      sample driver code "tcuh2cr.f" for an example
of how bndyc is
c      can beset up) .
c
c      * * * * * * * * * * * * * * * * y=yd
c      *         kbdf=4          *
c      *                     *
c      *                     *
c      *                     *
c      * kbdf=1           kbdf=2   *
c      *                     *
c      *                     *
c      *                     *
c      *                     *
c      *         kbdf=3          *
c      * * * * * * * * * * * * * * * * y=yc
c
c      x=xa                  x=xb
c
c
c
c      (1)    the kbdf=1 boundary
c
c      this is the edge x=xa where nxa=iparm(2) = 2
flags
c      a mixed boundary condition of the form
c
c      alfx(x,y)*px + betx(x,y)*py +
gamt(x,y)*p(xa,y) = gbdx(x,y)
c
c      in this case kbdf=1,xory=y will be input to
bndyc and

```

```

c           alfa,beta,gama,gbdy corresponding to
alfa(y),betxa(y),gamxa(y),
c           gbdxa(y) must be returned. alfxa(y) = 0. is
not allowed for any y.
c           (see ierror = 13)
c
c   (2)    the kbdy=2 boundary
c
c           this is the edge x=xb where nxb=iparm(3) = 2
flags
c           a mixed boundary condition of the form
c
c           alfxb(y)*px + betxb(y)*py +
gamxb(y)*p(xb,y) = gbdxb(y)
c
c           in this case kbdy=2,xory=y will be input to
bndyc and
c           alfa,beta,gama,gbdy corresponding to
alfxb(y),betxb(y),gamxb(y),
c           gbdxb(y) must be returned. alfxb(y) = 0.0 is
not allowed for any y.
c           (see ierror = 13)
c
c   (3)    the kbdy=3 boundary
c
c           this is the edge y=yc where nyc=iparm(4) = 2
flags
c           a mixed boundary condition of the form
c
c           alfyc(x)*px + betyc(x)*py +
gamyc(x)*p(x,yc) = gbdyc(x)
c
c           in this case kbdy=3,xory=x will be input to
bndyc and
c           alfa,beta,gama,gbdy corresponding to
alfyc(x),betyc(x),gamyc(x),
c           gbdyc(x) must be returned. betyc(x) = 0.0 is
not allowed for any x.
c           (see ierror = 13)
c
c   (4)    the kbdy=4 boundary
c
c           this is the edge y=yd where nyd=iparm(5) = 2
flags

```

```

c           a mixed boundary condition of the form
c
c           alfyd(x)*px + betyd(x)*py +
gamyd(x)*p(x,yd) = gbdyd(x)
c
c           in this case kbdy=4,xory=x will be input to
bndyc and
c           alfa,beta,gama,gbdy corresponding to
alfyd(x),betyd(x),gamyd(x),
c           gbdyd(x) must be returned. betyd(x) = 0.0 is
not allowed for any x.
c           (see ierror = 13)
c
c
c ***      bndyc must provide the mixed boundary
condition values
c           in correspondence with those flagged in
iparm(2) thru
c           iparm(5). if all boundaries are specified or
periodic
c           cuh2cr will never call bndyc. even then it
must be entered
c           as a dummy subroutine. bndyc must be declared
"external"
c           in the routine calling cuh2cr. the actual
name chosen may
c           be different.
c
c
c ... coef
c
c           a subroutine with arguments
(x,y,cxx,cxy,cyy,cx,cy,ce) which
c           provides the known complex coefficients for
the elliptic pde at
c           any grid point (x,y). the name chosen in the
calling routine
c           may be different where the coefficient routine
must be declared
c           "external."
c
c ... rhs
c
c           a complex array dimensioned nx by ny which

```

```

contains the given
c           right hand side values on the uniform 2-d
mesh.
c
c           rhs(i,j) = r(xi,yj) for i=1,...,nx and
j=1,...,ny
c
c ... phi
c
c           a complex array dimensioned nx by ny.  on
input phi must contain
c           specified boundary values. for example, if
nyd=iparm(5)=1
c           then phi(i,ny) must be set equal to p(xi,yd)
for i=1,...nx
c           prior to calling cuh2cr.  these values are
preserved by cuh2cr.
c           if an initial guess is provided
(iguess=iparm(11)=1) it must
c           be input thru phi.
c
c
c ***      if no initial guess is given (iguess=0) then
phi must still
c           be initialized at all grid points (this is
not checked).  these
c           values will serve as an initial guess to the
pde at the coarsest
c           grid level after a transfer from the fine
solution grid.  set phi
c           equal to to 0.0 at all internal and non-
specified boundaries
c           grid points if nothing better is available.
c
c
c ... mgopt
c
c           an integer vector of length 4 which allows
the user to select
c           among various multigrid options.  if
mgopt(1)=0 is input then
c           a default set of multigrid arguments (chosen
for robustness)
c           will be internally selected and the

```

```
remaining values in mgopt
c           will be ignored. if mgopt(1) is nonzero
then the arguments
c           in mgopt are set internally and defined as
follows: (see the
c           basic coarse grid correction algorithm
below)

c
c
c      kcycle = mgopt(1)
c
c      = 0 if default multigrid options are to be
used
c
c      = 1 if v cycling is to be used (the least
expensive per cycle)
c
c      = 2 if w cycling is to be used (the
default)
c
c      > 2 if more general k cycling is to be used
c      *** warning--values larger than 2 increase
c      the execution time per cycle
considerably and
c           result in the nonfatal error ierror =
-5
c           which indicates inefficient multigrid
cycling.

c
c      iprер = mgopt(2)
c
c           the number of "pre-relaxation" sweeps
executed before the
c           residual is restricted and cycling is
invoked at the next
c           coarser grid level (default value is 2
whenever mgopt(1)=0)
c
c      ipost = mgopt(3)
c
c           the number of "post relaxation" sweeps
executed after cycling
c           has been invoked at the next coarser grid
level and the residual
```

```
c           correction has been transferred back
(default value is 1
c           whenever mgopt(1)=0) .
c
c *** if iprер, ipost, or (especially) kcycle is greater
than 2
c       than inefficient multigrid cycling has probably
been chosen and
c       the nonfatal error (see below) ierror = -5 will be
set. note
c       this warning may be overridden by any other
nonzero value
c       for ierror.
c
c   interpol = mgopt(4)
c
c           = 1 if multilinear prolongation
(interpolation) is used to
c           transfer residual corrections and the pde
approximation
c           from coarse to fine grids within full
multigrid cycling.
c
c           = 3 if multicubic prolongation
(interpolation) is used to
c           transfer residual corrections and the pde
approximation
c           from coarse to fine grids within full
multigrid cycling.
c           (this is the default value whenever
mgopt(1)=0) .
c
c *** the default values (2,2,1,3) in the vector mgopt
were chosen for
c       robustness. in some cases v(2,1) cycles with
linear prolongation will
c       give good results with less computation
(especially in two-dimensions).
c       this was the default and only choice in an
earlier version of mudpack
c       (see [1]) and can be set with the integer vector
(1,2,1,1) in mgopt.
c
c *** the schedules for one full multigrid cycle
```

```

(iguess=0) using v(2,1)
c      cycles and w(2,1) cycles are depicted for a four
level grid below.
c      the number of relaxation sweeps when each grid is
visited are indicated.
c      the "*" stands for prolongation of the full
approximation and the "."
c      stands for transfer of residuals and residual
corrections within the
c      coarse grid correction algorithm (see below). all
version 5.0.1
c      mudpack solvers use only fully weighted residual
restriction
c
c      one fmg with v(2,1) cycles:
c
c
c      -----
c      -----2-----1-----
c      ----- level 4
c
c      *
c      *
c      -----
c      -----2-----1-----2-----1-----
c      ----- level 3
c
c      *
c      *
c      -----
c      -----2-----1-----2-----1-----2-----1-----
c      ----- level 2
c
c      *
c      *
c      -----
c      ---3---3-----3-----3-----
c      ----- level 1
c
c
c      one fmg with w(2,1) cycles:
c
c
c      -----
c      -----2-----
c      --1-- level 4
c
c      *
c      .
c
c      -----
c      -----2-----1-----2-----3-----
c      1---- level 3
c
c      *
c      .
c      -----
c      -----2---1---2---3---1-----2---3---1---2---3---1-----
c      ----- level 2

```

```

C      * . . . . . . . . .
C      --6---6-----6--6-----6---6-----6---6---
----- level 1
C
C
C      the form of the "recursive" coarse grid correction
C      cycling used
C      when kcycle.ge.0 is input is described below in
C      pseudo-algorithmic
C      language. it is implemented non-recursively in
C      fortran in mudpack.
C
C      algorithm
cgc(k,l(k),u(k),r(k),kcycle,iprer,ipost,iressw,intpol)
C
C *** approximately solve l(k)*u(k) = r(k) using
C      multigrid iteration
C *** k is the current grid level
C *** l(k) is the discretized pde operator at level k
C *** u(k) is the initial guess at level k
C *** r(k) is the right hand side at level k
C *** i(k,k-1) is the restriction operator from level k
C      to level k-1
C *** (the form of i(k,k-1) depends on iressw)
C *** i(k-1,k) is the prolongation operator from level
C      k-1 to level k
C *** (the form of i(k-1,k) depends on intpol)
C
C      begin algorithm cgc
C
C *** pre-relax at level k
C
C      . do (i=1,iprer)
C
C      . . relax(l(k),u(k),r(k))
C
C      . end do
C
C      . if (k > 1) then
C
C ***      restrict the residual from level k to level k-
C      1
C
C      . . r(k-1) = i(k,k-1) (r(k)-l(k)*u(k))

```

```

C
C      . . kount = 0
C
C      . . repeat
C
C ***      solve for the residual correction at level k-1
in u(k-1)
C ***      using algorithm cgc "kcycle" times (this is
the recursion)
C
C      . . . kount = kount+1
C
C      . . . invoke cgc(k-1,l(k-1),u(k-1),r(k-
1),kcycle,iprер,ipost,iresw)
C
C
C      . . until (kount.eq.kcycle)
C
C ***      transfer residual correction in u(k-1) to
level k
C ***      with the prolongation operator and add to u(k)
C
C      . . u(k) = u(k) + i(k-1,k) (u(k-1))
C
C      . end if
C
C ***      post relax at level k
C
C      . do (i=1,ipost)
C
C      . . relax(l(k),u(k),r(k))
C
C      . end do
C
C      . return
C
C      end algorithm cgc
C
C
C
*****
*****  

C *** output arguments
*****

```

```

C
*****
*****  

C
C
C ... iparm(16) *** set for intl=0 calls only
C
C           on output iparm(16) contains the actual work
space length
C           required. this will usually be less than
that given by the
C           simplified formula for length=iparm(15) (see
as input argument)
C
C
C ... iparm(17) *** set for intl=1 calls only
C
C           on output iparm(17) contains the actual
number of multigrid cycles
C           between the finest and coarsest grid levels
used to obtain the
C           approximation when error control (tolmax >
0.0) is set.
C
C
C ... fparm(6) *** set for intl=1 calls with fparm(5)
> 0. only
C
C           on output fparm(6) contains the final
computed maximum relative
C           difference between the last two iterates at
the finest grid level.
C           fparm(6) is computed only if there is error
control (tolmax > 0.0)
C           assume phi1(i,j,k) and phi2(i,j,k) are the
last two computed
C           values for phi(i,j,k) at all points of the
finest grid level.
C           if we define
C
C           phdif = max (abs(phi2(i,j)-phi1(i,j)))
over all i,j
C
C           and

```

```
C
C           phmax = max(abs(phi2(i,j)) over all i,j
C
C           then
C
C           fparm(6) = phdif/phmax
C
C           is returned whenever phmax > 0.0. in the
C degenerate case
C           phmax = 0.0, fparm(6) = phdif is returned.
C
C
C ... work
C
C           on output work contains intermediate values
C that must not
C           be destroyed if cuh2cr is to be called again
C with intl=1
C
C
C ... phi   *** for intl=1 calls only
C
C           on output phi(i,j) contains the approximation
C to p(xi,yj)
C           for all mesh points i = 1,...,nx and
C j=1,...,ny. the last
C           computed iterate in phi is returned even if
C convergence is
C           not obtained
C
C
C ... ierror
C
C           for intl=iparm(1)=0 initialization calls,
C ierror is an
C           error flag that indicates invalid input
C arguments when
C           returned positive and nonfatal warnings when
C returned
C           negative. argument checking and
C discretization
C           is bypassed for intl=1 calls which can only
C return
C           ierror = -1 or 0 or 1.
```

```
c
c
c      non-fatal warnings * * *
c
c
c      ==5 if kcycle=mgopt(1) is greater than 2. values
lager than 2 results
c          in an algorithm which probably does far more
computation than
c          necessary.  kcycle = 1 (v cycles) or kcycle=2
(w cycles) should
c          suffice for most problems.  ierror = -5 is
also set if either
c          iprер = mgopt(2) or ipost=mgopt(3) is greater
than 2.  the
c          ierror=-5 flag is overridden by any other
fatal or non-fatal
c          error.
c
c      ==4 if there are dominant nonzero first order
terms in the pde which
c          make it "hyperbolic" at the finest grid level.
numerically, this
c          happens if:
c
c          abs(cx)*dlx > 2.*abs(cxx)    (dlx = (xb-
xa) / (nx-1))
c
c                      (or)
c
c          abs(cy)*dly > 2.*abs(cyy)    (dly = (yd-
yc) / (ny-1))
c
c
c          at some fine grid point (xi,yj).  if an
adjustment is not made the
c          condition can lead to a matrix coming from the
discretization
c          which is not diagonally dominant and
divergence is possible. since
c          the condition is "likely" at coarser grid
levels for pde's with
c          nonzero first order terms, the adjustments
(actually first order
```

```

c      approximations to the pde)
c
c
c      cxx = amax1(cxx,0.5*abs(cx)*dx)
c
c              (and)
c
c      cyy = amax1(cyy,0.5*abs(cy)*dy)
c
c
c      (here dx,dy are the x,y mesh sizes of the
c      subgrid)
c
c      are made to preserve convergence of multigrid
c      iteration. if made
c          at the finest grid level, it can lead to
c      convergence to an
c          erroneous solution (flagged by ierror = -4).
c      a possible remedy
c          is to increase resolution. the ierror = -4
c      flag overrides the
c          nonfatal ierror = -5 flag.
c
c
c
c      ==3 if the continuous elliptic pde is singular.
c      this means the
c          boundary conditions are periodic or pure
c      derivative at all
c          boundaries and ce(x,y) = 0.0 for all x,y. a
c      solution is still
c          attempted but convergence may not occur due
c      to ill-conditioning
c          of the linear system coming from the
c      discretization. the
c          ierror = -3 flag overrides the ierror=-4 and
c      ierror=-5 nonfatal
c          flags.
c
c
c
c      ==2 if the pde is not elliptic (i.e.,
c      cxx*cyy.le.0.0 for some (xi,yj))
c          in this case a solution is still attempted
c      although convergence
c          may not occur due to ill-conditioning of the

```

```

linear system.
c           the ierror = -2 flag overrides the ierror=-
5,-4,-3 nonfatal
c           flags.
c
c
c     ==1 if convergence to the tolerance specified in
tolmax=iparm(5)>0.
c           is not obtained in maxcy=iparm(13) multigrid
cycles between the
c           coarsest (ixp+1,jyq+1) and finest (nx,ny)
grid levels.
c           in this case the last computed iterate is
still returned.
c           the ierror = -1 flag overrides all other
nonfatal flags
c
c
c     no errors * * *
c
c     = 0
c
c     fatal argument errors * * *
c
c     = 1 if intl=iparm(1) is not 0 on initial call or
not 0 or 1
c           on subsequent calls
c
c     = 2 if any of the boundary condition flags
nxa,nxb,nyc,nyd
c           in iparm(2),iparm(3),iparm(4),iparm(5) are not
0,1 or 2
c           or if nxa,nxb or nyc,nyd are not pairwise
zero.
c
c     = 3 if min0(ixp,jyq) < 2 (ixp = iparm(6), jyq =
iparm(7))
c
c     = 4 if min0(iex,jey) < 1 (iex = iparm(8), jey =
iparm(9)) or
c           if max0(iex,jey) > 50
c
c     = 5 if nx.ne.ixp*2** (iex-1)+1 or
ny.ne.jyq*2** (jey-1)+1

```

```

c      (nx = iparm(10), ny = iparm(11))
c
c      = 6 if iguess = iparm(12) is not equal to 0 or 1
c
c      = 7 if maxcy = iparm(13) < 1
c
c      = 8 if method = iparm(14) is not 0,1,2, or 3
c
c      = 9 if length = iparm(15) is too small (see
iparm(16) on output
c          for minimum required work space length)
c
c      =10 if xa >= xb or yc >= yd
c
c      (xa=fparm(1),xb=fparm(2),yc=fparm(3),yd=fparm(4))
c
c      =11 if tolmax = fparm(5) < 0.0
c
c      errors in setting multigrid options * * * (see
also ierror=-5)
c
c      =12 if kcycle = mgopt(1) < 0 or
c          if iprer = mgopt(1) < 1 or
c          if ipost = mgopt(3) < 1 or
c          if interpol = mgopt(4) is not 1 or 3
c
c      =13 if there is a pure tangential derivative along
a mixed derivative
c          boundary (e.g., nyd = 2 and betyd(x) = 0.0 for
some
c          grid point x along y = yd)
c
c      =14 if there is the "singular" condition described
below at a
c          corner which is the intersection of two
derivative boundaries.
c
c          (1) the corner (xa,yc) if nxa=nyc=2 and
c              alfxa(yc)*betyc(xa)-alfyc(xa)*betxa(yc) =
0.0.
c
c          (2) the corner (xa,yd) if nxa=nyd=2 and
c              alfxa(yd)*betyd(xa)-alfyd(xa)*betxa(yd) =
0.0.

```

```

c      (3) the corner (xb,yc) if nxb=nyc=2 and
c          alfxb(yc)*betyc(xb)-alfyc(xb)*betxb(yc) =
0.0.

c
c      (4) the corner (xb,yd) if nxb=nyd=2 and
c          alfxb(yd)*betyd(xb)-alfyd(xb)*betxb(yd) =
0.0.

c
c *** the conditions described in ierror = 13 or 14 will
lead to division
c      by zero during discretization if undetected.

c
c
c
c ****
*
C
*****
*
C
*****
C
***** end of cuh2cr documentation
C
C
*****
**
C
*****
**
C

```

CUH3

```
* * * * * *
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK version 5.0.1
*
C      *
*
C      *               A Fortran Package of Multigrid
*
C      *
*
C      *               Subroutines and Example Programs
*
C      *
*
C      *               for Solving Elliptic Partial Differential
Equations        *
C      *
*
C      *               by
*
C      *
*
C      *               John Adams
*
C      *
*
C      *               of
*
C      *
*
C      *               the National Center for Atmospheric
```

```
Research          *
C      *
*
C      *                      Boulder, Colorado (80307)
U.S.A.          *
C      *
*
C      *                      which is sponsored by
*
C      *
*
C      *                      the National Science Foundation
*
C      *
*
C      *      *      *      *      *      *      *      *      *      *      *
*      *      *      *      *      *      *      *      *      *      *      *
C
C ... file cuh3.d
C
C      contains documentation for:
C      subroutine
cuh3(iparm,fparm,wk,iw,coef,bndyc,rhs,phi,mgopt,ierror)
C      a sample fortran driver is file "tcuh3.f".
C
C ... required mudpack files
C
C      cudcom.f, cud3ln.f, cud3pn.f
C
C ... purpose
C
C      the complex "hybrid" multigrid/direct method code
cuh3 approximates
C      the same 3-d nonseparable elliptic pde as the
mudpack solver cud3.
C      the basic algorithm is modified by using block
banded gaussian
C      elimination in place of relaxation whenever the
coarsest subgrid is
C      encountered within multigrid cycling.  use of the
direct method at
C      the coarsest grid level gives cuh3 at least two
advantages over cud3:
C
```

```
c      (1) improved convergence rates
c
c      the use of a direct method at the coarsest
grid level can
c      improve convergence rates at a small
additional computational
c      cost if the coarse grid parameters
ixp=iparm(8),jyq=iparm(9),
c      kzs=iparm(10) are small relative to the fine
grid parameters
c      nx=iparm(14),ny=iparm(15),nz=iparm(16). this
is especially true
c      in the presence of certain boundary
conditions. for example,
c      if all boundary conditions are neuman (pure
derivative) and/or
c      periodic then cud3 may fail to converge. cuh3
should handle
c      these boundary conditions with the expected
multigrid efficiency
c      (see tcuh3.f). in all cases, cuh3 should give
convergence
c      rates which equal or exceed cud3.
c
c
c      (2) more resolution choices
c
c      cuh3 allows more grid size flexibility by
"relaxing" the
c      constraint on the coarse grid parameters that
ixp,jyq,kzs
c      be "very" small (2 or 3 for cud3) for
effective error
c      error reduction within multigrid cycling.
convergence
c      rates will not deteriorate with larger values
for ixp,
c      jyq,kzs.
c
c *** caution
c
c      because of the very large computational and
storage
c      requirements, the three-dimensional dimensional
```

```

direct
c      method costs can overwhelm the multigrid cycling
costs
c      if the coarsest grid is not small relative to the
finest
c      solution grid.  this is a user decision set by the
choice
c      of coarse and fine grid parameters (see iparm(8)
through
c      iparm(16) and iparm(21) descriptions)
c
c      subroutine cuh3 automatically discretizes and
attempts to compute
c      the second order finite difference approximation
to a three-
c      dimensional linear nonseparable elliptic partial
differential
c      equation on a box.  the approximation is generated
on a uniform
c      grid covering the box (see mesh description
below).  boundary
c      conditions may be any combination of mixed,
specified (Dirchlet)
c      or periodic.  the form of the pde solved is . . .
c
c      cxx(x,y,z)*pxx + cyy(x,y,z)*pyy +
czz(z,y,z)*pzz +
c
c      cx(x,y,z)*px + cy(x,y,z)*py + cz(x,y,z)*pz +
c
c      ce(x,y,z)*p(x,y,z) = r(x,y,z)
c
c      here cxx,cyy,czz,cx,cy,cz,ce are the known complex
coefficients
c      of the pde; pxx,pyy,pzz,px,py,pz are the second
and first partial
c      derivatives of the unknown complex solution
function p(x,y,z)
c      with respect to the independent variables x,y,z;
r(x,y,z) is
c      is the known complex right hand side of the
elliptic pde.  cxx,cyy
c      and czz should have real or imaginary parts
positive for all (x,y,z)

```

```

c
c
c ... argument differences with cud3.f
c
c      the input and output arguments of cuh3 are almost
c      identical to the
c      arguments of cud3 (see cud3.d) with the following
c      exceptions:
c
c      (1) let mx=ixp+1, my=jyq+1, mz=kzr+1 (the coarsest
c      grid
c          resolutions, ixp=iparm(8), jyq=iparm(9),
c          kzr=iparm(10))
c          then the work space vector "wk" requires
c
c          mx*my*mz* (2*mx*my+1))
c (nze.ne.0)
c
c          additional words of storage if periodic
c boundary conditions
c          are not flagged in the z direction or
c
c          mx*my* (mz* (4*mx*my+1))           (nze=0)
c
c          additional words of storage if periodic
c boundary conditions are
c          flagged in the z direction (nze = 0). the
c extra work space is
c          used for a direct solution with gaussian
c elimination whenever the
c          coarsest grid is encountered within multigrid
c cycling.
c
c      (2) an integer work space iwk of length at least
c      mx*my*mz words
c          must be provided. the length of iwk is not
c checked!
c
c      (3) kzr > 2 if nze=0, jyq > 2 if nyc=0, ixp > 2 if
c      nxe = 0 are
c          required (i.e., the coarsest grid must contain
c at least four
c          points in any direction with periodic boundary
c conditions,

```

```
c      see the expanded meaning of ierror=3).
c
c *** (4) it is no longer necessary that ixp,jyq,kzr be
2 or 3 for
c      effective error reduction with multigrid
iteration. there
c      is no reduction in convergence rates when
larger values for
c      ixp,jyq,kzr are used . this provides
additional flexibility
c      in choosing grid size. in many cases cuh3
provides more
c      robust convergence than cud3. it can be used
in place of
c      cud3 for all nonsingular problems (see (5)
below).
c
c      (5) iguess = iparm(17) = 1 (flagging an initial
guess) or
c      maxcy = iparm(18) > 1 (setting more than one
multigrid
c      cycle) are not allowed if cuh3 becomes a full
direct method
c      by choosing iex=jey=kez=1. this conflicting
combination
c      of input arguments for multigrid iteration and
a full
c      direct method set the fatal error flag
c
c      ierror = 13
c
c      iguess = 0 and maxcy = 1 are required when
cuh3 becomes a
c      full direct method. ordinarily (see ***
caution above) this
c      should not happen except when testing with
very coarse resolution.
c
c
c      (6) if a "singular" pde is detected (see ierror=-3
description in
c      cud3.d, ce(x,y) = 0.0 for all x,y and the
boundary conditions
c      are a combination of periodic and/or pure
```

```

derivatives) then cuh3
c           sets the fatal error flag
c
c           ierror = 14
c
c           the direct method utilized by cuh3 would
likely cause a near
c           division by zero in the singular case.  cud3
can be tried for
c           singular problems.
c
c ... grid size considerations
c
c     (1) flexibility
c
c           cuh3 should be used in place of cud3 whenever
grid size
c           requirements do not allow choosing ixp,jyq,kzr
to be 2 or 3.
c
c     (2) additional work space (see (1) under
"arguments differences") is
c           required by cuh3 to implement gaussian
elimination at the coarsest
c           grid level.  this may limit the size of
ixp,jyq,kzr.
c
c     (3) operation counts
c
k
c           for simplicity, assume p=ixp=jyq=kzr and
n=nx=ny=nz=2 *p.
c           gaussian elimination requires  $\mathcal{O}(p^{**7})$ 
operations for solution
c           on the coarsest subgrid while multigrid
iteration is a  $\mathcal{O}(n^{**3})$ 
c           algorithm.  consequently the storage and
computational
c           requirements for the 3-d direct method will
dominate the
c           calculation if p is "large."  note that
 $\mathcal{O}(p^{**7}) =: \mathcal{O}(n^{**3})$ 
c           whenever k =: (4/3)*log2(p) grid levels are
used in cycling.

```

```
c           larger values mean the direct method will
dominate the
c           calculation. smaller values for k mean the
direct method
c           will only marginally add to the cost of
multigrid iteration
c           alone.
c
c *** the remaining documentation is almost identical to
cud3.d
c       except for the modifications already indicated.
c
c
c ... mesh description . . .
c
c       the approximation is generated on a uniform nx by
ny by nz grid.
c       the grid is superimposed on the rectangular
solution region
c
c       [xa,xb] x [yc,yd] x [ze,zf].
c
c       let
c
c       dlx = (xb-xa) / (nx-1), dly = (yd-yc) / (ny-1),
dlz = (zf-ze) / (nz-1)
c
c       be the uniform grid increments in the x,y,z
directions. then
c
c       xi=xa+(i-1)*dlx,   yj=yc+(j-1)*dly,  zk =
ze+(k-1)*dlz
c
c       for i=1,...,nx; j=1,...,ny; k=1,...,nz  denote the
x,y,z uniform
c       mesh points.
c
c
c ... language
c
c       fortran90/fortran77
c
c
c ... portability
```

```
c
c      mudpack5.0.1 software has been compiled and tested
with fortran77
c      and fortran90 on a variety of platforms.
c
c ... methods
c
c      details of the methods employed by the solvers in
mudpack are given
c      in [1,9]. [1,2,9] contain performance
measurements on a variety of
c      elliptic pdes (see "references" in the file
"readme"). in summary:
c
c *** discretization and solution (second-order solvers)
(see [1])
c
c      the pde and boundary conditions are automatically
discretized at all
c      grid levels using second-order finite difference
formula. diagonal
c      dominance at coarser grid levels is maintained in
the presence of
c      nonzero first-order terms by adjusting the second-
order coefficient
c      when necessary. the resulting block tri-diagonal
linear system is
c      approximated using multigrid iteration
[10,11,13,15,16,18]. version
c      5.0.1 of mudpack uses only fully weighted residual
restriction. defaults
c      include cubic prolongation and w(2,1) cycles.
these can be overridden
c      with selected multigrid options (see "mgopt").
error control based on
c      maximum relative differences is available. full
multigrid cycling (fmg)
c      or cycling beginning or restarting at the finest
grid level can be
c      selected. a menu of relaxation methods including
gauss-seidel point,
c      line relaxation(s) (in any combination of
directions) and planar
c      relaxation (for three-dimensional anisotropic
```

```
problems) are provided.  
c      all methods use ordering based on alternating  
points (red/black),  
c      lines, or planes for cray vectorization and  
improved convergence  
c      rates [14].  
c  
c *** higher order solution (fourth-order solvers) (see  
[9,19,21])  
c  
c      if the multigrid cycling results in a second-order  
estimate (i.e.,  
c      discretization level error is reached) then this  
can be improved to a  
c      fourth-order estimate using the technique of  
"deferred corrections."  
c      the values in the solution array are used to  
generate a fourth-order  
c      approximation to the truncation error. second-  
order finite difference  
c      formula are used to approximate third and fourth  
partial derivatives  
c      of the solution function [3]. the truncation  
error estimate is  
c      transferred down to all grid levels using weighted  
averaging where  
c      it serves as a new right hand side. the default  
multigrid options  
c      are used to compute the fourth-order correction  
term which is added  
c      to the original solution array.  
c  
c  
c ... references (partial)  
c  
c  
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software  
for the Efficient  
c      Solution of Linear Elliptic Partial Differential  
Equations,"  
c      Applied Math. and Comput. vol.34, Nov 1989,  
pp.113-146.  
c  
c      [2] J. Adams, "FMG Results with the Multigrid
```

```
Software Package MUDPACK,"  
c      proceedings of the fourth Copper Mountain  
Conference on Multigrid, SIAM,  
c      1989, pp.1-12.  
c      .  
c      .  
c      .  
c      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.  
Haidvogel, and V. Pizzo,  
c      "Applications of Multigrid Software in the  
Atmospheric Sciences,"  
c      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-  
1458.  
c      .  
c      .  
c      .  
c      [9] J. Adams, "Recent Enhancements in MUDPACK, a  
Multigrid Software  
c      package for Elliptic Partial Differential  
Equations," Applied Math.  
c      and Comp., 1991, vol. 43, May 1991, pp. 79-94.  
c  
c      [10]J. Adams, "MUDPACK-2: Multigrid Software for  
Approximating  
c      Elliptic Partial Differential Equations on Uniform  
Grids with  
c      any Resolution," Applied Math. and Comp., 1993,  
vol. 53, February  
c      1993, pp. 235-249  
c      .  
c      .  
c      .  
c  
c ... argument description  
c  
c  
c  
*****  
*****  
c *** input arguments  
*****  
c  
*****  
*****
```

```
c
c
c ... iparm
c
c           an integer vector of length 23 used to
efficiently pass
c           integer arguments. iparm is set internally
in cuh3
c           and defined as follows . . .
c
c
c ... intl=iparm(1)
c
c           an initialization argument. intl=0 must be
input
c           on an initial call. in this case input
arguments will
c           be checked for errors and the elliptic
partial differential
c           equation and boundary conditions will be
discretized using
c           second order finite difference formula.
c
c ***      an approximation is not generated after an
intl=0 call!
c           cuh3 should be called with intl=1 to
approximate the elliptic
c           pde discretized by the intl=0 call. intl=1
should also
c           be input if cuh3 has been called earlier and
only the
c           values in in rhs (see below) or gbdy (see
bndyc below)
c           or phi (see below) have changed. this will
bypass
c           redundant pde discretization and argument
checking
c           and save computational time. some examples
of when
c           intl=1 calls should be used are:
c
c           (0) after a intl=0 argument checking and
discretization call
c
```

```
c           (1) cuh3 is being recalled for additional
accuracy.  in
c           this case iguess=iparm(12)=1 should also
be used.
c
c           (2) cuh3 is being called every time step in a
time dependent
c           problem (see discussion below) where the
elliptic operator
c           does not depend on time.
c
c           (3) cuh3 is being used to solve the same
elliptic equation
c           for several different right hand sides
(igueess=0 should
c           probably be used for each new righthand
side).
c
c           intl = 0 must be input before calling with
intl = 1 when any
c           of the following conditions hold:
c
c           (a) the initial call to cuh3
c
c           (b) any of the integer arguments other than
igueess=iparm(12)
c           or maxcy=iparm(13) or mgopt have changed
since the previous
c           call.
c
c           (c) any of the floating point arguments other
than tolmax=
c           fparm(5) have changed since the previous
call
c
c           (d) any of the coefficients input by coef
(see below) have
c           changed since the previous call
c
c           (e) any of the "alfa" coefficients input by
bndyc (see below)
c           have changed since the previous call.
c
c           if any of (a) through (e) are true then the
```

```
elliptic pde
c           must be discretized or rediscretized.  if
none of (a)
c           through (e) holds, calls can be made with
intl=1.
c           incorrect calls with intl=1 will produce
erroneous results.
c ***      the values set in the saved work space "wk"
(see below) with
c           an intl=0 call must be preserved with
subsequent intl=1 calls.
c
c           MUDPACK software performance should be
monitored for intl=1
c           calls.  The intl=0 discretization call
performance depends
c           primarily on the efficiency or lack of
efficiency of the
c           user provided subroutines for pde
coefficients and
c           boundary conditions.

c
c
c ... nxa=iparm(2)
c
c           flags boundary conditions on the (y,z) plane
x=xa
c
c           = 0 if p(x,y,z) is periodic in x on [xa,xb]
c           (i.e., p(x+xb-xa,y,z) = p(x,y,z) for all
x,y,z)
c
c           = 1 if p(xa,y,z) is specified (this must be
input thru phi(1,j,k))
c
c           = 2 if there are mixed derivative boundary
conditions at x=xa
c           (see "bndyc" description below where kbdy =
1)
c
c
c ... nxb=iparm(3)
c
c           flags boundary conditions on the (y,z) plane
```

```

x=xb
c
c      = 0 if p(x,y,z) is periodic in x on [xa,xb]
c          (i.e., p(x+xb-xa,y,z) = p(x,y,z) for all
x,y,z)
c
c      = 1 if p(xb,y,z) is specified (this must be
input thru phi(nx,j,k))
c
c      = 2 if there are mixed derivative boundary
conditions at x=xb
c          (see "bndyc" description below where kbdy =
2)
c
c
c ... nyc=iparm(4)
c
c      flags boundary conditions on the (x,z) plane
y=yc
c
c      = 0 if p(x,y,z) is periodic in y on [yc,yd]
c          (i.e., p(x,y+yd-yc,z) = p(x,y,z) for all
x,y,z)

c      = 1 if p(x,yc,z) is specified (this must be
input thru phi(i,1,k))

c      = 2 if there are mixed derivative boundary
conditions at y=yc
c          (see "bndyc" description below where kbdy =
3)
c
c
c ... nyd=iparm(5)
c
c      flags boundary conditions on the (x,z) plane
y=yd
c
c      = 0 if p(x,y,z) is periodic in y on [yc,yd]
c          (i.e., p(x,y+yd-yc,z) = p(x,y,z) for all
x,y,z)

c      = 1 if p(x,yd,z) is specified (this must be
input thru phi(i,ny,k))

```

```

c      = 2 if there are mixed derivative boundary
conditions at y=yd
c      (see "bndyc" description below where kbdy =
4)
c
c
c ... nze=iparm(6)
c
c      flags boundary conditions on the (x,y) plane
z=ze
c
c      = 0 if p(x,y,z) is periodic in z on [ze,zf]
c      (i.e., p(x,y,z+zf-ze) = p(x,y,z) for all
x,y,z

c      = 1 if p(x,y,ze) is specified (this must be
input thru phi(i,j,1))

c      = 2 if there are mixed derivative boundary
conditions at z=ze
c      (see "bndyc" description below where kbdy =
5)
c
c
c ... nzf=iparm(7)
c
c      flags boundary conditions on the (x,y) plane
z=zf
c
c      = 0 if p(x,y,z) is periodic in z on [ze,zf]
c      (i.e., p(x,y,z+zf-ze) = p(x,y,z) for all
x,y,z

c      = 1 if p(x,y,zf) is specified (this must be
input thru phi(i,j,nz))

c      = 2 if there are mixed derivative boundary
conditions at z=zf
c      (see "bndyc" description below where kbdy =
6)
c
c
c *** grid size arguments

```

```
c  
c  
c ... ixp = iparm(8)  
c  
c           an integer greater than one which is used in  
defining the number  
c           of grid points in the x direction (see nx =  
iparm(14)). "ixp+1"  
c           is the number of points on the coarsest x  
grid visited during  
c           multigrid cycling. ixp should be chosen as  
small as possible  
c           within grid size requirements.  
c  
c  
c ... jyq = iparm(9)  
c  
c           an integer greater than one which is used in  
defining the number  
c           of grid points in the y direction (see ny =  
iparm(15)). "jyq+1"  
c           is the number of points on the coarsest y  
grid visited during  
c           multigrid cycling. jyq should be chosen as  
small as possible  
c           within grid size requirements.  
c  
c  
c ... kzs = iparm(10)  
c  
c           an integer greater than one which is used in  
defining the number  
c           of grid points in the z direction (see nz =  
iparm(16)). "kzs+1"  
c           is the number of points on the coarsest z  
grid visited during  
c           multigrid cycling. kzs should be chosen as  
small as possible  
c           within grid size requirements.  
c  
c  
c ... iex = iparm(11)  
c  
c           a positive integer exponent of 2 used in
```

```
defining the number
c          of grid points in the x direction (see nx =
iparm(14)).
c          iex .le. 50 is required. for efficient
multigrid cycling,
c          iex should be chosen as large as possible and
ixp=iparm(8)
c          as small as possible within grid size
constraints when
c          defining nx = iparm(14).

c
c
c ... jey = iparm(12)
c
c          a positive integer exponent of 2 used in
defining the number
c          of grid points in the y direction (see ny =
iparm(15)).
c          jey .le. 50 is required. for efficient
multigrid cycling,
c          jey should be chosen as large as possible and
jyq=iparm(9)
c          as small as possible within grid size
constraints when
c          defining ny = iparm(15).

c
c
c ... kez = iparm(13)
c
c          a positive integer exponent of 2 used in
defining the number
c          of grid points in the z direction (see nz =
iparm(16)).
c          kez .le. 50 is required. for efficient
multigrid cycling,
c          kez should be chosen as large as possible and
kzr=iparm(10)
c          as small as possible within grid size
constraints when
c          defining nz = iparm(16).

c
c
c ... nx = iparm(14)
c
```

```

c           the number of equally spaced grid points in
the interval [xa,xb]
c           (including the boundaries).  nx must have the
form
c
c           nx = ixp*(2**(iex-1)) + 1
c
c           where ixp = iparm(8), iex = iparm(11).
c
c
c ... ny = iparm(15)
c
c           the number of equally spaced grid points in
the interval [yc,yd]
c           (including the boundaries).  ny must have the
form:
c
c           ny = jyq*(2**(jey-1)) + 1
c
c           where jyq = iparm(9), jey = iparm(12).
c
c
c ... nz = iparm(16)
c
c           the number of equally spaced grid points in
the interval [ze,zf]
c           (including the boundaries).  nz must have the
form
c
c           nz = kzs*(2**(kez-1)) + 1
c
c           where kzs = iparm(10), kez = iparm(13)
c
c
c *** note
c
c     let g be the nx by ny by nz fine grid on which the
approximation is
c     generated and let n = max0(iex,jey,kez).  in
mudpack, multigrid
c     cycling is implemented on the ascending chain of
grids
c
c           g(1) < ... < g(k) < ... < g(n) = g.

```

```

c
c      each g(k) (k=1,...,n) has mx(k) by my(k) by mz(k)
grid points
c      given by:
c
c          mx(k) = ixp*[2** (max0(iex+k-n,1)-1)] + 1
c
c          my(k) = jyq*[2** (max0(jey+k-n,1)-1)] + 1
c
c          mz(k) = kxr*[2** (max0(kez+k-n,1)-1)] + 1
c
c      additionally cuh3 implements a direct method
whenever g(1) is
c      encountered.
c
c ... iguess=iparm(17)
c
c          = 0 if no initial guess to the pde is
provided
c          and/or full multigrid cycling beginning
at the
c          coarsest grid level is desired.
c
c          = 1 if an initial guess to the pde at the
finest grid
c          level is provided in phi (see below). in
this case
c          cycling beginning or restarting at the
finest grid
c          is initiated.
c
c *** comments on iguess = 0 or 1 . . .
c
c
c      setting iguess=0 forces full multigrid or "fmg"
cycling. phi
c      must be initialized at all grid points. it can be
set to zero at
c      non-Dirchlet grid points if nothing better is
available.
c
c      if iguess = 1 then the values input in phi are an
initial guess to the
c      pde at the finest grid level where cycling begins.

```

```

this option should
c      be used only if a "very good" initial guess is
available (as, for
c      example, when restarting from a previous iguess=0
call).
c
c *** time dependent problems . . .
c
c      assume we are solving an elliptic pde every time
step in a
c      marching problem of the form:
c
c          l(p(t)) = r(t)
c
c      where the differential operator "l" has no time
dependence,
c      "p(t)" is the solution and "r(t)" is the right
hand side at
c      current time "t". let "dt" be the increment
between time steps.
c      then p(t) can be used as an initial guess to
p(t+dt) with
c      intl = 1 when solving
c
c          l(p(t+dt)) = r(t+dt).
c
c      after the first two time steps, rather than
continue, it would
c      be better to define the "correction" term:
c
c          e(t,dt) = p(t+dt) - p(t)
c
c      this clearly satisfies the equation
c
c          l(e(t,dt)) = r(t+dt) - r(t).
c
c      this should be solved with iguess = 0 and intl =
1. boundary
c      conditions for e(t,dt) are obtained from the
boundary conditions
c      for p(t) by subtracting given values at t from
given values at
c      t+dt. for example if
c

```

```

c      d(p(t))/dx = f(t), d(p(t+dt))/dx = f(t+dt)
c
c      at some x boundary then e(t,dt) satisfies the
derivative
c      boundary condition
c
c      d(e(t,dt))/dx = f(t+dt) - f(t).
c
c      e(t,dt) can be preset to 0.0 (at non-Dirchlet
points) or (if p(t-dt)
c      is saved) to p(t)-p(t-dt). with iguess = 0, these
values will serve
c      as an initial guess to e(t,dt) at the coarsest
grid level. this
c      approach has the advantage that a full sequence of
multigrid cycles,
c      beginning at the coarsest grid level, is invoked
every time step in
c      solving for e(t,dt). a few digits of accuracy in
e(t,dt), which is
c      ordinarily much smaller than p(t), will yield
several more digits of
c      accuracy in the final approximation:
c
c      p(t+dt) = p(t) + e(t,dt).
c
c      using this approach to integrate in time will give
more accuracy
c      then using p(t) as an initial guess to p(t+dt) for
all time steps.
c      it does require additional storage.
c
c      if the differential operator "l" has time
dependence (either thru
c      the coefficients in the pde or the coefficients in
the derivative
c      boundary conditions) then use p(t) as an initial
guess to p(t+dt)
c      when solving
c
c      l(t+dt)(p(t+dt)) = r(t+dt)
c
c      with intl = 0 for all time steps (the
discretization must be repeated

```

```

c      for each new "t"). either iguess = 0 (p(t) will
then be an initial
c      guess at the coarsest grid level where cycles will
commence) or
c      iguess = 1 (p(t) will then be an initial guess at
the finest grid
c      level where cycles will remain fixed) can be
tried.
c
c
c ... maxcy = iparm(18)
c
c           the exact number of cycles executed between
the finest
c           (nx by ny by nz) and the coarsest ((ixp+1) by
(jyq+1) by
c           (kzr+1)) grid levels when tolmax=fparm(7)=0.0
(no error
c           control). when tolmax=fparm(7).gt.0.0 is
input (error control)
c           then maxcy is a limit on the number of cycles
between the
c           finest and coarsest grid levels. in any
case, at most
c           maxcy*(iprert+ipost) relaxation sweeps are
performed at the
c           finest grid level (see
iprer=mgopt(2),ipost=mgopt(3) below)
c           when multigrid iteration is working
"correctly" only a few
c           cycles are required for convergence. large
values for maxcy
c           should not be required.
c
c
c ... method = iparm(19)
c
c           this sets the method of relaxation (all
relaxation
c           schemes in mudpack use red/black type
ordering)
c
c           = 0 for gauss-seidel pointwise relaxation
c

```

```
C      = 1 for line relaxation in the x direction
C
C      = 2 for line relaxation in the y direction
C
C      = 3 for line relaxation in the z direction
C
C      = 4 for line relaxation in the x and y
direction
C
C      = 5 for line relaxation in the x and z
direction
C
C      = 6 for line relaxation in the y and z
direction
C
C      = 7 for line relaxation in the x,y and z
direction
C
C      = 8 for x,y planar relaxation
C
C      = 9 for x,z planar relaxation
C
C      =10 for y,z planar relaxation
C
C *** choice of method
C
C      this is very important for efficient convergence.
in some cases
C      experimentation may be required.
C
C      let fx represent the quantity cxx(x,y,z)/dlx**2
over the solution box
C
C      let fy represent the quantity cyy(x,y,z)/dly**2
over the solution box
C
C      let fz represent the quantity czz(x,y,z)/dlz**2
over the solution box
C
C      (0) if fx,fy,fz are roughly the same size and do
not vary too
C            much choose method = 0.  if this fails try
method = 7.
C
```

```
c      (1) if fx is much greater than fy,fz and fy,fz  
are roughly the same  
c          size choose method = 1  
c  
c      (2) if fy is much greater than fx,fz and fx,fz  
are roughly the same  
c          size choose method = 2  
c  
c      (3) if fz is much greater than fx,fy and fx,fy  
are roughly the same  
c          size choose method = 3  
c  
c      (4) if fx,fy are roughly the same and both are  
much greater than fz  
c          try method = 4.  if this fails try method = 8  
c  
c      (5) if fx,fz are roughly the same and both are  
much greater than fy  
c          try method = 5.  if this fails try method = 9  
c  
c      (6) if fy,fz are roughly the same and both are  
much greater than fx  
c          try method = 6.  if this fails try method =  
10  
c  
c      (7) if fx,fy,fz vary considerably with none  
dominating try method = 7  
c  
c      (8) if fx and fy are considerably greater than fz  
but not necessarily  
c          the same size and method=4 fails try method =  
8  
c  
c      (9) if fx and fz are considerably greater than fy  
but not necessarily  
c          the same size and method=5 fails try method =  
9  
c  
c      (10)if fy and fz are considerably greater than fx  
but not necessarily  
c          the same size and method=6 fails try method =  
10  
c  
c
```

```

c ... meth2 = iparm(20) determines the method of
relaxation used in the planes
c           when method = 8 or 9 or 10.
c
c
c           as above, let fx,fy,fz represent the
quantities cxx/dlx**2,
c           cyy/dly**2,czz/dlz**2 over the box.
c
c           (if method = 8)
c
c           = 0 for gauss-seidel pointwise relaxation
c               in the x,y plane for each fixed z
c           = 1 for line relaxation in the x direction
c               in the x,y plane for each fixed z
c           = 2 for line relaxation in the y direction
c               in the x,y plane for each fixed z
c           = 3 for line relaxation in the x and y
direction
c               in the x,y plane for each fixed z
c
c           (1) if fx,fy are roughly the same and vary
little choose meth2 = 0
c           (2) if fx is much greater then fy choose
meth2 = 1
c           (3) if fy is much greater then fx choose
meth2 = 2
c           (4) if none of the above or meth2 = 0 fails
choose meth2 = 3
c
c           (if method = 9)
c
c           = 0 for gauss-seidel pointwise relaxation
with red/black ordering
c               in the x,z plane for each fixed y
c           = 1 for simultaneous line relaxation in the x
direction
c               of the x,z plane for each fixed y
c           = 2 for simultaneous line relaxation in the z
direction
c               of the x,z plane for each fixed y
c           = 3 for simultaneous line relaxation in the x
and z direction
c               of the x,z plane for each fixed y

```

```

c
c           (1) if fx,fz are roughly the same and vary
little choose meth2 = 0
c           (2) if fx is much greater then fz choose
meth2 = 1
c           (3) if fz is much greater then fx choose
meth2 = 2
c           (4) if none of the above or meth2 = 0 fails
choose meth2 = 3
c
c           (if method = 10)
c
c           = 0 for gauss-seidel pointwise relaxation
with red/black ordering
c           in the y,z plane for each fixed x
c           = 1 for simultaneous line relaxation in the y
direction
c           of the y,z plane for each fixed x
c           = 2 for simultaneous line relaxation in the z
direction
c           of the y,z plane for each fixed x
c           = 3 for simultaneous line relaxation in the y
and z direction
c           of the y,z plane for each fixed x
c
c           (1) if fy,fz are roughly the same and vary
little choose meth2 = 0
c           (2) if fy is much greater then fz choose
meth2 = 1
c           (3) if fz is much greater then fy choose
meth2 = 2
c           (4) if none of the above or meth2 = 0 fails
choose meth2 = 3
c
c
c *** note that planar relaxation implements full two-
dimensional multigrid
c     cycling for each plane visited during three
dimensional multigrid
c     cycling. Consequently it can be computationally
expensive.
c
c ... length = iparm(21)
c

```

```

c           the length of the work space provided in
vector wk.

c
c           let isx = 3 if method = 1,4,5 or 7 and
nxa.ne.0
c           let isx = 5 if method = 1,4,5 or 7 and
nxa.eq.0
c           let isx = 0 if method has any other value
c
c           let jsy = 3 if method = 2,4,6 or 7 and
nyc.ne.0
c           let jsy = 5 if method = 2,4,6 or 7 and
nyc.eq.0
c           let jsy = 0 if method has any other value
c
c           let ksz = 3 if method = 3,5,6 or 7 and
nze.ne.0
c           let ksz = 5 if method = 3,5,6 or 7 and
nze.eq.0
c           let ksz = 0 if method has any other value
c
c
c           let ls =
(nx+2) * (ny+2) * (nz+2) * (10+isx+jsy+ksz)
c
c           let mx = ixp+1; my = jyq+1; mz = kzs+1.  the
block gaussian
c           elimination at the coarsest mx by my by mz
grid level requires
c
c           ld = mx*my*mz*(2*mx*my+1)
(nze.ne.0)
c
c           words of storage if z boundary conditions are
not periodic or
c
c           ld = mx*my*(mz*(4*mx*my+1))
(nze=0)
c
c           words of storage if z boundary conditions are
periodic.
c           if ixp,jyq,kzs are not the same, this
quantity is

```

```
c           minimized if they are chosen so that kzs >
max0(ixp,jyq) .
c
c           finally
c
c           length = ls + ld

c           will usually but not always suffice.  the
exact minimal length depends,
c           in a complex way, on the grid size arguments
and method chosen.
c ***      it can be predetermined for the current input
arguments by calling
c           cuh3 with length=iparm(21)=0 and printing
iparm(22) or (in f90)
c           dynamically allocating the work space using
the value in iparm(22)
c           in a subsequent cuh3 call.
c
c ... fparm
c
c           a floating point vector of length 8 used to
efficiently
c           pass floating point arguments.  fparm is set
internally
c           in cuh3 and defined as follows . . .
c
c
c ... xa=fparm(1), xb=fparm(2)
c
c           the range of the x independent variable.  xa
must
c           be less than xb
c
c
c ... yc=fparm(3), yd=fparm(4)
c
c           the range of the y independent variable.  yc
must
c           be less than yd.
c
c
c ... ze=fparm(5), zf=fparm(6)
c
```

```
c           the range of the z independent variable. ze
must
c           be less than zf.
c
c
c ... tolmax = fparm(7)
c
c           when input positive, tolmax is a maximum
relative error tolerance
c           used to terminate the relaxation iterations.
assume phil(i,j,k)
c           and phi2(i,j,k) are the last two computed
approximations at all
c           grid points of the finest grid level. if we
define
c
c           phdif = max(abs(phi2(i,j,k)-phil(i,j,k)))
for all i,j,k
c
c           and
c
c           phmax = max(abs(phi2(i,j,k))) for all
i,j,k
c
c           then "convergence" is considered to have
occurred if and only if
c
c           phdif/phmax < tolmax.
c
c
c           if tolmax=fparm(7)=0.0 is input then there is
no error control
c           and maxcy cycles from the finest grid level
are executed. maxcy
c           is a limit which cannot be exceeded even with
error control.
c           *** calls with tolmax=0.0, when appropriate
because of known
c           convergence behavior, are more efficient than
calls with tolmax
c           positive (i.e., if possible DO NOT use error
control!).
c
c
```

```

c ... wk
c
c           a one dimensional array that must be provided
for work space.
c           see length = iparm(21). the values in wk must
be preserved
c           if cuh3 is called again with
intl=iparm(1).ne.0 or if cuh34
c           is called to improve accuracy.
c
c
c ... iwk
c
c           an integer vector dimensioned of length at
least
c
c           (ixp+1)*(jyq+1)*(kzr+1)
c
c           in the routine calling cuh3. the length of
iwk is not
c           checked! if iwk has length too small then
undetectable
c           undetectable errors will result.
c
c
c ... bndyc
c
c           a subroutine with arguments
(kbdy,xory,yorz,alfa,gbdy) .
c           which are used to input mixed boundary
conditions to cuh3.
c           the boundaries are numbered one thru six and
the form of
c           conditions are described below.
c
c
c           (1) the kbdy=1 boundary
c
c           this is the (y,z) plane x=xa where nxa=iparm(2) =
2 flags
c           a mixed boundary condition of the form
c
c           dp/dx + alfxa(y,z)*p(xa,y,z) = gbdxa(y,z)
c

```

```

c      in this case kbdy=1,xory=y,yorz=z will be input to
bndyc and
c      alfa,gbdy corresponding to alfxa(y,z),gbdxa(y,z)
must be returned.
c
c
c      (2) the kbdy=2 boundary
c
c      this is the (y,z) plane x=xb where nxb=iparm(3) =
2 flags
c      a mixed boundary condition of the form
c
c          dp/dx + alfxb(y,z)*p(xb,y,z) = gbdxb(y,z)
c
c      in this case kbdy=2,xory=y,yorz=z will be input to
bndyc and
c      alfa,gbdy corresponding to alfxb(y,z),gbdxb(y,z)
must be returned.
c
c
c      (3) the kbdy=3 boundary
c
c      this is the (x,z) plane y=yc where nyc=iparm(4) =
2 flags
c      a mixed boundary condition of the form
c
c          dp/dy + alfyc(x,z)*p(x,yc,z) = gbdyc(x,z)
c
c      in this case kbdy=3,xory=x,yorz=z will be input to
bndyc and
c      alfa,gbdy corresponding to alfyc(x,z),gbdyc(x,z)
must be returned.
c
c
c      (4) the kbdy=4 boundary
c
c      this is the (x,z) plane y=yd where nyd=iparm(5) =
2 flags
c      a mixed boundary condition of the form
c
c          dp/dy + alfyd(x,z)*p(x,yd,z) = gbdyd(x,z)
c
c      in this case kbdy=4,xory=x,yorz=z will be input to
bndyc and

```

```

c      alfa,gbdy corresponding to alfyd(x,z),gbdyd(x,z)
must be returned.

c
c
c      (5) the kbdy=5 boundary
c
c      this is the (x,y) plane z=ze where nze=iparm(6) =
2 flags
c      a mixed boundary condition of the form
c
c      dp/dz + alfze(x,y)*p(x,y,ze) = gbdze(x,y)
c
c      in this case kbdy=5,xory=x,yorz=y will be input to
bndyc and
c      alfa,gbdy corresponding to alfze(x,y),gbdze(x,y)
must be returned.

c
c
c      (6) the kbdy=6 boundary
c
c      this is the (x,y) plane z=zf where nzf=iparm(7) =
2 flags
c      a mixed boundary condition of the form
c
c      dp/dz + alfzf(x,y)*p(x,y,zf) = gbdzf(x,y)
c
c      in this case kbdy=6,xory=x,yorz=y will be input to
bndyc and
c      alfa,gbdy corresponding to alfzf(x,y),gbdzf(x,y)
must be returned.

c
c
c *** alfxa,alfyc,alfze nonpositive and
alfxb,alfyd,alfze nonnegative
c      will help maintain matrix diagonal dominance
during discretization
c      aiding convergence.

c
c *** bndyc must provide the mixed boundary condition
c      values in correspondence with those flagged in
iparm(2)
c      thru iparm(7). if all boundaries are specified
then
c      cuh3 will never call bndyc. even then it must be

```

```

entered
c      as a dummy subroutine. bndyc must be declared
c      external in the routine calling cuh3. the actual
c      name chosen may be different.
c
c
c ... coef
c
c      a subroutine with arguments
(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
c      which provides the known complex coefficients
for the elliptic pde
c      at any grid point (x,y,z). the name chosen in
the calling routine
c      may be different where the coefficient routine
must be declared
c      external.
c
c ... rhs
c
c      a complex array dimensioned nx by ny by nz
which contains
c      the given right hand side values on the
uniform 3-d mesh.
c      rhs(i,j,k) = r(xi,yj,zk) for i=1,...,nx and
j=1,...,ny
c      and k=1,...,nz.
c
c ... phi
c
c      a complex array dimensioned nx by ny by nz .
on input phi must
c      contain specified boundary values and an
initial guess
c      to the solution if flagged (see
iguess=iparm(17)=1). for
c      example, if nyd=iparm(5)=1 then phi(i,ny,k)
must be set
c      equal to p(xi,yd,zk) for i=1,...,nx and
k=1,...,nz prior to
c      calling cuh3. the specified values are
preserved by cuh3.
c
c ***      if no initial guess is given (iguess=0) then

```

```
phi must still
c           be initialized at non-Dirchlet grid points
(this is not
c           checked). these values are projected down and
serve as an initial
c           guess to the pde at the coarsest grid level.
set phi to 0.0 at
c           nonDirchlet grid points if nothing better is
available.

c
c
c ... mgopt
c
c           an integer vector of length 4 which allows
the user to select
c           among various multigrid options. if
mgopt(1)=0 is input then
c           a default set of multigrid arguments (chosen
for robustness)
c           will be internally selected and the
remaining values in mgopt
c           will be ignored. if mgopt(1) is nonzero
then the arguments
c           in mgopt are set internally and defined as
follows: (see the
c           basic coarse grid correction algorithm
below)

c
c
c     kcycle = mgopt(1)
c
c           = 0 if default multigrid options are to be
used
c
c           = 1 if v cycling is to be used (the least
expensive per cycle)
c
c           = 2 if w cycling is to be used (the
default)
c
c           > 2 if more general k cycling is to be used
c           *** warning--values larger than 2 increase
c           the execution time per cycle
considerably and
```

```
c           result in the nonfatal error ierror =
-5
c           which indicates inefficient multigrid
c cycling.
c
c     iprер = mgopt(2)
c
c           the number of "pre-relaxation" sweeps
c executed before the
c           residual is restricted and cycling is
c invoked at the next
c           coarser grid level (default value is 2
c whenever mgopt(1)=0)
c
c     ipost = mgopt(3)
c
c           the number of "post relaxation" sweeps
c executed after cycling
c           has been invoked at the next coarser grid
c level and the residual
c           correction has been transferred back
c (default value is 1
c           whenever mgopt(1)=0).
c
c *** if iprер, ipost, or (especially) kcycle is greater
c than 2
c           than inefficient multigrid cycling has probably
c been chosen and
c           the nonfatal error (see below) ierror = -5 will be
c set. note
c           this warning may be overridden by any other
c nonzero value
c           for ierror.
c
c     interpol = mgopt(4)
c
c           = 1 if multilinear prolongation
c (interpolation) is used to
c           transfer residual corrections and the pde
c approximation
c           from coarse to fine grids within full
c multigrid cycling.
c
c           = 3 if multicubic prolongation
```

```

(interpolation) is used to
c           transfer residual corrections and the pde
approximation
c           from coarse to fine grids within full
multigrid cycling.
c           (this is the default value whenever
mgopt(1)=0).
c
c *** the default values (2,2,1,3) in the vector mgopt
were chosen for
c   robustness. in some cases v(2,1) cycles with
linear prolongation will
c   give good results with less computation
(especially in two-dimensions).
c   this was the default and only choice in an
earlier version of mudpack
c   (see [1]) and can be set with the integer vector
(1,2,1,1) in mgopt.
c
c *** the schedules for one full multigrid cycle
(iguess=0) using v(2,1)
c   cycles and w(2,1) cycles are depicted for a four
level grid below.
c   the number of relaxation sweeps when each grid is
visited are indicated.
c   the "*" stands for prolongation of the full
approximation and the "."
c   stands for transfer of residuals and residual
corrections within the
c   coarse grid correction algorithm (see below). the
"d" at level 1
c   indicates a direct method is used
c
c   one fmg with v(2,1) cycles:
c
c
c   -----
-----2-----1-
-----      level 4
c           *
c           .
c           *
c           .
c   -----
-----2-----1-----2-----1-----
-----      level 3
c           *
c           .
c           *
c           .

```

```

c      -----2-----1-----2-----1-----2-----1-----
-----      level 2
c      * . .
c      * . .
c      ---d---d-----d-----d-----
-----      level 1
c
c
c      one fmg with w(2,1) cycles:
c
c      -----
--1--      level 4
c                  *
.
c      -----2-----1-----2-----3-----
1----      level 3
c                  *
c      -----2-----1-----2-----3-----1-----
c      -----2-----1-----2-----3-----1-----
-----      level 2
c      * . . . . . . . . . . . .
c      --d---d-----d---d-----d---d-----d---d-----
-----      level 1
c
c
c      the form of the "recursive" coarse grid correction
cycling used
c      when kcyle.ge.0 is input is described below in
pseudo-algorithmic
c      language. it is implemented non-recursively in
fortran in mudpack.
c *** this algorithm is modified with the hybrid
solvers which use
c      a direct method whenever grid level 1 is
encountered.
c
c      algorithm
cgc(k,l(k),u(k),r(k),kcycle,iprer,ipost,iresw,intpol)
c
c *** approximately solve l(k)*u(k) = r(k) using
multigrid iteration
c *** k is the current grid level
c *** l(k) is the discretized pde operator at level k
c *** u(k) is the initial guess at level k
c *** r(k) is the right hand side at level k

```

```

c *** i(k,k-1) is the restriction operator from level k
to level k-1
c *** (the form of i(k,k-1) depends on iresw)
c *** i(k-1,k) is the prolongation operator from level
k-1 to level k
c *** (the form of i(k-1,k) depends on interpol)
c
c      begin algorithm cgc
c
c ***     pre-relax at level k
c
c      . do (i=1,iprer)
c
c      . . relax(l(k),u(k),r(k))
c
c      . end do
c
c      . if (k > 1) then
c
c ***      restrict the residual from level k to level k-
c
c      . . r(k-1) = i(k,k-1) (r(k)-l(k)*u(k))
c
c      . . kount = 0
c
c      . . repeat
c
c ***      solve for the residual correction at level k-1
in u(k-1)
c ***      using algorithm cgc "kcycle" times (this is
the recursion)
c
c      . . . kount = kount+1
c
c      . . . invoke cgc(k-1,l(k-1),u(k-1),r(k-
c      1),kcycle,iprer,ipost,iresw)
c
c
c      . . until (kount.eq.kcycle)
c
c ***      transfer residual correction in u(k-1) to
level k
c ***      with the prolongation operator and add to u(k)

```

```

C
C      . . u(k) = u(k) + i(k-1,k) (u(k-1))
C
C      . end if
C
C *** post relax at level k
C
C      . do (i=1,ipost)
C
C      . . relax(l(k),u(k),r(k))
C
C      . end do
C
C      . return
C
C      end algorithm cgc
C
C
C ****output
arguments*****
***

C
*****
C ... iparm(22)
C
C      on output iparm(22) contains the actual work
space length
C      required for the current grid sizes and
method. this value
C      will be computed and returned even if
iparm(21) is less then
C      iparm(22) (see ierror=9).
C
C
C ... iparm(23)
C
C      if error control is selected (tolmax =
fparm(7) .gt. 0.0) then
C      on output iparm(23) contains the actual

```

```

number of cycles executed
c           between the coarsest and finest grid levels
in obtaining the
c           approximation in phi.  the quantity
(iprter+ipost)*iparm(23) is
c           the number of relaxation sweeps performed at
the finest grid level.

c
c
c ... fparm(8)
c
c           on output fparm(8) contains the final
computed maximum relative
c           difference between the last two iterates at
the finest grid level.
c           fparm(8) is computed only if there is error
control (tolmax.gt.0.)
c           assume phi1(i,j,k) and phi2(i,j,k) are the
last two computed
c           values for phi(i,j,k) at all points of the
finest grid level.
c           if we define
c
c           phdif = max(abs(phi2(i,j,k)-phi1(i,j,k)))
for all i,j,k
c
c           and
c
c           phmax = max(abs(phi2(i,j,k))) for all
i,j,k
c
c           then
c
c           fparm(8) = phdif/phmax
c
c           is returned whenever phmax.gt.0.0.  in the
degenerate case
c           phmax = 0.0, fparm(8) = phdif is returned.
c
c
c ... wk
c
c           on output wk contains intermediate values

```

```

that must not be
c           destroyed if cuh3 is to be called again with
iparm(1)=1 or
c           if cuh34 is to be called to improve the
estimate to fourth
c           order.
c
c ... phi
c
c           on output phi(i,j,k) contains the
approximation to
c           p(xi,yj,zk) for all mesh points i=1,...,nx;
j=1,...,ny;
c           k=1,...,nz. the last computed iterate in phi
is returned
c           even if convergence is not obtained (ierror=-
1)
c
c ... ierror
c
c           for intl=iparm(1)=0 initialization calls,
ierror is an
c           error flag that indicates invalid input
arguments when
c           returned positive and nonfatal warnings when
returned
c           negative. argument checking and
discretization
c           is bypassed for intl=1 calls which can only
return
c           ierror = -1 or 0 or 1.
c
c
c     non-fatal warnings * * *
c
c
c     ==5 if kcycle=mgopt(1) is greater than 2. values
larger than 2 results
c           in an algorithm which probably does far more
computation than
c           necessary. kcycle = 1 (v cycles) or kcycle=2
(w cycles) should
c           suffice for most problems. ierror = -5 is
also set if either

```

```

c           iprer = mgopt(2) or ipost=mgopt(3) is greater
than 2.  the
c           ierror=-5 flag is overridden by any other
fatal or non-fatal
c           error.
c
c           ==4 if there are dominant nonzero first order
terms in the pde which
c           make it "hyperbolic" at the finest grid level.
numerically, this
c           happens if:
c
c           abs(cx)*dlx > 2.*abs(cxx)      (dlx = (xb-
xa) / (nx-1))
c
c                           (or)
c
c           abs(cy)*dly > 2.*abs(cyy)      (dly = (yd-
yc) / (ny-1))
c
c
c           at some fine grid point (xi,yj).  if an
adjustment is not made the
c           condition can lead to a matrix coming from the
discretization
c           which is not diagonally dominant and
divergence is possible. since
c           the condition is "likely" at coarser grid
levels for pde's with
c           nonzero first order terms, the adjustments
(actually first order
c           approximations to the pde)
c
c
c           cxx = amax1(cxx,0.5*abs(cx)*dx)
c
c                           (and)
c
c           cyy = amax1(cyy,0.5*abs(cy)*dy)
c
c
c           (here dx,dy are the x,y mesh sizes of the
subgrid)
c

```

```

c      are made to preserve convergence of multigrid
iteration. if made
c      at the finest grid level, it can lead to
convergence to an
c      erroneous solution (flagged by ierror = -4).
a possible remedy
c      is to increase resolution. the ierror = -4
flag overrides the
c      nonfatal ierror = -5 flag.

c
c
c      ==3 if the continuous elliptic pde is singular.
this means the
c      boundary conditions are periodic or pure
derivative at all
c      boundaries and ce(x,y) = 0.0 for all x,y. a
solution is still
c      attempted but convergence may not occur due
to ill-conditioning
c      of the linear system coming from the
discretization. the
c      ierror = -3 flag overrides the ierror=-4 and
ierror=-5 nonfatal
c      flags.

c
c
c      ==2 if the pde is not elliptic (i.e.,
cxx*cyy.le.0.0 for some (xi,yj))
c      in this case a solution is still attempted
although convergence
c      may not occur due to ill-conditioning of the
linear system.
c      the ierror = -2 flag overrides the ierror=-
5,-4,-3 nonfatal
c      flags.

c
c
c      ==1 if convergence to the tolerance specified in
tolmax=fparm(5)>0.
c      is not obtained in maxcy=iparm(13) multigrid
cycles between the
c      coarsest (ixp+1,jyq+1) and finest (nx,ny)
grid levels.
c      in this case the last computed iterate is

```

```
still returned.  
C           the ierror = -1 flag overrides all other  
nonfatal flags  
C  
C  
C       no errors * * *  
C  
C       = 0  
C  
C       fatal argument errors * * *  
C  
C       = 1 if intl=iparm(1) is not 0 on initial call or  
not 0 or 1  
C           on subsequent calls  
C  
C       = 2 if any of the boundary condition flags  
nxa,nxb,nyc,nyd,nze,nzf  
C           in iparm(2) through iparm(7) is not 0,1 or 2 or  
if  
C           (nxa,nxb) or (nyc,nyd) or (nze,nzf) are not  
pairwise zero.  
C  
C       = 3 if mino(ixp,jyq,kzr) < 2  
(ixp=iparm(8),jyq=iparm(9),kzr=iparm(10))  
C           or if ixp<3 when nxa=0 or jyq<3 when nyc=0 or  
kzr<3 when nze=0.  
C  
C       = 4 if min0(iex,jey,kez) < 1  
(iex=iparm(11),jey=iparm(12),kez=iparm(13))  
C           or if max0(iex,jey,kez) > 50  
C  
C       = 5 if nx.ne.ixp*2** (iex-1)+1 or if  
ny.ne.jyq*2** (jey-1)+1 or  
C           if nz.ne.kzr*2** (kez-1)+1  
(nx=iparm(14),ny=iparm(15),nz=iparm(16))  
C  
C       = 6 if iguess = iparm(17) is not equal to 0 or 1  
C  
C       = 7 if maxcy = iparm(18) < 1 (large values for  
maxcy should not be used)  
C  
C       = 8 if method = iparm(19) is less than 0 or  
greater than 10  
C
```

```

c      = 9 if length = iparm(20) is too small (see
iparm(21) on output
c          for minimum required work space length)
c
c      =10 if xa > xb or yc > yd or ze > zf
c
(xa=fparm(1),xb=fparm(2),yc=fparm(3),yd=fparm(4),ze=fpar
m(5),zf=fparm(6))
c
c      =11 if tolmax = fparm(7) < 0.0
c
c      errors in setting multigrid options * * * (see
also ierror=-5)
c
c      =12 if kcycle = mgopt(1) < 0 or
c          if iprer = mgopt(2) < 1 or
c          if ipost = mgopt(3) < 1 or
c          if interpol = mgopt(4) is not 1 or 3
c
c      =13 if iex=jey=kez=1 (full direct method) and
iguess=1 or maxcy > 1
c
c      =14 if the elliptic pde is singular (see ierror=-3
in cud3.d)
c
c
c
*****
*
c
*****
*
c
*****
end of cuh3 documentation
c
c
*****
**
c
*****
**
c
c

```

---

## CUH34

```
C
C      file cuh34.d
C
C      * * * * * * * * * * * * * * * *
* * * * * * *
C      *
*
C      *           copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *           all rights reserved
*
C      *
*
C      *           MUDPACK version 5.0.1
*
C      *
*
C      *           A Fortran Package of Multigrid
*
C      *
*
C      *           Subroutines and Example Programs
*
C      *
*
C      *           for Solving Elliptic Partial Differential
Equations          *
C      *
*
C      *           by
*
C      *
```

```
*  
C      *                                John Adams  
*  
C      *  
*  
C      *                                of  
*  
C      *  
*  
C      *          the National Center for Atmospheric  
Research           *  
C      *  
*  
C      *          Boulder, Colorado (80307)  
U.S.A.           *  
C      *  
*  
C      *          which is sponsored by  
*  
C      *  
*  
C      *          the National Science Foundation  
*  
C      *  
*  
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C  
C ... file cuh34.d  
C  
C     contains documentation for:  
C     subroutine cuh34(wk,iwk,phi,ierror)  
C     A sample fortran driver is file "tcuh34.f".  
C  
C ... required MUDPACK files  
C  
C     cuh3.f, cudcom.f, cud3ln.f, cud3pn.f  
C  
C ... purpose  
C  
C     cuh34 attempts to improve the estimate in phi,  
obtained by calling  
C     cuh3, from second to fourth order accuracy. see  
the file "cuh3.d"
```

```
c      for a detailed discussion of the elliptic pde
approximated and
c      arguments "wk,iwk,phi" which are also part of the
argument list for
c      cuh3.
c
c ... assumptions
c
c      * phi contains a second-order approximation from
an earlier cuh3 call
c
c      * arguments "wk,iwk,phi" are the same used in
calling cuh3
c
c      * "wk,iwk,phi" have not changed since the last
call to cuh3
c
c      * the finest grid level contains at least 6
points in each direction
c
c
c *** warning
c
c      if the first assumption is not true then a fourth
order approximation
c      cannot be produced in phi. the last assumption
(adequate grid size)
c      is the only one checked. failure in any of the
others can result in
c      in an undetectable error.
c
c ... language
c
c      fortran90/fortran77
c
c ... portability
c
c      mudpack5.0.1 software has been compiled and tested
with fortran77
c      and fortran90 on a variety of platforms.
c
c ... methods
c
c      details of the methods employed by the solvers in
```

```
mudpack are given
c      in [1,9]. [1,2,9] contain performance
measurements on a variety of
c      elliptic pdes (see "references" in the file
"readme"). in summary:
c
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
estimate (i.e.,
c      discretization level error is reached) then this
can be improved to a
c      fourth-order estimate using the technique of
"deferred corrections"
c      the values in the solution array are used to
generate a fourth-order
c      approximation to the truncation error. second-
order finite difference
c      formula are used to approximate third and fourth
partial derivatives
c      of the solution function [3]. the truncation
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side. the default
multigrid options
c      are used to compute the fourth-order correction
term which is added
c      to the original solution array.
c
c
c ... references (partial)
c
c
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
c      Solution of Linear Elliptic Partial Differential
Equations,"
c      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
c
c      [2] J. Adams, "FMG Results with the Multigrid
```

```
Software Package MUDPACK,"  
c      proceedings of the fourth Copper Mountain  
Conference on Multigrid, SIAM,  
c      1989, pp.1-12.  
c      .  
c      .  
c      .  
c      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.  
Haidvogel, and V. Pizzo,  
c      "Applications of Multigrid Software in the  
Atmospheric Sciences,"  
c      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-  
1458.  
c      .  
c      .  
c      .  
c      [9] J. Adams, "Recent Enhancements in MUDPACK, a  
Multigrid Software  
c      package for Elliptic Partial Differential  
Equations," Applied Math.  
c      and Comp., 1991, vol. 43, May 1991, pp. 79-94.  
c  
c      [10]J. Adams, "MUDPACK-2: Multigrid Software for  
Approximating  
c      Elliptic Partial Differential Equations on Uniform  
Grids with  
c      any Resolution," Applied Math. and Comp., 1993,  
vol. 53, February  
c      1993, pp. 235-249  
c      .  
c      .  
c      .  
c  
c ... error parameter  
c  
c      = 0 if no error is detected  
c  
c      = 30 if min0(nx,ny,nz) < 6 where nx,ny,nz are the  
fine grid sizes  
c              in the x,y,z directions.  
c  
c  
c  
*****
```

```
*****
C
*****
C      end of cuh34 documentation
C
C
*****
C
*****
C
```

---

## MUD2

```
C
C      file mud2.d
C
C      * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK version 5.0.1
*
```



```

c
c ... file mud2.d
c
c      contains documentation for:
c      subroutine
mud2(iparm,fparm,work,coef,bndyc,rhs,phi,mgopt,ierror)
c      A sample fortran driver is file "tmud2.f".
c
c ... required MUDPACK files
c
c      mudcom.f
c
c ... purpose
c
c      subroutine mud2 automatically discretizes and
attempts to compute
c      the second-order difference approximation to the
two-dimensional
c      linear nonseparable elliptic partial differential
equation on a
c      rectangle. the approximation is generated on a
uniform grid covering
c      the rectangle (see mesh description below).
boundary conditions
c      may be specified (dirchlet), periodic, or mixed
derivative in any
c      combination. the form of the pde solved is:
c
c
c      cxx(x,y)*pxx + cyy(x,y)*pyy + cx(x,y)*px +
cy(x,y)*py +
c
c      ce(x,y)*p(x,y) = r(x,y).
c
c
c      pxx,pyy,px,py are second and first partial
derivatives of the
c      unknown real solution function p(x,y) with respect
to the
c      independent variables x,y. cxx,cyy,cx,cy,ce are
the known
c      real coefficients of the elliptic pde and r(x,y)
is the known
c      real right hand side of the equation. cxx and cyy

```

```
should be
c      positive for all x,y in the solution region.
Nonseparability
c      means some of the coefficients depend on both x
and y.  If
c      the PDE is separable subroutine mud2sp should be
used instead
c      of mud2 or muh2.
c
c
c ... mesh description . . .
c
c      the approximation is generated on a uniform nx by
ny grid.  the grid
c      is superimposed on the rectangular solution region
c
c      [xa,xb] x [yc,yd].
c
c      let
c
c      dlx = (xb-xa) / (nx-1),  dly = (yd-yc) / (ny-1)
c
c      be the uniform grid increments in the x,y
directions. then
c
c      xi=xa+(i-1)*dlx,    yj=yc+(j-1)*dly
c
c      for i=1,...,nx and j=1,...,ny  denote the x,y
uniform mesh points
c
c
c ... language
c
c      fortran90/fortran77
c
c
c ... portability
c
c      mudpack5.0.1 software has been compiled and tested
with Fortran77
c      and Fortran90 on a variety of platforms.
c
c ... methods
c
```

```
c      details of the methods employed by the solvers in
mudpack are given
c      in [1,9]. [1,2,9] contain performance
measurements on a variety of
c      elliptic pdes (see "references" in the file
"readme"). in summary:
c
c *** discretization and solution (second-order solvers)
(see [1])
c
c      the pde and boundary conditions are automatically
discretized at all
c      grid levels using second-order finite difference
formula. diagonal
c      dominance at coarser grid levels is maintained in
the presence of
c      nonzero first-order terms by adjusting the second-
order coefficient
c      when necessary. the resulting block tri-diagonal
linear system is
c      approximated using multigrid iteration
[10,11,13,15,16,18]. version
c      5.0.1 of mudpack uses only fully weighted residual
restriction. defaults
c      include cubic prolongation and w(2,1) cycles.
these can be overridden
c      with selected multigrid options (see "mgopt").
error control based on
c      maximum relative differences is available. full
multigrid cycling (fmg)
c      or cycling beginning or restarting at the finest
grid level can be
c      selected. a menu of relaxation methods including
gauss-seidel point,
c      line relaxation(s) (in any combination of
directions) and planar
c      relaxation (for three-dimensional anisotropic
problems) are provided.
c      all methods use ordering based on alternating
points (red/black),
c      lines, or planes for cray vectorization and
improved convergence
c      rates [14].
c
```

```
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
estimate (i.e.,
c      discretization level error is reached) then this
can be improved to a
c      fourth-order estimate using the technique of
"deferred corrections."
c      the values in the solution array are used to
generate a fourth-order
c      approximation to the truncation error. second-
order finite difference
c      formula are used to approximate third and fourth
partial derivatives
c      of the solution function [3]. the truncation
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side. the default
multigrid options
c      are used to compute the fourth-order correction
term which is added
c      to the original solution array.
c
c
c ... references (partial)
c
c
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
c      Solution of Linear Elliptic Partial Differential
Equations,"
c      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
c
c      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
c      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
c      1989, pp.1-12.
c      .
c      .
c      .
```

```
c      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.  
Haidvogel, and V. Pizzo,  
c      "Applications of Multigrid Software in the  
Atmospheric Sciences,"  
c      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-  
1458.  
c      .  
c      .  
c      .  
c      [9] J. Adams, "Recent Enhancements in MUDPACK, a  
Multigrid Software  
c      package for Elliptic Partial Differential  
Equations," Applied Math.  
c      and Comp., 1991, vol. 43, May 1991, pp. 79-94.  
c  
c      [10]J. Adams, "MUDPACK-2: Multigrid Software for  
Approximating  
c      Elliptic Partial Differential Equations on Uniform  
Grids with  
c      any Resolution," Applied Math. and Comp., 1993,  
vol. 53, February  
c      1993, pp. 235-249  
c      .  
c      .  
c      .  
c  
c ... argument description  
c  
c  
c  
*****  
*****  
c *** input arguments  
*****  
c  
*****  
*****  
c  
c  
c ... iparm  
c  
c      an integer vector of length 17 used to pass  
integer  
c      arguments. iparm is set internally and
```

```
defined as
c           follows:
c
c
c ... intl=iparm(1)
c
c           an initialization argument. intl=0 must be
input
c           on an initial call. in this case input
arguments will
c           be checked for errors and the elliptic
partial differential
c           equation and boundary conditions will be
discretized using
c           second order finite difference formula.
c
c ***      An approximation is NOT generated after an
intl=0 call!
c           mud2 should be called with intl=1 to
approximate the elliptic
c           PDE discretized by the intl=0 call. intl=1
should also
c           be input if mud2 has been called earlier and
only the
c           values in in rhs (see below) or gbdy (see
bndyc below)
c           or phi (see below) have changed. This will
bypass
c           redundant pde discretization and argument
checking
c           and save computational time. Some examples
of when
c           intl=1 calls should be used are:
c
c           (0) after a intl=0 argument checking and
discretization call
c
c           (1) mud2 is being recalled for additional
accuracy. In
c           this case iguess=iparm(12)=1 should also
be used.
c
c           (2) mud2 is being called every time step in a
time dependent
```

```
c           problem (see discussion below) where the
elliptic operator
c           does not depend on time.
c
c           (3) mud2 is being used to solve the same
elliptic equation
c           for several different right hand sides
(iguess=0 should
c           probably be used for each new righthand
side).
c
c           intl = 0 must be input before calling with
intl = 1 when any
c           of the following conditions hold:
c
c           (a) the initial call to mud2
c
c           (b) any of the integer arguments other than
iguess=iparm(12)
c           or maxcy=iparm(13) or mgopt have changed
since the previous
c           call.
c
c           (c) any of the floating point arguments other
than tolmax=
c           fparm(5) have changed since the previous
call
c
c           (d) any of the coefficients input by coef
(see below) have
c           changed since the previous call
c
c           (e) any of the "alfa" coefficients input by
bndyc (see below)
c           have changed since the previous call.
c
c           If any of (a) through (e) are true then the
elliptic PDE
c           must be discretized or rediscretized. If
none of (a)
c           through (e) holds, calls can be made with
intl=1.
c           Incorrect calls with intl=1 will produce
erroneous results.
```

```

c ***      The values set in the saved work space "work"
(see below) with
c           an intl=0 call must be preserved with
subsequent intl=1 calls.
c
c           MUDPACK software performance should be
monitored for intl=1
c           calls.  The intl=0 discretization call
performance depends
c           primarily on the efficiency or lack of
efficiency of the
c           user provided subroutines for pde
coefficients and
c           boundary conditions.
c
c ... nxa=iparm(2)
c
c           flags boundary conditions on the edge x=xa
c
c           = 0 if p(x,y) is periodic in x on [xa,xb]
c               (i.e., p(x+xb-xa,y) = p(x,y) for all x,y
c               (if nxa=0 then nxb=0 is required, see
ierror = 2)
c
c           = 1 if p(xa,y) is specified (this must be input
thru phi(1,j))
c
c           = 2 if there are mixed derivative boundary
conditions at x=xa
c               (see bndyc)
c
c
c ... nxb=iparm(3)
c
c           flags boundary conditions on the edge x=xb
c
c           = 0 if p(x,y) is periodic in x on [xa,xb]
c               (i.e., p(x+xb-xa,y) = p(x,y) for all x,y
c               (if nxb=0 then nxa=0 is required, see
ierror = 2)
c
c           = 1 if p(xb,y) is specified (this must be input
thru phi(nx,j))
c

```

```

c      = 2 if there are mixed derivative boundary
conditions at x=xb
c          (see bndyc)
c
c
c ... nyc=iparm(4)
c
c      flags boundary conditions on the edge y=yc
c
c      = 0 if p(x,y) is periodic in y on [yc,yd]
c          (i.e., p(x,y+yd-yc) = p(x,y) for all x,y
c          (if nyc=0 then nyd=0 is required, see
ierror = 2)
c
c      = 1 if p(x,yc) is specified (this must be input
thru phi(i,1))
c
c      = 2 if there are mixed derivative boundary
conditions at y=yc
c          (see bndyc)
c
c
c ... nyd=iparm(5)
c
c      flags boundary conditions on the edge y=yd
c
c      = 0 if p(x,y) is periodic in y on [yc,yd]
c          (i.e., p(x,y+yd-yc) = p(x,y) for all x,y
c          (if nyd=0 then nyc=0 is required, see
ierror = 2)
c
c      = 1 if p(x,yd) is specified (this must be input
thru phi(i,ny))
c
c      = 2 if there are mixed derivative boundary
conditions at y=yd
c          (see bndyc)
c
c
c *** grid size arguments
c
c
c ... ixp = iparm(6)
c

```

```
c      an integer greater than one which is used in
defining the number
c      of grid points in the x direction (see nx =
iparm(10)). "ixp+1"
c      is the number of points on the coarsest x
grid visited during
c      multigrid cycling. ixp should be chosen as
small as possible.
c      recommended values are the small primes 2 or
3.
c      larger values can reduce multigrid
convergence rates considerably,
c      especially if line relaxation in the x
direction is not used.
c      if ixp > 2 then it should be 2 or a small odd
value since a power
c      of 2 factor of ixp can be removed by
increasing iex = iparm(8)
c      without changing nx = iparm(10).

c
c
c ... jyq = iparm(7)
c
c      an integer greater than one which is used in
defining the number
c      of grid points in the y direction (see ny =
iparm(11)). "jyq+1"
c      is the number of points on the coarsest y
grid visited during
c      multigrid cycling. jyq should be chosen as
small as possible.
c      recommended values are the small primes 2 or
3.
c      larger values can reduce multigrid
convergence rates considerably,
c      especially if line relaxation in the y
direction is not used.
c      if jyq > 2 then it should be 2 or a small odd
value since a power
c      of 2 factor of jyq can be removed by
increasing jey = iparm(9)
c      without changing ny = iparm(11).

c
```

```

c ... iex = iparm(8)
c
c           a positive integer exponent of 2 used in
defining the number
c           of grid points in the x direction (see nx =
iparm(10)).
c           iex .le. 50 is required. for efficient
multigrid cycling,
c           iex should be chosen as large as possible and
ixp=iparm(8)
c           as small as possible within grid size
constraints when
c           defining nx.

c
c
c ... jey = iparm(9)
c
c           a positive integer exponent of 2 used in
defining the number
c           of grid points in the y direction (see ny =
iparm(11)).
c           jey .le. 50 is required. for efficient
multigrid cycling,
c           jey should be chosen as large as possible and
jyq=iparm(7)
c           as small as possible within grid size
constraints when
c           defining ny.

c
c
c
c ... nx = iparm(10)
c
c           the number of equally spaced grid points in
the interval [xa,xb]
c           (including the boundaries). nx must have the
form
c
c           nx = ixp*(2** (iex-1)) + 1
c
c           where ixp = iparm(6), iex = iparm(8).
c
c
c ... ny = iparm(11)

```

```

c
c      the number of equally spaced grid points in
the interval [yc,yd]
c      (including the boundaries). ny must have the
form:
c
c      ny = jyq* (2** (jey-1)) + 1
c
c      where jyq = iparm(7), jey = iparm(9).
c
c
c *** example
c
c      suppose a solution is wanted on a 33 by 97
grid. then
c      ixp=2, jyq=6 and iex=jey=5 could be used. a
better
c      choice would be ixp=2, jyq=3, and iex=5,
jey=6.
c
c *** grid size flexibility considerations:
c
c      the hybrid multigrid/direct method code muh2
provides more grid size
c      flexibility than mud2 by removing the constraint
that ixp and jyq are
c      2 or 3. This is accomplished by using a direct
method whenever the
c      coarsest (ixp+1) X (jyq+1) grid is encountered in
multigrid cycling.
c      If nx = ixp+1 and ny = jyq+1 then muh2 becomes a
full direct method.
c      muh2 is roughly equivalent to mud2 in efficiency
as long as ixp and
c      jyq remain "small" (see muh2.d). If the problem
to be approximated
c      requires a grid neither mud2 por muh2 can exactly
fit then another option
c      is to generate an approximation on a "close grid"
using mud2 or muh2.
c      Then transfer the result to the required grid
using cubic interpolation
c      via the package "regridpack" (contact John Adams
about this software)

```

```

C
C *** note
C
C      let G be the nx by ny fine grid on which the
C      approximation is
C      generated and let n = max0(iex,jey).  in mudpack,
C      multigrid
C      cycling is implemented on the ascending chain of
C      grids
C
C          G(1) < ... < G(k) < ... < G(n) = G.
C
C      each G(k) (k=1,...,n) has mx(k) by my(k) grid
C      points
C      given by:
C
C          mx(k) = ixp*[2**max0(iex+k-n,1)-1] + 1
C
C          my(k) = jyq*[2**max0(jey+k-n,1)-1] + 1
C
C
C
C ... iguess=iparm(12)
C
C          = 0 if no initial guess to the pde is
C          provided
C
C          = 1 if an initial guess to the pde is at the
C          finest grid
C          level is provided in phi (see below)
C
C      comments on iguess = 0 or 1 . . .
C
C      even if iguess = 0, phi must be initialized at all
C      grid points (this
C      is not checked).  phi can be set to 0.0 at non-
C      dirchlet grid points
C      if nothing better is available.  the values set in
C      phi when iguess = 0
C      are passed down and serve as an initial guess to
C      the pde at the coarsest
C      grid level where cycling commences.  in this
C      sense, values input in
C      phi always serve as an initial guess.  setting

```

```

iguess = 0 forces full
c      multigrid cycling beginning at the coarsest and
finishing at the finest
c      grid level.
c
c      if iguess = 1 then the values input in phi are an
initial guess to the
c      pde at the finest grid level where cycling begins.
this option should
c      be used only if a "very good" initial guess is
available (as, for
c      example, when restarting from a previous iguess=0
call).
c
c      time dependent problems . . .
c
c *** assume we are solving an elliptic pde every time
step in a
c      marching problem of the form:
c
c      l(p(t)) = r(t)
c
c      where the differential operator "l" has no time
dependence,
c      "p(t)" is the solution and "r(t)" is the right
hand side at
c      current time "t". let "dt" be the increment
between time steps.
c      then p(t) can be used as an initial guess to
p(t+dt) with
c      intl = 1 when solving
c
c      l(p(t+dt)) = r(t+dt) .
c
c      after the first two time steps, rather than
continue, it would
c      be better to define the "correction" term:
c
c      e(t,dt) = p(t+dt) - p(t)
c
c      this clearly satisfies the equation
c
c      l(e(t,dt)) = r(t+dt) - r(t) .
c

```

```

c      this should be solved with iguess = 0 and intl =
1. boundary
c      conditions for e(t,dt) are obtained from the
boundary conditions
c      for p(t) by subtracting given values at t from
given values at
c      t+dt. for example if
c
c           $d(p(t))/dx = f(t)$ ,  $d(p(t+dt))/dx = f(t+dt)$ 
c
c      at some x boundary then e(t,dt) satisfies the
derivative
c      boundary condition
c
c           $d(e(t,dt))/dx = f(t+dt) - f(t)$ .
c
c      e(t,dt) can be preset to 0.0 (at nondirichlet
points) or (if p(t-dt)
c      is saved) to p(t)-p(t-dt). with iguess = 0, these
values will serve
c      as an initial guess to e(t,dt) at the coarsest
grid level. this
c      approach has the advantage that a full sequence of
multigrid cycles,
c      beginning at the coarsest grid level, is invoked
every time step in
c      solving for e(t,dt). a few digits of accuracy in
e(t,dt), which is
c      ordinarily much smaller than p(t), will yield
several more digits of
c      accuracy in the final approximation:
c
c           $p(t+dt) = p(t) + e(t,dt)$ .
c
c      using this approach to integrate in time will give
more accuracy
c      then using p(t) as an initial guess to p(t+dt) for
all time steps.
c      it does require additional storage.
c
c      if the differential operator "l" has time
dependence (either thru
c      the coefficients in the pde or the coefficients in
the derivative

```

```

c      boundary conditions) then use p(t) as an initial
guess to p(t+dt)
c      when solving
c
c      l(t+dt) (p(t+dt)) = r(t+dt)
c
c      with intl = 0 for all time steps (the
discretization must be repeated
c      for each new "t"). either iguess = 0 (p(t) will
then be an initial
c      guess at the coarsest grid level where cycles will
commence) or
c      iguess = 1 (p(t) will then be an initial guess at
the finest grid
c      level where cycles will remain fixed) can be
tried.
c
c
c ... maxcy = iparm(13)
c
c      the exact number of cycles executed between
the finest (nx by
c      ny) and the coarsest ((ixp+1) by (jyq+1))
grid levels when
c      tolmax=fparm(5)=0.0 (no error control). when
tolmax > 0.0
c      is input (error control) then maxcy is a
limit on the number
c      of cycles between the finest and coarsest
grid levels. in
c      any case, at most maxcy*(iprert+ipost)
relaxation sweeps are
c      are performed at the finest grid level (see
iprert=mgopt(2),
c      ipost=mgopt(3) below). when multigrid
iteration is working
c      "correctly" only a few are required for
convergence. large
c      values for maxcy should not be necessary.
c
c
c ... method = iparm(14) determines the method of
relaxation
c      (gauss-seidel based on alternating points

```

```
or lines)
C
C      = 0 for point relaxation
C
C      = 1 for line relaxation in the x direction
C
C      = 2 for line relaxation in the y direction
C
C      = 3 for line relaxation in both the x and y
direction
C
C
C *** choice of method. . .
C
C      let fx represent the quantity cxx(x,y)/dlx**2 over
the solution region.
C
C      let fy represent the quantity cyy(x,y)/dly**2 over
the solution region
C
C      if fx,fy are roughly the same size and do not vary
too much over
C      the solution region choose method = 0. if this
fails try method=3.
C
C      if fx is much greater than fy choose method = 1.
C
C      if fy is much greater than fx choose method = 2
C
C      if neither fx or fy dominates over the solution
region and they
C      both vary considerably choose method = 3.
C
C
C ... length = iparm(15)
C
C      the length of the work space provided in
vector work (see below).
C      let isx = 0 if method = 0 or method = 2
C      let isx = 3 if method = 1 or method = 3 and
nxa.ne.0
C      let isx = 5 if method = 1 or method = 3 and
nxa.eq.0
C      let jsy = 0 if method = 0 or method = 1
```

```
c           let jsy = 3 if method = 2 or method = 3 and
nyc.ne.0
c           let jsy = 5 if method = 2 or method = 3 and
nyc.eq.0
c           then . . .
c
c           length =
4*[nx*ny*(10+isxt+jsy)+8*(nx+ny+2)]/3
c
c           will suffice in most cases. the exact
minimal work space
c           length required for the current nx,ny and
method is output
c           in iparm(16) (even if iparm(15) is too
small). this will be
c           less then the value given by the simplified
formula above
c           in most cases.
c
c
c ... fparm
c
c           a floating point vector of length 6 used to
efficiently
c           pass floating point arguments. fparm is set
internally
c           in mud2 and defined as follows . . .
c
c
c ... xa=fparm(1), xb=fparm(2)
c
c           the range of the x independent variable. xa
must
c           be less than xb
c
c
c ... yc=fparm(3), yd=fparm(4)
c
c           the range of the y independent variable. yc
must
c           be less than yd.
c
c
c ... tolmax = fparm(5)
```

```

c
c           when input positive, tolmax is a maximum
relative error tolerance
c           used to terminate the relaxation iterations.
assume phi1(i,j)
c           and phi2(i,j) are the last two computed
approximations at all
c           grid points of the finest grid level. if we
define
c
c           phdif = max(abs(phi2(i,j)-phi1(i,j))) for
all i,j
c
c           and
c
c           phmax = max(abs(phi2(i,j))) for all i,j
c
c           then "convergence" is considered to have
occurred if and only if
c
c           phdif/phmax < tolmax.
c
c
c           if tolmax=fparm(5)=0.0 is input then there is
no error control
c           and maxcy cycles from the finest grid level
are executed. maxcy
c           is a limit which cannot be exceeded even with
error control.
c           *** calls with tolmax=0.0, when appropriate
because of known
c           convergence behavior, are more efficient than
calls with tolmax
c           positive (i.e., if possible DO NOT use error
control!).
c
c ... work
c
c           a one dimensional real saved work space (see
iparm(15) for
c           length) which must be preserved from the
previous call when
c           calling with intl=iparm(1)=1.
c

```

```

c ... bndyc
c
c           a subroutine with arguments
(kbdy,xory,alfa,gbdy) which
c           are used to input mixed boundary conditions
to mud2. bndyc
c           must be declared "external" in the program
calling mud2.
c           the boundaries are numbered one thru four and
the mixed
c           derivative boundary conditions are described
below (see the
c           sample driver code "tmud2.f" for an example
of how bndyc is
c           can beset up).
c
c           * * * * * * * * * * * * * * * * y=yd
c           *         kbdy=4      *
c           *
c           *
c           *
c           *
c           * kbdy=1      kbdy=2 *
c           *
c           *
c           *
c           *
c           *         kbdy=3      *
c           * * * * * * * * * * * * * * * * y=yc
c
c           x=xa          xb
c
c
c           (1) the kbdy=1 boundary
c
c           this is the edge x=xa where nx=iparm(2)=2
flags
c           a mixed boundary condition of the form
c
c           dp/dx + alfx(x,y)*p(xa,y) = gbdxa(y)
c
c           in this case kbdy=1,xory=y will be input to
bndyc and
c           alfa,gbdy corresponding to alfx(x,y),gbdxa(y)
must be returned.
c

```

```

C
C          (2) the kbdy=2 boundary
C
C          this is the edge x=xb where nxb=iparm(3)=2
flags      a mixed boundary condition of the form
C
C           $\frac{dp}{dx} + \text{alfxb}(y) * p(xb, y) = \text{gbdxb}(y)$ 
C
C          in this case kbdy=2, xory=y, will be input to
bndyc and
C          alfa, gbdy corresponding to alfxb(y), gbdxb(y)
must be returned.

C
C
C          (3) the kbdy=3 boundary
C
C          this is the edge y=yc where nyc=iparm(4)=2
flags      a mixed boundary condition of the form
C
C           $\frac{dp}{dy} + \text{alfyc}(x) * p(x, yc) = \text{gbdyc}(x)$ 
C
C          in this case kbdy=3, xory=x will be input to
bndyc and
C          alfa, gbdy corresponding to alfy(c)(x), gbdyc(x)
must be returned.

C
C
C          (4) the kbdy=4 boundary
C
C          this is the edge y=yd where nyd=iparm(5)=2
flags      a mixed boundary condition of the form
C
C           $\frac{dp}{dy} + \text{alfyd}(x) * p(x, yd) = \text{gbdyd}(x)$ 
C
C          in this case kbdy=4, xory=x will be input to
bndyc and
C          alfa, gbdy corresponding to alfyd(x), gbdyd(x)
must be returned.

C
C
C ***      bndyc must provide the mixed boundary

```

```

condition values
c           in correspondence with those flagged in
iparm(2) thru
c           iparm(5). if all boundaries are specified or
periodic
c           mud2 will never call bndyc. even then it
must be entered
c           as a dummy subroutine. bndyc must be declared
"external"
c           in the routine calling mud2. the actual name
chosen may
c           be different.

c
c
c ... coef
c
c           a subroutine with arguments
(x,y,cxx,cyy,cx,cy,ce) which
c           provides the known real coefficients for the
elliptic pde at
c           any grid point (x,y). the name chosen in the
calling routine
c           may be different where the coefficient routine
must be declared
c           "external."
c
c ... rhs
c
c           an array dimensioned nx by ny which contains
the given
c           right hand side values on the uniform 2-d
mesh.

c
c           rhs(i,j) = r(xi,yj) for i=1,...,nx and
j=1,...,ny
c
c ... phi
c
c           an array dimensioned nx by ny. on input phi
must contain
c           specified boundary values. for example, if
nyd=iparm(5)=1
c           then phi(i,ny) must be set equal to p(xi,yd)
for i=1,...nx

```

```
c           prior to calling mud2.  these values are
preserved by mud2.
c           if an initial guess is provided
(iguess=iparm(11)=1) it must
c           be input thru phi.
c
c
c ***      if no initial guess is given (iguess=0) then
phi must still
c           be initialized at all grid points (this is
not checked).  these
c           values will serve as an initial guess to the
pde at the coarsest
c           grid level after a transfer from the fine
solution grid.  set phi
c           equal to to 0.0 at all internal and non-
specified boundaries
c           grid points if nothing better is available.
c
c
c ... mgopt
c
c           an integer vector of length 4 which allows
the user to select
c           among various multigrid options.  if
mgopt(1)=0 is input then
c           a default set of multigrid arguments (chosen
for robustness)
c           will be internally selected and the
remaining values in mgopt
c           will be ignored.  if mgopt(1) is nonzero
then the arguments
c           in mgopt are set internally and defined as
follows: (see the
c           basic coarse grid correction algorithm
below)
c
c
c       kcycle = mgopt(1)
c
c           = 0 if default multigrid options are to be
used
c
c           = 1 if v cycling is to be used (the least
```

```
expensive per cycle)
c
c           = 2 if w cycling is to be used (the
default)
c
c           > 2 if more general k cycling is to be used
c           *** warning--values larger than 2 increase
c           the execution time per cycle
considerably and
c           result in the nonfatal error ierror =
-5
c           which indicates inefficient multigrid
cycling.
c
c     iprер = mgopt(2)
c
c           the number of "pre-relaxation" sweeps
executed before the
c           residual is restricted and cycling is
invoked at the next
c           coarser grid level (default value is 2
whenever mgopt(1)=0)
c
c     ipost = mgopt(3)
c
c           the number of "post relaxation" sweeps
executed after cycling
c           has been invoked at the next coarser grid
level and the residual
c           correction has been transferred back
(default value is 1
c           whenever mgopt(1)=0).
c
c *** if iprер, ipost, or (especially) kcycle is greater
than 2
c           than inefficient multigrid cycling has probably
been chosen and
c           the nonfatal error (see below) ierror = -5 will be
set. note
c           this warning may be overridden by any other
nonzero value
c           for ierror.
c
c     interpol = mgopt(4)
```

```

c
c           = 1 if multilinear prolongation
(interpolation) is used to
c           transfer residual corrections and the pde
approximation
c           from coarse to fine grids within full
multigrid cycling.

c
c           = 3 if multicubic prolongation
(interpolation) is used to
c           transfer residual corrections and the pde
approximation
c           from coarse to fine grids within full
multigrid cycling.

c           (this is the default value whenever
mgopt(1)=0).

c
c *** the default values (2,2,1,3) in the vector mgopt
were chosen for
c   robustness. in some cases v(2,1) cycles with
linear prolongation will
c   give good results with less computation
(especially in two-dimensions).
c   this was the default and only choice in an
earlier version of mudpack
c   (see [1]) and can be set with the integer vector
(1,2,1,1) in mgopt.

c
c *** the schedules for one full multigrid cycle
(iguess=0) using v(2,1)
c   cycles and w(2,1) cycles are depicted for a four
level grid below.
c   the number of relaxation sweeps when each grid is
visited are indicated.
c   the "*" stands for prolongation of the full
approximation and the "."
c   stands for transfer of residuals and residual
corrections within the
c   coarse grid correction algorithm (see below). all
version 5.0.1
c   mudpack solvers use only fully weighted residual
restriction
c
c   one fmg with v(2,1) cycles:

```

```

C
C
C      -----2-----1-
-----      level 4
C          * .
C          *
C      -----2-----1-----2-----1-----
C      -----      level 3
C          * .
C          *
C      -----2-----1-----2-----1-----2-----1-----
C      -----      level 2
C          * .
C          *
C      ---3---3-----3-----3-----
C      -----      level 1
C
C
C      one fmg with w(2,1) cycles:
C
C      -----2-----
--1--      level 4
C          *
C          .
C      -----2-----1-----2-----3-----
1----      level 3
C          *
C      -----2-----1-----2-----3-----1-----
C      -----      level 2
C          * . . . . . . . . . . . . .
C      --6---6-----6-----6-----6-----6-----6-----6-----
C      -----      level 1
C
C
C      the form of the "recursive" coarse grid correction
cycling used
c      when kcycle.ge.0 is input is described below in
pseudo-algorithmic
c      language. it is implemented non-recursively in
fortran in mudpack.
c
c      algorithm
cgc(k,l(k),u(k),r(k),kcycle,iprer,ipost,iresw,intpol)
c

```

```

c *** approximately solve l(k)*u(k) = r(k) using
multigrid iteration
c *** k is the current grid level
c *** l(k) is the discretized pde operator at level k
c *** u(k) is the initial guess at level k
c *** r(k) is the right hand side at level k
c *** i(k,k-1) is the restriction operator from level k
to level k-1
c *** (the form of i(k,k-1) depends on iresw)
c *** i(k-1,k) is the prolongation operator from level
k-1 to level k
c *** (the form of i(k-1,k) depends on interpol)
c
c      begin algorithm cg
c
c ***      pre-relax at level k
c
c      . do (i=1,ipr)
c
c      . . relax(l(k),u(k),r(k))
c
c      . end do
c
c      . if (k > 1) then
c
c ***      restrict the residual from level k to level k-
1
c
c      . . r(k-1) = i(k,k-1)(r(k)-l(k)*u(k))
c
c      . . kount = 0
c
c      . . repeat
c
c ***      solve for the residual correction at level k-1
in u(k-1)
c ***      using algorithm cg "kcycle" times (this is
the recursion)
c
c      . . . kount = kount+1
c
c      . . . invoke cg(k-1,l(k-1),u(k-1),r(k-
1),kcycle,ipr,ipost,iresw)
c

```

```

C
C      . . until (kount.eq.kcycle)
C
C ***      transfer residual correction in u(k-1) to
level k
C ***      with the prolongation operator and add to u(k)
C
C      . . u(k) = u(k) + i(k-1,k) (u(k-1))
C
C      . end if
C
C ***      post relax at level k
C
C      . do (i=1,ipost)
C
C      . . relax(l(k),u(k),r(k))
C
C      . end do
C
C      . return
C
C      end algorithm cgc
C
C
C
C ****
C *** output arguments
C ****
C
C ****
C
C ... iparm(16) *** set for intl=0 calls only
C
C          on output iparm(16) contains the actual work
space length
C          required. this will usually be less than
that given by the
C          simplified formula for length=iparm(15) (see
as input argument)
C
C

```

```

c ... iparm(17) *** set for intl=1 calls only
c
c           on output iparm(17) contains the actual
number of multigrid cycles
c           between the finest and coarsest grid levels
used to obtain the
c           approximation when error control (tolmax >
0.0) is set.
c
c
c ... fparm(6) *** set for intl=1 calls with fparm(5)
> 0. only
c
c           on output fparm(6) contains the final
computed maximum relative
c           difference between the last two iterates at
the finest grid level.
c           fparm(6) is computed only if there is error
control (tolmax > 0.0)
c           assume phi1(i,j,k) and phi2(i,j,k) are the
last two computed
c           values for phi(i,j,k) at all points of the
finest grid level.
c           if we define
c
c           phdif = max(abs(phi2(i,j)-phi1(i,j)))
over all i,j
c
c           and
c
c           phmax = max(abs(phi2(i,j))) over all i,j
c
c           then
c
c           fparm(6) = phdif/phmax
c
c           is returned whenever phmax > 0.0. in the
degenerate case
c           phmax = 0.0, fparm(6) = phdif is returned.
c
c
c ... work
c
c           on output work contains intermediate values

```

```

that must not
c           be destroyed if mud2 is to be called again
with intl=1
c
c
c ... phi    *** for intl=1 calls only
c
c           on output phi(i,j) contains the approximation
to p(xi,yj)
c           for all mesh points i = 1,...,nx and
j=1,...,ny. the last
c           computed iterate in phi is returned even if
convergence is
c           not obtained
c
c
c ... ierror
c
c           For intl=iparm(1)=0 initialization calls,
ierror is an
c           error flag that indicates invalid input
arguments when
c           returned positive and nonfatal warnings when
returned
c           negative. Argument checking and
discretization
c           is bypassed for intl=1 calls which can only
return
c           ierror = -1 or 0 or 1.
c
c
c     non-fatal warnings * * *
c
c
c     ==5 if kcycle=mgopt(1) is greater than 2. values
larger than 2 results
c           in an algorithm which probably does far more
computation than
c           necessary. kcycle = 1 (v cycles) or kcycle=2
(w cycles) should
c           suffice for most problems. ierror = -5 is
also set if either
c           iprter = mgopt(2) or ipost=mgopt(3) is greater
than 2. the

```

```

c           ierror=-5 flag is overridden by any other
fatal or non-fatal
c           error.
c
c     ==-4 if there are dominant nonzero first order
terms in the pde which
c           make it "hyperbolic" at the finest grid level.
numerically, this
c           happens if:
c
c           abs(cx)*dlx > 2.*abs(cxx)      (dlx = (xb-
xa) / (nx-1))
c
c                           (or)
c
c           abs(cy)*dly > 2.*abs(cyy)      (dly = (yd-
yc) / (ny-1))
c
c
c           at some fine grid point (xi,yj).  if an
adjustment is not made the
c           condition can lead to a matrix coming from the
discretization
c           which is not diagonally dominant and
divergence is possible. since
c           the condition is "likely" at coarser grid
levels for pde's with
c           nonzero first order terms, the adjustments
(actualy first order
c           approximations to the pde)
c
c
c           cxx = amax1(cxx,0.5*abs(cx)*dx)
c
c                           (and)
c
c           cyy = amax1(cyy,0.5*abs(cy)*dy)
c
c
c           (here dx,dy are the x,y mesh sizes of the
subgrid)
c
c           are made to preserve convergence of multigrid
iteration. if made

```

```
c      at the finest grid level, it can lead to
convergence to an
c      erroneous solution (flagged by ierror = -4).
a possible remedy
c      is to increase resolution. the ierror = -4
flag overrides the
c      nonfatal ierror = -5 flag.
c
c
c      =-3 if the continuous elliptic pde is singular.
this means the
c      boundary conditions are periodic or pure
derivative at all
c      boundaries and ce(x,y) = 0.0 for all x,y. a
solution is still
c      attempted but convergence may not occur due
to ill-conditioning
c      of the linear system coming from the
discretization. the
c      ierror = -3 flag overrides the ierror=-4 and
ierror=-5 nonfatal
c      flags.
c
c
c      =-2 if the pde is not elliptic (i.e.,
cxx*cyy.le.0.0 for some (xi,yj))
c      in this case a solution is still attempted
although convergence
c      may not occur due to ill-conditioning of the
linear system.
c      the ierror = -2 flag overrides the ierror=-
5,-4,-3 nonfatal
c      flags.
c
c
c      =-1 if convergence to the tolerance specified in
tolmax=fparm(5)>0.
c      is not obtained in maxcy=iparm(13) multigrid
cycles between the
c      coarsest (ixp+1,jyq+1) and finest (nx,ny)
grid levels.
c      in this case the last computed iterate is
still returned.
c      the ierror = -1 flag overrides all other
```

```

nonfatal flags
C
C
C      no errors * * *
C
C      = 0
C
C      fatal argument errors * * *
C
C      = 1 if intl=iparm(1) is not 0 on initial call or
not 0 or 1
C          on subsequent calls
C
C      = 2 if any of the boundary condition flags
nxa,nxb,nyc,nyd
C          in iparm(2),iparm(3),iparm(4),iparm(5) are not
0,1 or 2
C          or if nxa,nxb or nyc,nyd are not pairwise
zero.
C
C      = 3 if min0(ixp,jyq) < 2 (ixp = iparm(6), jyq =
iparm(7))
C
C      = 4 if min0(iex,jey) < 1 (iex = iparm(8), jey =
iparm(9)) or
C          if max0(iex,jey) > 50
C
C      = 5 if nx.ne.ixp*2** (iex-1)+1 or
ny.ne.jyq*2** (jey-1)+1
C          (nx = iparm(10), ny = iparm(11))
C
C      = 6 if iguess = iparm(12) is not equal to 0 or 1
C
C      = 7 if maxcy = iparm(13) < 1
C
C      = 8 if method = iparm(14) is not 0,1,2, or 3
C
C      = 9 if length = iparm(15) is too small (see
iparm(16) on output
C          for minimum required work space length)
C
C      =10 if xa >= xb or yc >= yd
C
(xa=fparm(1),xb=fparm(2),yc=fparm(3),yd=fparm(4))

```

```
c      =11 if tolmax = fparm(5) < 0.0
c
c      errors in setting multigrid options * * * (see
also ierror=-5)
c
c      =12 if kcycle = mgopt(1) < 0 or
c          if iprер = mgopt(2) < 1 or
c          if ipost = mgopt(3) < 1 or
c          if interpol = mgopt(4) is not 1 or 3
c
c
*****
*
C
*****
*
C
C      end of mud2 documentation
C
C
*****
**
C
*****
**
C
```

MUD24

```
C      *          copyright (c) 2008 by UCAR
*
C      *
*
C      *          University Corporation for Atmospheric
Research           *
C      *
*
C      *          all rights reserved
*
C      *
*
C      *          MUDPACK version 5.0.1
*
C      *
*
C      *          A Fortran Package of Multigrid
*
C      *
*
C      *          Subroutines and Example Programs
*
C      *
*
C      *          for Solving Elliptic Partial Differential
Equations        *
C      *
*
C      *          by
*
C      *
*
C      *          John Adams
*
C      *
*
C      *          of
*
C      *
*
C      *          the National Center for Atmospheric
Research           *
C      *
*
```

```
C      *                                Boulder, Colorado (80307)
U.S.A.          *
C      *
*
C      *                                which is sponsored by
*
C      *
*
C      *                                the National Science Foundation
*
C      *
*
C      * ***** *
* * * * *
C
C ... file mud24.d
C
C      contains documentation for subroutine
mud24(work,phi,ierror)
C      A sample fortran driver is file "tmud24.f".
C
C ... required MUDPACK files
C
C      mud2.f, mudcom.f
C
C ... purpose
C
C      mud24 attempts to improve the estimate in phi,
obtained by calling
C      mud2, from second to fourth order accuracy. see
the file "mud2.d"
C      for a detailed discussion of the elliptic pde
approximated and
C      arguments "work,phi" which are also part of the
argument list for
C      mud2.
C
C ... assumptions
C
C      * phi contains a second-order approximation from
an earlier mud2 call
C
C      * arguments "work,phi" are the same used in
calling mud2
```

```
c
c      * "work,phi" have not changed since the last call
to mud2
c
c      * the finest grid level contains at least 6
points in each direction
c
c
c *** warning
c
c      if the first assumption is not true then a fourth
order approximation
c      cannot be produced in phi.  the last assumption
(adequate grid size)
c      is the only one checked. failure in any of the
others can result in
c      in an undetectable error.
c
c ...
language
c
c      fortran90/fortran77
c
c ...
portability
c
c      mudpack5.0.1 software has been compiled and tested
with fortran77
c      and fortran90 on a variety of platforms.
c
c ...
methods
c
c      details of the methods employed by the solvers in
mudpack are given
c      in [1,9]. [1,2,9] contain performance
measurements on a variety of
c      elliptic pdes (see "references" in the file
"readme") . in summary:
c
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
estimate (i.e.,
c      discretization level error is reached) then this
```

```
can be improved to a
c      fourth-order estimate using the technique of
"deferred corrections"
c      the values in the solution array are used to
generate a fourth-order
c      approximation to the truncation error. second-
order finite difference
c      formula are used to approximate third and fourth
partial derivatives
c      of the solution function [3]. the truncation
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side. the default
multigrid options
c      are used to compute the fourth-order correction
term which is added
c      to the original solution array.

c
c
c ... references (partial)
c
c
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
c      Solution of Linear Elliptic Partial Differential
Equations,"
c      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
c
c      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
c      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
c      1989, pp.1-12.
c      .
c      .
c      .
c      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
Haidvogel, and V. Pizzo,
c      "Applications of Multigrid Software in the
Atmospheric Sciences,"
c      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-
1458.
```

```
C .
C .
C .
C     [9] J. Adams, "Recent Enhancements in MUDPACK, a
Multigrid Software
C     package for Elliptic Partial Differential
Equations," Applied Math.
C     and Comp., 1991, vol. 43, May 1991, pp. 79-94.
C
C     [10]J. Adams, "MUDPACK-2: Multigrid Software for
Approximating
C     Elliptic Partial Differential Equations on Uniform
Grids with
C     any Resolution," Applied Math. and Comp., 1993,
vol. 53, February
C     1993, pp. 235-249
C .
C .
C .
C
C ... error argument
C
C     = 0 if no error is detected
C
C     = 30 if min0(nx,ny) < 6 where nx,ny are the fine
grid sizes
C             in the x,y directions.
C
C
C
*****
*****
```

\*\*\*\*\*

```
C
*****
*****
```

\*\*\*\*\*

```
C
C     end of mud24 documentation
C
C
*****
*****
```

\*\*\*\*\*

```
C
*****
```

```
*****
```

```
C
```

---

## MUD24CR

```
C  
C      file mud24cr.d  
C  
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C      *  
*  
C      *          copyright (c) 2008 by UCAR  
*  
C      *  
*  
C      *          University Corporation for Atmospheric  
Research           *  
C      *  
*  
C      *          all rights reserved  
*  
C      *  
*  
C      *          MUDPACK version 5.0.1  
*  
C      *  
*  
C      *          A Fortran Package of Multigrid  
*  
C      *  
*  
C      *          Subroutines and Example Programs  
*  
C      *  
*  
C      *          for Solving Elliptic Partial Differential  
Equations       *
```

```
C      *
*
C      *                                by
*
C      *
*
C      *                                John Adams
*
C      *
*
C      *                                of
*
C      *
*
C      *                                the National Center for Atmospheric
Research          *
C      *
*
C      *                                Boulder, Colorado (80307)
U.S.A.          *
C      *
*
C      *                                which is sponsored by
*
C      *
*
C      *                                the National Science Foundation
*
C      *
*
C      *      * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C
C ... file mud24cr.d
C
C contains documentation for:
C subroutine mud24cr(work,coef,bndyc,phi,ierror)
C A sample fortran driver is file "tmud24cr.f".
C
C ... required MUDPACK files
C
C     mud2cr.f, mudcom.f
C
C ... purpose
```

```
c
c      mud24cr attempts to improve the estimate in phi,
obtained by calling
c      mud2cr, from second to fourth order accuracy.
see the file "mud2cr.d"
c      for a detailed discussion of the elliptic pde
approximated and
c      arguments "work,coef,bndyc,phi" which are also
part of the argument
c      list for mud2cr
c
c ... assumptions
c
c      * phi contains a second-order approximation from
an earlier mud2cr call
c
c      * arguments "work,coef,bndyc,phi" are the same
used in calling mud2cr
c
c      * "work,coef,bndyc,phi" have not changed since
the last call to mud2cr
c
c      * the finest grid level contains at least 6
points in each direction
c
c
c *** warning
c
c      if the first assumption is not true then a fourth
order approximation
c      cannot be produced in phi. the last assumption
(adequate grid size)
c      is the only one checked. failure in any of the
others can result in
c      in an undetectable error.
c
c ... language
c
c      fortran90/fortran77
c
c ... portability
c
c      mudpack5.0.1 software has been compiled and tested
with fortran77
```

```
c      and fortran90 on a variety of platforms.  
c  
c ... methods  
c  
c      details of the methods employed by the solvers in  
mudpack are given  
c      in [1,9]. [1,2,9] contain performance  
measurements on a variety of  
c      elliptic pdes (see "references" in the file  
"readme"). in summary:  
c  
c  
c *** higher order solution (fourth-order solvers) (see  
[9,19,21])  
c  
c      if the multigrid cycling results in a second-order  
estimate (i.e.,  
c      discretization level error is reached) then this  
can be improved to a  
c      fourth-order estimate using the technique of  
"deferred corrections"  
c      the values in the solution array are used to  
generate a fourth-order  
c      approximation to the truncation error. second-  
order finite difference  
c      formula are used to approximate third and fourth  
partial derivatives  
c      of the solution function [3]. the truncation  
error estimate is  
c      transferred down to all grid levels using weighted  
averaging where  
c      it serves as a new right hand side. the default  
multigrid options  
c      are used to compute the fourth-order correction  
term which is added  
c      to the original solution array.  
c  
c  
c ... references (partial)  
c  
c  
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software  
for the Efficient  
c      Solution of Linear Elliptic Partial Differential
```

Equations,"

c        Applied Math. and Comput. vol.34, Nov 1989,  
pp.113-146.

c

c        [2] J. Adams, "FMG Results with the Multigrid  
Software Package MUDPACK,"

c        proceedings of the fourth Copper Mountain  
Conference on Multigrid, SIAM,

c        1989, pp.1-12.

c        .

c        .

c        .

c        [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.  
Haidvogel, and V. Pizzo,

c        "Applications of Multigrid Software in the  
Atmospheric Sciences,"

c        Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-  
1458.

c        .

c        .

c        .

c        [9] J. Adams, "Recent Enhancements in MUDPACK, a  
Multigrid Software

c        package for Elliptic Partial Differential  
Equations," Applied Math.

c        and Comp., 1991, vol. 43, May 1991, pp. 79-94.

c

c        [10]J. Adams, "MUDPACK-2: Multigrid Software for  
Approximating

c        Elliptic Partial Differential Equations on Uniform  
Grids with

c        any Resolution," Applied Math. and Comp., 1993,  
vol. 53, February

c        1993, pp. 235-249

c        .

c        .

c        .

c

c ... error argument

c

c        = 0 if no error is detected

c

c        = 30 if min0(nx,ny) < 6 where nx,ny are the fine  
grid sizes

```
C           in the x,y directions.  
C  
C  
C  
*****  
*****  
C  
*****  
*****  
C  
*****  
*****  
C  
      end of mud24cr documentation  
C  
C  
*****  
*****  
C  
*****  
*****  
C
```

---

## MUD24SP

```
C  
C      file mud24sp.d  
C  
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * * *  
C      *  
*  
C      *          copyright (c) 2008 by UCAR  
*  
C      *  
*  
C      *          University Corporation for Atmospheric  
Research          *  
C      *  
*  
C      *          all rights reserved
```

\*  
C \*  
\*  
C \* MUDPACK version 5.0.1  
\*  
C \*  
\*  
C \* A Fortran Package of Multigrid  
\*  
C \*  
\*  
C \* Subroutines and Example Programs  
\*  
C \*  
\*  
C \* for Solving Elliptic Partial Differential  
Equations \*  
C \*  
\*  
C \* by  
\*  
C \*  
\*  
C \* John Adams  
\*  
C \*  
\*  
C \* of  
\*  
C \*  
\*  
C \* the National Center for Atmospheric  
Research \*  
C \*  
\*  
C \* Boulder, Colorado (80307)  
U.S.A. \*  
C \*  
\*  
C \* which is sponsored by  
\*  
C \*  
\*  
C \* the National Science Foundation

```
*  
C      *  
*  
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C  
C ... file mud24sp.d  
C  
C      contains documentation for subroutine  
mud24sp(work,phi,ierror)  
C      A sample fortran driver is file "tmud24sp.f".  
C  
C ... required MUDPACK files  
C  
C      mud2sp.f, mudcom.f  
C  
C ... purpose  
C  
C      mud24sp attempts to improve the estimate in phi,  
obtained by calling  
C      mud2sp, from second to fourth order accuracy.  
see the file "mud2sp.d"  
C      for a detailed discussion of the elliptic pde  
approximated and  
C      arguments "work,phi" which are also part of the  
argument list for  
C      mud2sp.  
C  
C ... assumptions  
C  
C      * phi contains a second-order approximation from  
an earlier mud2sp call  
C  
C      * arguments "work,phi" are the same used in  
calling mud2sp  
C  
C      * "work,phi" have not changed since the last call  
to mud2sp  
C  
C      * the finest grid level contains at least 6  
points in each direction  
C  
C  
C *** warning
```

```
c
c      if the first assumption is not true then a fourth
order approximation
c      cannot be produced in phi.  the last assumption
(adequate grid size)
c      is the only one checked. failure in any of the
others can result in
c      in an undetectable error.
c
c ... language
c
c      fortran90/fortran77
c
c ... portability
c
c      mudpack5.0.1 software has been compiled and tested
with fortran77
c      and fortran90 on a variety of platforms.
c
c ... methods
c
c      details of the methods employed by the solvers in
mudpack are given
c      in [1,9]. [1,2,9] contain performance
measurements on a variety of
c      elliptic pdes (see "references" in the file
"readme"). in summary:
c
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
estimate (i.e.,
c      discretization level error is reached) then this
can be improved to a
c      fourth-order estimate using the technique of
"deferred corrections"
c      the values in the solution array are used to
generate a fourth-order
c      approximation to the truncation error. second-
order finite difference
c      formula are used to approximate third and fourth
partial derivatives
```

```
c      of the solution function [3].  the truncation
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side.  the default
multigrid options
c      are used to compute the fourth-order correction
term which is added
c      to the original solution array.
c
c
c ... references (partial)
c
c
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
c      Solution of Linear Elliptic Partial Differential
Equations,"
c      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
c
c      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
c      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
c      1989, pp.1-12.
c      .
c      .
c      .
c      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
Haidvogel, and V. Pizzo,
c      "Applications of Multigrid Software in the
Atmospheric Sciences,"
c      Mon. Wea. Rev.,vol. 120 # 7, July 1992, pp. 1447-
1458.
c      .
c      .
c      .
c      [9] J. Adams, "Recent Enhancements in MUDPACK, a
Multigrid Software
c      package for Elliptic Partial Differential
Equations," Applied Math.
c      and Comp., 1991, vol. 43, May 1991, pp. 79-94.
c
```



```
C
C      file mud2cr.d
C
C      * * * * *
* * * * *
C      *
*
C      *           copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *           all rights reserved
*
C      *
*
C      *           MUDPACK version 5.0.1
*
C      *
*
C      *           A Fortran Package of Multigrid
*
C      *
*
C      *           Subroutines and Example Programs
*
C      *
*
C      *           for Solving Elliptic Partial Differential
Equations          *
C      *
*
C      *           by
*
C      *
*
C      *           John Adams
*
C      *
```

C \* of  
\*  
C \*  
\*  
C \* the National Center for Atmospheric  
Research \*  
C \*  
\*  
C \* Boulder, Colorado (80307)  
U.S.A. \*  
C \*  
\*  
C \* which is sponsored by  
\*  
C \*  
\*  
C \* the National Science Foundation  
\*  
C \*  
\*  
C \*  
\* \* \* \* \* \* \*  
C  
C ... file mud2cr.d  
C  
C contains documentation for:  
C subroutine  
mud2cr(iparm,fparm,work,coef,bndyc,rhs,phi,mgopt,ierror)  
C a sample fortran driver is file "tmud2cr.f".  
C  
C ... required mudpack files  
C  
C mudcom.f  
C  
C ... purpose  
C  
C subroutine mud2cr automatically discretizes and  
attempts to compute  
C the second-order difference approximation to the  
two-dimensional  
C linear nonseparable elliptic partial differential  
equation with cross  
C derivative term on a rectangle. the approximation  
is generated on a

```

c      uniform grid covering the rectangle (see mesh
description below) .
c      boundary conditions may be specified (dirchlet),
periodic, or mixed
c      oblique derivative (see bndyc) in any combination.
the form of the pde
c      approximated is:
c
c
c      cxx(x,y)*pxx + cxy(x,y)*pxy + cyy(x,y)*pyy +
cx(x,y)*px +
c
c      cy(x,y)*py + ce(x,y)*p(x,y) = r(x,y) .
c
c
c      pxx,pxy,pyy,px,py are second and first partial
derivatives of the
c      unknown real solution function p(x,y) with respect
to the
c      independent variables x,y.  cxx,cxy,cyy,cx,cy,ce
are the known
c      real coefficients of the elliptic pde and r(x,y)
is the known
c      real right hand side of the equation.  cxx and cyy
should be
c      positive for all x,y in the solution region and
c
c      4*cxx(x,y)*cyy(x,y) .le. cxy(x,y)**2
c
c      for ellipticity (see ierror=-2).  nonseparability
means some
c      of the coefficients depend on both x and y and
cxy.ne.0.  if
c      the pde is separable and cxy = 0 then subroutine
mud2sp should
c      be used.
c
c
c ... mesh description . . .
c
c      the approximation is generated on a uniform nx by
ny grid.  the grid
c      is superimposed on the rectangular solution region
c

```

```
c [xa,xb] x [yc,yd].  
c  
c let  
c  
c dlx = (xb-xa) / (nx-1), dly = (yd-yc) / (ny-1)  
c  
c be the uniform grid increments in the x,y  
directions. then  
c  
c xi=xa+(i-1)*dlx, yj=yc+(j-1)*dly  
c  
c for i=1,...,nx and j=1,...,ny denote the x,y  
uniform mesh points  
c  
c  
c ... language  
c  
c fortran90/fortran77  
c  
c  
c ... portability  
c  
c mudpack5.0.1 software has been compiled and tested  
with fortran77  
c and fortran90 on a variety of platforms.  
c  
c ... methods  
c  
c details of the methods employed by the solvers in  
mudpack are given  
c in [1,9]. [1,2,9] contain performance  
measurements on a variety of  
c elliptic pdes (see "references" in the file  
"readme"). in summary:  
c  
c *** discretization and solution (second-order solvers)  
(see [1])  
c  
c the pde and boundary conditions are automatically  
discretized at all  
c grid levels using second-order finite difference  
formula. diagonal  
c dominance at coarser grid levels is maintained in  
the presence of
```

```
c      nonzero first-order terms by adjusting the second-
order coefficient
c      when necessary. the resulting block tri-diagonal
linear system is
c      approximated using multigrid iteration
[10,11,13,15,16,18]. version
c      5.0.1 of mudpack uses only fully weighted residual
restriction. defaults
c      include cubic prolongation and w(2,1) cycles.
these can be overridden
c      with selected multigrid options (see "mgopt").
error control based on
c      maximum relative differences is available. full
multigrid cycling (fmg)
c      or cycling beginning or restarting at the finest
grid level can be
c      selected. a menu of relaxation methods including
gauss-seidel point,
c      line relaxation(s) (in any combination of
directions) and planar
c      relaxation (for three-dimensional anisotropic
problems) are provided.
c      all methods use ordering based on alternating
points (red/black),
c      lines, or planes for cray vectorization and
improved convergence
c      rates [14].
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
estimate (i.e.,
c      discretization level error is reached) then this
can be improved to a
c      fourth-order estimate using the technique of
"deferred corrections."
c      the values in the solution array are used to
generate a fourth-order
c      approximation to the truncation error. second-
order finite difference
c      formula are used to approximate third and fourth
partial derivatives
c      of the solution function [3]. the truncation
```

```
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side.  the default
multigrid options
c      are used to compute the fourth-order correction
term which is added
c      to the original solution array.
c
c
c ... references (partial)
c
c
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
c      Solution of Linear Elliptic Partial Differential
Equations,"
c      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
c
c      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
c      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
c      1989, pp.1-12.
c      .
c      .
c      .
c      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
Haidvogel, and V. Pizzo,
c      "Applications of Multigrid Software in the
Atmospheric Sciences,"
c      Mon. Wea. Rev.,vol. 120 # 7, July 1992, pp. 1447-
1458.
c      .
c      .
c      .
c      [9] J. Adams, "Recent Enhancements in MUDPACK, a
Multigrid Software
c      package for Elliptic Partial Differential
Equations," Applied Math.
c      and Comp., 1991, vol. 43, May 1991, pp. 79-94.
c
c      [10]J. Adams, "MUDPACK-2: Multigrid Software for
```

```
Approximating
c      Elliptic Partial Differential Equations on Uniform
Grids with
c      any Resolution," Applied Math. and Comp., 1993,
vol. 53, February
c      1993, pp. 235-249
c      .
c      .
c      .
c
c ... argument description
c
c
c
c *****
***** *****
c *** input arguments
***** *****
c
***** *****
***** *****
c
c
c ... iparm
c
c           an integer vector of length 17 used to pass
integer
c           arguments. iparm is set internally and
defined as
c           follows:
c
c
c ... intl=iparm(1)
c
c           an initialization argument. intl=0 must be
input
c           on an initial call. in this case input
arguments will
c           be checked for errors and the elliptic
partial differential
c           equation and boundary conditions will be
discretized using
c           second order finite difference formula.
c
```

```
c ***      an approximation is not generated after an
intl=0 call!
c           mud2cr should be called with intl=1 to
approximate the elliptic
c           pde discretized by the intl=0 call.  intl=1
should also
c           be input if mud2cr has been called earlier
and only the
c           values in in rhs (see below) or gbdy (see
bndyc below)
c           or phi (see below) have changed.  this will
bypass
c           redundant pde discretization and argument
checking
c           and save computational time.  some examples
of when
c           intl=1 calls should be used are:
c
c           (0) after a intl=0 argument checking and
discretization call
c
c           (1) mud2cr is being recalled for additional
accuracy.  in
c               this case iguess=iparm(12)=1 should also
be used.
c
c           (2) mud2cr is being called every time step in
a time dependent
c               problem (see discussion below) where the
elliptic operator
c               does not depend on time.
c
c           (3) mud2cr is being used to solve the same
elliptic equation
c               for several different right hand sides
(iguess=0 should
c               probably be used for each new righthand
side).
c
c           intl = 0 must be input before calling with
intl = 1 when any
c               of the following conditions hold:
c
c           (a) the initial call to mud2cr
```

```
c          (b) any of the integer arguments other than
iguess=iparm(12)
c          or maxcy=iparm(13) or mgopt have changed
since the previous
c          call.
c
c          (c) any of the floating point arguments other
than tolmax=
c          fparm(5) have changed since the previous
call
c
c          (d) any of the coefficients input by coef
(see below) have
c          changed since the previous call
c
c          (e) any of the "alfa" coefficients input by
bndyc (see below)
c          have changed since the previous call.
c
c          if any of (a) through (e) are true then the
elliptic pde
c          must be discretized or rediscretized. if
none of (a)
c          through (e) holds, calls can be made with
intl=1.
c          incorrect calls with intl=1 will produce
erroneous results.
c ***      the values set in the saved work space "work"
(see below) with
c          an intl=0 call must be preserved with
subsequent intl=1 calls.
c
c          MUDPACK software performance should be
monitored for intl=1
c          calls. The intl=0 discretization call
performance depends
c          primarily on the efficiency or lack of
efficiency of the
c          user provided subroutines for pde
coefficients and
c          boundary conditions.
c
```

```

c ... nxa=iparm(2)
c
c           flags boundary conditions on the edge x=xa
c
c           = 0 if p(x,y) is periodic in x on [xa,xb]
c               (i.e., p(x+xb-xa,y) = p(x,y) for all x,y
c               (if nxa=0 then nxb=0 is required, see
ierror = 2)
c
c           = 1 if p(xa,y) is specified (this must be input
thru phi(1,j))
c
c           = 2 if there are mixed derivative boundary
conditions at x=xa
c               (see bndyc)
c
c
c ... nxn=iparm(3)
c
c           flags boundary conditions on the edge x=xb
c
c           = 0 if p(x,y) is periodic in x on [xa,xb]
c               (i.e., p(x+xb-xa,y) = p(x,y) for all x,y
c               (if nxb=0 then nxa=0 is required, see
ierror = 2)
c
c           = 1 if p(xb,y) is specified (this must be input
thru phi(nx,j))
c
c           = 2 if there are mixed derivative boundary
conditions at x=xb
c               (see bndyc)
c
c
c ... nyc=iparm(4)
c
c           flags boundary conditions on the edge y=yc
c
c           = 0 if p(x,y) is periodic in y on [yc,yd]
c               (i.e., p(x,y+yd-yc) = p(x,y) for all x,y
c               (if nyc=0 then nyd=0 is required, see
ierror = 2)
c
c           = 1 if p(x,yc) is specified (this must be input

```

```

thru phi(i,1))
C
C      = 2 if there are mixed derivative boundary
conditions at y=yc
C          (see bndyc)
C
C
C ... nyd=iparm(5)
C
C      flags boundary conditions on the edge y=yd
C
C      = 0 if p(x,y) is periodic in y on [yc,yd]
C          (i.e., p(x,y+yd-yc) = p(x,y) for all x,y
C          (if nyd=0 then nyc=0 is required, see
ierror = 2)
C
C      = 1 if p(x,yd) is specified (this must be input
thru phi(i,ny))
C
C      = 2 if there are mixed derivative boundary
conditions at y=yd
C          (see bndyc)
C
C
C *** grid size arguments
C
C
C ... ixp = iparm(6)
C
C      an integer greater than one which is used in
defining the number
C          of grid points in the x direction (see nx =
iparm(10)). "ixp+1"
C          is the number of points on the coarsest x
grid visited during
C          multigrid cycling. ixp should be chosen as
small as possible.
C          recommended values are the small primes 2 or
3.
C          larger values can reduce multigrid
convergence rates considerably,
C          especially if line relaxation in the x
direction is not used.
C          if ixp > 2 then it should be 2 or a small odd

```

```
value since a power
c          of 2 factor of ixp can be removed by
increasing iex = iparm(8)
c          without changing nx = iparm(10).
c
c
c ... jyq = iparm(7)
c
c          an integer greater than one which is used in
defining the number
c          of grid points in the y direction (see ny =
iparm(11)). "jyq+1"
c          is the number of points on the coarsest y
grid visited during
c          multigrid cycling. jyq should be chosen as
small as possible.
c          recommended values are the small primes 2 or
3.
c          larger values can reduce multigrid
convergence rates considerably,
c          especially if line relaxation in the y
direction is not used.
c          if jyq > 2 then it should be 2 or a small odd
value since a power
c          of 2 factor of jyq can be removed by
increasing jey = iparm(9)
c          without changing ny = iparm(11).
c
c
c ... iex = iparm(8)
c
c          a positive integer exponent of 2 used in
defining the number
c          of grid points in the x direction (see nx =
iparm(10)).
c          iex .le. 50 is required. for efficient
multigrid cycling,
c          iex should be chosen as large as possible and
ixp=iparm(8)
c          as small as possible within grid size
constraints when
c          defining nx.
c
c
```

```

c ... jey = iparm(9)
c
c           a positive integer exponent of 2 used in
defining the number
c           of grid points in the y direction (see ny =
iparm(11)).
c           jey .le. 50 is required. for efficient
multigrid cycling,
c           jey should be chosen as large as possible and
jyq=iparm(7)
c           as small as possible within grid size
constraints when
c           defining ny.
c
c
c
c ... nx = iparm(10)
c
c           the number of equally spaced grid points in
the interval [xa,xb]
c           (including the boundaries). nx must have the
form
c
c           nx = ixp*(2** (iex-1)) + 1
c
c           where ixp = iparm(6), iex = iparm(8).
c
c
c ... ny = iparm(11)
c
c           the number of equally spaced grid points in
the interval [yc,yd]
c           (including the boundaries). ny must have the
form:
c
c           ny = jyq*(2** (jey-1)) + 1
c
c           where jyq = iparm(7), jey = iparm(9).
c
c
c *** example
c
c           suppose a solution is wanted on a 33 by 97
grid. then

```

```

c           ixp=2, jyq=6 and iex=jey=5 could be used. a
better
c           choice would be ixp=2, jyq=3, and iex=5,
jey=6.
c
c *** grid size flexibility considerations:
c
c       the hybrid multigrid/direct method code muh2cr
provides more grid size
c       flexibility than mud2cr by removing the constraint
that ixp and jyq are
c       2 or 3. this is accomplished by using a direct
method whenever the
c       coarsest (ixp+1) x (jyq+1) grid is encountered in
multigrid cycling.
c       if nx = ixp+1 and ny = jyq+1 then muh2cr becomes a
full direct method.
c       muh2cr is roughly equivalent to mud2cr in
efficiency as long as ixp and
c       jyq remain "small" (see muh2cr.d). if the problem
to be approximated
c       requires a grid neither mud2cr or muh2cr can
exactly fit then another option
c       is to generate an approximation on a "close grid"
using mud2cr or muh2cr.
c       then transfer the result to the required grid
using cubic interpolation
c       via the package "regridpack" (contact john adams
about this software)
c
c *** note
c
c       let g be the nx by ny fine grid on which the
approximation is
c       generated and let n = max0(iex,jey). in mudpack,
multigrid
c       cycling is implemented on the ascending chain of
grids
c
c           g(1) < ... < g(k) < ... < g(n) = g.
c
c       each g(k) (k=1,...,n) has mx(k) by my(k) grid
points
c       given by:

```

```

C
C      mx(k) = ixp*[2** (max0(iex+k-n,1)-1)] + 1
C
C      my(k) = jyq*[2** (max0(jey+k-n,1)-1)] + 1
C
C
C
C ... iguess=iparm(12)
C
C           = 0 if no initial guess to the pde is
provided
C
C           = 1 if an initial guess to the pde is at the
finest grid
C           level is provided in phi (see below)
C
C           comments on iguess = 0 or 1 . . .
C
C     even if iguess = 0, phi must be initialized at all
grid points (this
C     is not checked). phi can be set to 0.0 at non-
dirchlet grid points
C     if nothing better is available. the values set in
phi when iguess = 0
C     are passed down and serve as an initial guess to
the pde at the coarsest
C     grid level where cycling commences. in this
sense, values input in
C     phi always serve as an initial guess. setting
iguess = 0 forces full
C     multigrid cycling beginning at the coarsest and
finishing at the finest
C     grid level.
C
C     if iguess = 1 then the values input in phi are an
initial guess to the
C     pde at the finest grid level where cycling begins.
this option should
C     be used only if a "very good" initial guess is
available (as, for
C     example, when restarting from a previous iguess=0
call).
C
C     time dependent problems . . .

```

```

c
c *** assume we are solving an elliptic pde every time
step in a
c      marching problem of the form:
c
c      l(p(t)) = r(t)
c
c      where the differential operator "l" has no time
dependence,
c      "p(t)" is the solution and "r(t)" is the right
hand side at
c      current time "t". let "dt" be the increment
between time steps.
c      then p(t) can be used as an initial guess to
p(t+dt) with
c      intl = 1 when solving
c
c      l(p(t+dt)) = r(t+dt).
c
c      after the first two time steps, rather than
continue, it would
c      be better to define the "correction" term:
c
c      e(t,dt) = p(t+dt) - p(t)
c
c      this clearly satisfies the equation
c
c      l(e(t,dt)) = r(t+dt) - r(t).
c
c      this should be solved with iguess = 0 and intl =
1. boundary
c      conditions for e(t,dt) are obtained from the
boundary conditions
c      for p(t) by subtracting given values at t from
given values at
c      t+dt. for example if
c
c      d(p(t))/dx = f(t), d(p(t+dt))/dx = f(t+dt)
c
c      at some x boundary then e(t,dt) satisfies the
derivative
c      boundary condition
c
c      d(e(t,dt))/dx = f(t+dt) - f(t).

```

```
c
c      e(t,dt) can be preset to 0.0 (at nondirichlet
points) or (if p(t-dt)
c      is saved) to p(t)-p(t-dt). with iguess = 0, these
values will serve
c      as an initial guess to e(t,dt) at the coarsest
grid level. this
c      approach has the advantage that a full sequence of
multigrid cycles,
c      beginning at the coarsest grid level, is invoked
every time step in
c      solving for e(t,dt). a few digits of accuracy in
e(t,dt), which is
c      ordinarily much smaller than p(t), will yield
several more digits of
c      accuracy in the final approximation:
c
c          p(t+dt) = p(t) + e(t,dt).
c
c      using this approach to integrate in time will give
more accuracy
c      then using p(t) as an initial guess to p(t+dt) for
all time steps.
c      it does require additional storage.
c
c      if the differential operator "l" has time
dependence (either thru
c      the coefficients in the pde or the coefficients in
the derivative
c      boundary conditions) then use p(t) as an initial
guess to p(t+dt)
c      when solving
c
c          l(t+dt) (p(t+dt)) = r(t+dt)
c
c      with intl = 0 for all time steps (the
discretization must be repeated
c      for each new "t"). either iguess = 0 (p(t) will
then be an initial
c      guess at the coarsest grid level where cycles will
commence) or
c      iguess = 1 (p(t) will then be an initial guess at
the finest grid
c      level where cycles will remain fixed) can be
```

```
tried.  
C  
C  
C ... maxcy = iparm(13)  
C  
C           the exact number of cycles executed between  
the finest (nx by  
C           ny) and the coarsest ((ixp+1) by (jyq+1))  
grid levels when  
C           tolmax=fparm(5)=0.0 (no error control). when  
tolmax > 0.0  
C           is input (error control) then maxcy is a  
limit on the number  
C           of cycles between the finest and coarsest  
grid levels. in  
C           any case, at most maxcy*(iprert+ipost)  
relaxation sweeps are  
C           are performed at the finest grid level (see  
iprert=mgopt(2),  
C           ipost=mgopt(3) below). when multigrid  
iteration is working  
C           "correctly" only a few are required for  
convergence. large  
C           values for maxcy should not be necessary.  
C  
C  
C ... method = iparm(14) determines the method of  
relaxation  
C           (gauss-seidel based on alternating points  
or lines)  
C  
C           = 0 for point relaxation  
C  
C           = 1 for line relaxation in the x direction  
C  
C           = 2 for line relaxation in the y direction  
C  
C           = 3 for line relaxation in both the x and y  
direction  
C  
C  
C *** choice of method. . .  
C  
C           let fx represent the quantity cxx(x,y)/dlx**2 over
```

```

the solution region.

c
c      let fy represent the quantity cyy(x,y)/dly**2 over
the solution region
c
c      if fx,fy are roughly the same size and do not vary
too much over
c      the solution region choose method = 0.  if this
fails try method=3.
c
c      if fx is much greater than fy choose method = 1.
c
c      if fy is much greater than fx choose method = 2
c
c      if neither fx or fy dominates over the solution
region and they
c      both vary considerably choose method = 3.
c
c
c ... length = iparm(15)
c
c      the length of the work space provided in
vector work (see below).
c      let isx = 0 if method = 0 or method = 2
c      let isx = 3 if method = 1 or method = 3 and
nxa.ne.0
c      let isx = 5 if method = 1 or method = 3 and
nxa.eq.0
c      let jsy = 0 if method = 0 or method = 1
c      let jsy = 3 if method = 2 or method = 3 and
nyc.ne.0
c      let jsy = 5 if method = 2 or method = 3 and
nyc.eq.0
c      then . . .
c
c      length =
[7* (nx+2) * (ny+2)+4* (11+isx+jsy) *nx*ny]/3
c
c      will suffice in most cases.  the exact
minimal work space
c      length required for the current nx,ny and
method is output
c      in iparm(16) (even if iparm(15) is too
small).  this will be

```

```
c           less then the value given by the simplified
formula above
c           in most cases.
c
c
c ... fparm
c
c           a floating point vector of length 6 used to
efficiently
c           pass floating point arguments.  fparm is set
internally
c           in mud2cr and defined as follows . . .
c
c
c ... xa=fparm(1), xb=fparm(2)
c
c           the range of the x independent variable. xa
must
c           be less than xb
c
c
c ... yc=fparm(3), yd=fparm(4)
c
c           the range of the y independent variable. yc
must
c           be less than yd.
c
c
c ... tolmax = fparm(5)
c
c           when input positive, tolmax is a maximum
relative error tolerance
c           used to terminate the relaxation iterations.
assume phi1(i,j)
c           and phi2(i,j) are the last two computed
approximations at all
c           grid points of the finest grid level. if we
define
c
c           phdif = max(abs(phi2(i,j)-phi1(i,j))) for
all i,j
c
c           and
c
```

```
c           phmax = max(abs(phi2(i,j))) for all i,j
c
c           then "convergence" is considered to have
occurred if and only if
c
c           phdif/phmax < tolmax.
c
c
c           if tolmax=fparm(5)=0.0 is input then there is
no error control
c           and maxcy cycles from the finest grid level
are executed. maxcy
c           is a limit which cannot be exceeded even with
error control.
c           *** calls with tolmax=0.0, when appropriate
because of known
c           convergence behavior, are more efficient than
calls with tolmax
c           positive (i.e., if possible do not use error
control!).
c
c ... work
c
c           a one dimensional real saved work space (see
iparm(15) for
c           length) which must be preserved from the
previous call when
c           calling with intl=iparm(1)=1.
c
c ... bndyc
c
c           a subroutine with arguments
(kbdy,xory,alfa,beta,gama,gbody) which
c           are used to input mixed boundary conditions
to mud2cr. bndyc
c           must be declared "external" in the program
calling mud2cr.
c           the boundaries are numbered one thru four and
the mixed
c           derivative boundary conditions are described
below (see the
c           sample driver code "tmud2cr.f" for an example
of how bndyc is
c           can beset up).
```

```

C
C          * * * * * * * * * * * * * * * *   y=yd
C          *           kbdy=4           *
C          *
C          *
C          *
C          *
C          * kbdy=1           kbdy=2 *
C          *
C          *
C          *
C          *
C          *           kbdy=3           *
C          * * * * * * * * * * * * * * * *   y=yc
C
C          x=xa           x=xb
C
C
C
C      (1)    the kbdy=1 boundary
C
C          this is the edge x=xa where nxa=iparm(2) = 2
flags
C          a mixed boundary condition of the form
C
C          alfxa(y)*px + betxa(y)*py +
gamxa(y)*p(xa,y) = gbdxa(y)
C
C          in this case kbdy=1,xory=y will be input to
bndyc and
C          alfa,beta,gama,gbdy corresponding to
alfxa(y),betxa(y),gamxa(y),
C          gbdxa(y) must be returned.  alfxa(y) = 0. is
not allowed for any y.
C          (see ierror = 13)
C
C      (2)    the kbdy=2 boundary
C
C          this is the edge x=xb where nxb=iparm(3) = 2
flags
C          a mixed boundary condition of the form
C
C          alfxb(y)*px + betxb(y)*py +
gamxb(y)*p(xb,y) = gbdxb(y)
C
C          in this case kbdy=2,xory=y will be input to

```

```

bndyc and
c           alfa,beta,gama,gbdy corresponding to
alfa xb(y), betxb(y), gamxb(y),
c           gbdxb(y) must be returned. alfa xb(y) = 0.0 is
not allowed for any y.
c           (see ierror = 13)
c
c      (3)   the kbdy=3 boundary
c
c           this is the edge y=yc where nyc=iparm(4) = 2
flags
c           a mixed boundary condition of the form
c
c           alfy c(x)*px + bety c(x)*py +
gamyc(x)*p(x,yc) = gbdyc(x)
c
c           in this case kbdy=3,xory=x will be input to
bndyc and
c           alfa,beta,gama,gbdy corresponding to
alfy c(x), bety c(x), gamyc(x),
c           gbdyc(x) must be returned. bety c(x) = 0.0 is
not allowed for any x.
c           (see ierror = 13)
c
c      (4)   the kbdy=4 boundary
c
c           this is the edge y=yd where nyd=iparm(5) = 2
flags
c           a mixed boundary condition of the form
c
c           alfy d(x)*px + bety d(x)*py +
gamyd(x)*p(x,yd) = gbdyd(x)
c
c           in this case kbdy=4,xory=x will be input to
bndyc and
c           alfa,beta,gama,gbdy corresponding to
alfy d(x), bety d(x), gamyd(x),
c           gbdyd(x) must be returned. bety d(x) = 0.0 is
not allowed for any x.
c           (see ierror = 13)
c
c
c ***      bndyc must provide the mixed boundary
condition values

```

```
c           in correspondence with those flagged in
iparm(2) thru
c           iparm(5). if all boundaries are specified or
periodic
c           mud2cr will never call bndyc. even then it
must be entered
c           as a dummy subroutine. bndyc must be declared
"external"
c           in the routine calling mud2cr. the actual
name chosen may
c           be different.

c
c
c ... coef
c
c           a subroutine with arguments
(x,y,cxx,cxy,cyy,cx,cy,ce) which
c           provides the known real coefficients for the
elliptic pde at
c           any grid point (x,y). the name chosen in the
calling routine
c           may be different where the coefficient routine
must be declared
c           "external."
c
c ... rhs
c
c           an array dimensioned nx by ny which contains
the given
c           right hand side values on the uniform 2-d
mesh.
c
c           rhs(i,j) = r(xi,yj) for i=1,...,nx and
j=1,...,ny
c
c ... phi
c
c           an array dimensioned nx by ny. on input phi
must contain
c           specified boundary values. for example, if
nyd=iparm(5)=1
c           then phi(i,ny) must be set equal to p(xi,yd)
for i=1,...nx
c           prior to calling mud2cr. these values are
```

```
preserved by mud2cr.  
c           if an initial guess is provided  
(iguess=iparm(11)=1) it must  
c           be input thru phi.  
c  
c  
c ***      if no initial guess is given (iguess=0) then  
phi must still  
c           be initialized at all grid points (this is  
not checked). these  
c           values will serve as an initial guess to the  
pde at the coarsest  
c           grid level after a transfer from the fine  
solution grid. set phi  
c           equal to to 0.0 at all internal and non-  
specified boundaries  
c           grid points if nothing better is available.  
c  
c  
c ... mgopt  
c  
c           an integer vector of length 4 which allows  
the user to select  
c           among various multigrid options. if  
mgopt(1)=0 is input then  
c           a default set of multigrid arguments (chosen  
for robustness)  
c           will be internally selected and the  
remaining values in mgopt  
c           will be ignored. if mgopt(1) is nonzero  
then the arguments  
c           in mgopt are set internally and defined as  
follows: (see the  
c           basic coarse grid correction algorithm  
below)  
c  
c  
c     kcycle = mgopt(1)  
c  
c           = 0 if default multigrid options are to be  
used  
c  
c           = 1 if v cycling is to be used (the least  
expensive per cycle)
```

```

c
c           = 2 if w cycling is to be used (the
default)
c
c           > 2 if more general k cycling is to be used
c           *** warning--values larger than 2 increase
c           the execution time per cycle
considerably and
c           result in the nonfatal error ierror =
-5
c           which indicates inefficient multigrid
cycling.
c
c     iprер = mgopt(2)
c
c           the number of "pre-relaxation" sweeps
executed before the
c           residual is restricted and cycling is
invoked at the next
c           coarser grid level (default value is 2
whenever mgopt(1)=0)
c
c     ipost = mgopt(3)
c
c           the number of "post relaxation" sweeps
executed after cycling
c           has been invoked at the next coarser grid
level and the residual
c           correction has been transferred back
(default value is 1
c           whenever mgopt(1)=0) .
c
c *** if iprер, ipost, or (especially) kcycle is greater
than 2
c           than inefficient multigrid cycling has probably
been chosen and
c           the nonfatal error (see below) ierror = -5 will be
set. note
c           this warning may be overridden by any other
nonzero value
c           for ierror.
c
c     interpol = mgopt(4)
c

```

```

c      = 1 if multilinear prolongation
(interpolation) is used to
c      transfer residual corrections and the pde
approximation
c      from coarse to fine grids within full
multigrid cycling.
c
c      = 3 if multicubic prolongation
(interpolation) is used to
c      transfer residual corrections and the pde
approximation
c      from coarse to fine grids within full
multigrid cycling.
c      (this is the default value whenever
mgopt(1)=0).
c
c *** the default values (2,2,1,3) in the vector mgopt
were chosen for
c      robustness. in some cases v(2,1) cycles with
linear prolongation will
c      give good results with less computation
(especially in two-dimensions).
c      this was the default and only choice in an
earlier version of mudpack
c      (see [1]) and can be set with the integer vector
(1,2,1,1) in mgopt.
c
c *** the schedules for one full multigrid cycle
(iguess=0) using v(2,1)
c      cycles and w(2,1) cycles are depicted for a four
level grid below.
c      the number of relaxation sweeps when each grid is
visited are indicated.
c      the "*" stands for prolongation of the full
approximation and the "."
c      stands for transfer of residuals and residual
corrections within the
c      coarse grid correction algorithm (see below). all
version 5.0.1
c      mudpack solvers use only fully weighted residual
restriction
c
c      one fmg with v(2,1) cycles:
c

```

```

C
C      -----2-----1-
----      level 4
C          * .
C          *
C          .
C      -----2-----1-----2-----1-----
----      level 3
C          * .
C          *
C          .
C      -----2-----1-----2-----1-----2-----1-----
----      level 2
C          * .
C          *
C          .
C      ---3---3-----3-----3-----
----      level 1
C
C
C      one fmg with w(2,1) cycles:
C
C      -----2-----
--1--      level 4
C          *
C          .
C      -----2-----1-----2-----3-----
1----      level 3
C          *
C          .
C      ----2---1---2---3---1-----2---3---1---2---3---1-
-----      level 2
C          *   .   .   .   .   .   .   .   .   .   .
C      --6---6-----6---6-----6---6-----6---6-----6---6---
-----      level 1
C
C
C      the form of the "recursive" coarse grid correction
cycling used
C      when kcycle.ge.0 is input is described below in
pseudo-algorithmic
C      language. it is implemented non-recursively in
fortran in mudpack.
C
C      algorithm
cgc(k,l(k),u(k),r(k),kcycle,iprer,ipost,iresw,intpol)
C
C *** approximately solve l(k)*u(k) = r(k) using

```

```

multigrid iteration
c *** k is the current grid level
c *** l(k) is the discretized pde operator at level k
c *** u(k) is the initial guess at level k
c *** r(k) is the right hand side at level k
c *** i(k,k-1) is the restriction operator from level k
to level k-1
c *** (the form of i(k,k-1) depends on iresw)
c *** i(k-1,k) is the prolongation operator from level
k-1 to level k
c *** (the form of i(k-1,k) depends on interpol)
c
c      begin algorithm cgc
c
c ***      pre-relax at level k
c
c      . do (i=1,iprer)
c
c      . . relax(l(k),u(k),r(k))
c
c      . end do
c
c      . if (k > 1) then
c
c ***      restrict the residual from level k to level k-
1
c
c      . . r(k-1) = i(k,k-1) (r(k)-l(k)*u(k))
c
c      . . kount = 0
c
c      . . repeat
c
c ***      solve for the residual correction at level k-1
in u(k-1)
c ***      using algorithm cgc "kcycle" times (this is
the recursion)
c
c      . . . kount = kount+1
c
c      . . . invoke cgc(k-1,l(k-1),u(k-1),r(k-
1),kcycle,iprer,ipost,iresw)
c
c

```

```

C      . . until (kount.eq.kcycle)
C
C ***      transfer residual correction in u(k-1) to
level k
C ***      with the prolongation operator and add to u(k)
C
C      . . u(k) = u(k) + i(k-1,k) (u(k-1))
C
C      . end if
C
C ***      post relax at level k
C
C      . do (i=1,ipost)
C
C      . . relax(l(k),u(k),r(k))
C
C      . end do
C
C      . return
C
C
C      end algorithm cgc
C
C
C
C ****
*****
C *** output arguments
*****
C
*****
C
C ... iparm(16) *** set for intl=0 calls only
C
C          on output iparm(16) contains the actual work
space length
C          required. this will usually be less than
that given by the
C          simplified formula for length=iparm(15) (see
as input argument)
C
C
C ... iparm(17) *** set for intl=1 calls only

```

```

C
C           on output iparm(17) contains the actual
number of multigrid cycles
C           between the finest and coarsest grid levels
used to obtain the
C           approximation when error control (tolmax >
0.0) is set.

C
C
C ... fparm(6) *** set for intl=1 calls with fparm(5)
> 0. only
C
C           on output fparm(6) contains the final
computed maximum relative
C           difference between the last two iterates at
the finest grid level.
C           fparm(6) is computed only if there is error
control (tolmax > 0.0)
C           assume phi1(i,j,k) and phi2(i,j,k) are the
last two computed
C           values for phi(i,j,k) at all points of the
finest grid level.
C           if we define
C
C           phdif = max(abs(phi2(i,j)-phi1(i,j)))
over all i,j
C
C           and
C
C           phmax = max(abs(phi2(i,j))) over all i,j
C
C           then
C
C           fparm(6) = phdif/phmax
C
C           is returned whenever phmax > 0.0. in the
degenerate case
C           phmax = 0.0, fparm(6) = phdif is returned.
C
C
C ... work
C
C           on output work contains intermediate values
that must not

```

```

c           be destroyed if mud2cr is to be called again
with intl=1
c
c
c ... phi    *** for intl=1 calls only
c
c           on output phi(i,j) contains the approximation
to p(xi,yj)
c           for all mesh points i = 1,...,nx and
j=1,...,ny. the last
c           computed iterate in phi is returned even if
convergence is
c           not obtained
c
c
c ... ierror
c
c           for intl=iparm(1)=0 initialization calls,
ierror is an
c           error flag that indicates invalid input
arguments when
c           returned positive and nonfatal warnings when
returned
c           negative. argument checking and
discretization
c           is bypassed for intl=1 calls which can only
return
c           ierror = -1 or 0 or 1.
c
c
c           non-fatal warnings * * *
c
c
c     ==-5 if kcycle=mgopt(1) is greater than 2. values
larger than 2 results
c           in an algorithm which probably does far more
computation than
c           necessary. kcycle = 1 (v cycles) or kcycle=2
(w cycles) should
c           suffice for most problems. ierror = -5 is
also set if either
c           iprер = mgopt(2) or ipost=mgopt(3) is greater
than 2. the
c           ierror=-5 flag is overridden by any other

```

```

fatal or non-fatal
c           error.

c
c      ==4 if there are dominant nonzero first order
terms in the pde which
c          make it "hyperbolic" at the finest grid level.
numerically, this
c          happens if:
c
c          abs(cx)*dlx > 2.*abs(cxx)    (dlx = (xb-
xa) / (nx-1))
c
c                      (or)
c
c          abs(cy)*dly > 2.*abs(cyy)    (dly = (yd-
yc) / (ny-1))
c
c
c          at some fine grid point (xi,yj). if an
adjustment is not made the
c          condition can lead to a matrix coming from the
discretization
c          which is not diagonally dominant and
divergence is possible. since
c          the condition is "likely" at coarser grid
levels for pde's with
c          nonzero first order terms, the adjustments
(actually first order
c          approximations to the pde)

c
c
c          cxx = amax1(cxx,0.5*abs(cx)*dx)
c
c                      (and)
c
c          cyy = amax1(cyy,0.5*abs(cy)*dy)
c
c
c          (here dx,dy are the x,y mesh sizes of the
subgrid)
c
c          are made to preserve convergence of multigrid
iteration. if made
c          at the finest grid level, it can lead to

```

```
convergence to an
c           erroneous solution (flagged by ierror = -4).
a possible remedy
c           is to increase resolution. the ierror = -4
flag overrides the
c           nonfatal ierror = -5 flag.

c
c
c     ==3 if the continuous elliptic pde is singular.
this means the
c           boundary conditions are periodic or pure
derivative at all
c           boundaries and ce(x,y) = 0.0 for all x,y. a
solution is still
c           attempted but convergence may not occur due
to ill-conditioning
c           of the linear system coming from the
discretization. the
c           ierror = -3 flag overrides the ierror=-4 and
ierror=-5 nonfatal
c           flags.

c
c
c     ==2 if the pde is not elliptic (i.e.,
cxx*cyy.le.0.0 for some (xi,yj))
c           in this case a solution is still attempted
although convergence
c           may not occur due to ill-conditioning of the
linear system.
c           the ierror = -2 flag overrides the ierror=-
5,-4,-3 nonfatal
c           flags.

c
c
c     ==1 if convergence to the tolerance specified in
tolmax=fparm(5)>0.
c           is not obtained in maxcy=iparm(13) multigrid
cycles between the
c           coarsest (ixp+1,jyq+1) and finest (nx,ny)
grid levels.
c           in this case the last computed iterate is
still returned.
c           the ierror = -1 flag overrides all other
nonfatal flags
```

```

C
C
C      no errors * * *
C
C      = 0
C
C      fatal argument errors * * *
C
C      = 1 if intl=iparm(1) is not 0 on initial call or
not 0 or 1
C          on subsequent calls
C
C      = 2 if any of the boundary condition flags
nxa,nxb,nyc,nyd
C          in iparm(2),iparm(3),iparm(4),iparm(5) are not
0,1 or 2
C          or if nxa,nxb or nyc,nyd are not pairwise
zero.
C
C      = 3 if min0(ixp,jyq) < 2 (ixp = iparm(6), jyq =
iparm(7))
C
C      = 4 if min0(iex,jey) < 1 (iex = iparm(8), jey =
iparm(9)) or
C          if max0(iex,jey) > 50
C
C      = 5 if nx.ne.ipx*2** (iex-1)+1 or
ny.ne.jyq*2** (jey-1)+1
C          (nx = iparm(10), ny = iparm(11))
C
C      = 6 if iguess = iparm(12) is not equal to 0 or 1
C
C      = 7 if maxcy = iparm(13) < 1
C
C      = 8 if method = iparm(14) is not 0,1,2, or 3
C
C      = 9 if length = iparm(15) is too small (see
iparm(16) on output
C          for minimum required work space length)
C
C      =10 if xa >= xb or yc >= yd
C
(xa=fparm(1),xb=fparm(2),yc=fparm(3),yd=fparm(4))
C

```

```

c      =11 if tolmax = fparm(5) < 0.0
c
c      errors in setting multigrid options * * * (see
also ierror=-5)
c
c      =12 if kcycle = mgopt(1) < 0 or
c          if iprer = mgopt(1) < 1 or
c          if ipost = mgopt(3) < 1 or
c          if interpol = mgopt(4) is not 1 or 3
c
c      =13 if there is a pure tangential derivative along
a mixed derivative
c          boundary (e.g., nyd = 2 and betyd(x) = 0.0 for
some
c          grid point x along y = yd)
c
c      =14 if there is the "singular" condition described
below at a
c          corner which is the intersection of two
derivative boundaries.
c
c          (1) the corner (xa,yc) if nxa=nyc=2 and
c              alfxa(yc)*betyc(xa)-alfyc(xa)*betxa(yc) =
0.0.
c
c          (2) the corner (xa,yd) if nxa=nyd=2 and
c              alfxa(yd)*betyd(xa)-alfyd(xa)*betxa(yd) =
0.0.
c
c          (3) the corner (xb,yc) if nxb=nyc=2 and
c              alfxb(yc)*betyc(xb)-alfyc(xb)*betxb(yc) =
0.0.
c
c          (4) the corner (xb,yd) if nxb=nyd=2 and
c              alfxb(yd)*betyd(xb)-alfyd(xb)*betxb(yd) =
0.0.
c
c *** the conditions described in ierror = 13 or 14 will
lead to division
c      by zero during discretization if undetected.
c
c
c
*****
```

```
*  
C  
*****  
*  
C  
C      end of mud2cr documentation  
C  
C  
*****  
**  
C  
*****  
**  
C  
C
```

---

## MUD2SA

```
C  
C      file mud2sa.d  
C  
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C      *  
*  
C      *          copyright (c) 2008 by UCAR  
*  
C      *  
*  
C      *          University Corporation for Atmospheric  
Research           *  
C      *  
*  
C      *          all rights reserved  
*  
C      *  
*  
C      *          MUDPACK version 5.0.1
```

\*  
C      \*  
\*  
C      \*                  A Fortran Package of Multigrid  
\*  
C      \*  
\*  
C      \*                  Subroutines and Example Programs  
\*  
C      \*  
\*  
C      \*                  for Solving Elliptic Partial Differential  
Equations                \*  
C      \*  
\*  
C      \*                  by  
\*  
C      \*  
\*  
C      \*                  John Adams  
\*  
C      \*  
\*  
C      \*                  of  
\*  
C      \*  
\*  
C      \*                  the National Center for Atmospheric  
Research                \*  
C      \*  
\*  
C      \*                  Boulder, Colorado (80307)  
U.S.A.                \*  
C      \*  
\*  
C      \*                  which is sponsored by  
\*  
C      \*  
\*  
C      \*                  the National Science Foundation  
\*  
C      \*  
\*  
C      \*                  \* \* \* \* \*

```

* * * * *
C
C ... file mud2sa.d
C
C      contains documentation for:
C      subroutine
mud2sa(iparm,fparm,work,sigx,sigy,bndyc,rhs,phi,mgopt,ie
rror)
C      a sample fortran driver is file "tmud2sa.f".
C
C ... required mudpack files
C
C      mudcom.f
C
C
C ... purpose
C
C      subroutine mud2sa automatically discretizes and
attempts to
C      compute the second order conservative finite
difference approximation
C      to a two dimensional linear nonseparable "self
adjoint" elliptic
C      partial differential equation on a rectangle.  the
approximation
C      is generated on a uniform grid covering the
rectangle.  boundary
C      conditions may be specified (Dirchlet), periodic,
or mixed.
C      the form of the pde solved is:
C
C      d(sigx(x,y)*dp/dx)/dx + 
d(sigy(x,y)*dp/dy)/dy -
C
C      xlmbda(x,y)*p(x,y) = r(x,y)
C
C      where sigx(x,y),sigy(x,y) (both positive),
xlmbda(x,y) (non-negative)
C      r(x,y) (the given right hand side) and p(x,y) (the
unknown solution
C      function) are all real valued functions of the
real independent
C      variables x,y. the use of the variable names "x,y"
is arbitrary and

```

```
c      does not imply the cartesian coordinate system
underlies the pde.
c      for example, any pde in divergence form in
cartesian coordinates can
c      be put in a self-adjoint form suitable for mud2sa
after a curvilinear
c      coordinate transform (see tmud2sa.f)
c
c
c ... mesh description . . .
c
c      the approximation is generated on a uniform nx by
ny grid.  the grid
c      is superimposed on the rectangular solution region
c
c            [xa,xb] x [yc,yd].
c
c      let
c
c      dlx = (xb-xa) / (nx-1), dly = (yd-yc) / (ny-1)
c
c      be the uniform grid increments in the x,y
directions. then
c
c            xi=xa+(i-1)*dlx,   yj=yc+(j-1)*dly
c
c      for i=1,...,nx and j=1,...,ny denote the x,y
uniform mesh points
c
c
c ... language
c
c      fortran90/fortran77
c
c
c ... portability
c
c      mudpack5.0.1 software has been compiled and tested
with fortran77
c      and fortran90 on a variety of platforms.
c
c ... methods
c
c      details of the methods employed by the solvers in
```

```
mudpack are given
c      in [1,9]. [1,2,9] contain performance
measurements on a variety of
c      elliptic pdes (see "references" in the file
"readme"). in summary:
c
c *** discretization and solution (second-order solvers)
(see [1])
c
c      the pde and boundary conditions are automatically
discretized at all
c      grid levels using second-order finite difference
formula. diagonal
c      dominance at coarser grid levels is maintained in
the presence of
c      nonzero first-order terms by adjusting the second-
order coefficient
c      when necessary. the resulting block tri-diagonal
linear system is
c      approximated using multigrid iteration
[10,11,13,15,16,18]. version
c      5.0.1 of mudpack uses only fully weighted residual
restriction. defaults
c      include cubic prolongation and w(2,1) cycles.
these can be overridden
c      with selected multigrid options (see "mgopt").
error control based on
c      maximum relative differences is available. full
multigrid cycling (fmg)
c      or cycling beginning or restarting at the finest
grid level can be
c      selected. a menu of relaxation methods including
gauss-seidel point,
c      line relaxation(s) (in any combination of
directions) and planar
c      relaxation (for three-dimensional anisotropic
problems) are provided.
c      all methods use ordering based on alternating
points (red/black),
c      lines, or planes for cray vectorization and
improved convergence
c      rates [14].
c
c *** higher order solution (fourth-order solvers) (see
```

```
[9,19,21])  
c  
c      if the multigrid cycling results in a second-order  
estimate (i.e.,  
c      discretization level error is reached) then this  
can be improved to a  
c      fourth-order estimate using the technique of  
"deferred corrections."  
c      the values in the solution array are used to  
generate a fourth-order  
c      approximation to the truncation error. second-  
order finite difference  
c      formula are used to approximate third and fourth  
partial derivatives  
c      of the solution function [3]. the truncation  
error estimate is  
c      transferred down to all grid levels using weighted  
averaging where  
c      it serves as a new right hand side. the default  
multigrid options  
c      are used to compute the fourth-order correction  
term which is added  
c      to the original solution array.  
c  
c  
c ... references (partial)  
c  
c  
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software  
for the Efficient  
c      Solution of Linear Elliptic Partial Differential  
Equations,"  
c      Applied Math. and Comput. vol.34, Nov 1989,  
pp.113-146.  
c  
c      [2] J. Adams, "FMG Results with the Multigrid  
Software Package MUDPACK,"  
c      proceedings of the fourth Copper Mountain  
Conference on Multigrid, SIAM,  
c      1989, pp.1-12.  
c      .  
c      .  
c      .  
c      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
```

Haidvogel, and V. Pizzo,  
c        "Applications of Multigrid Software in the  
Atmospheric Sciences,"  
c        Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-  
1458.  
c        .  
c        .  
c        .  
c        [9] J. Adams, "Recent Enhancements in MUDPACK, a  
Multigrid Software  
c        package for Elliptic Partial Differential  
Equations," Applied Math.  
c        and Comp., 1991, vol. 43, May 1991, pp. 79-94.  
c  
c        [10] J. Adams, "MUDPACK-2: Multigrid Software for  
Approximating  
c        Elliptic Partial Differential Equations on Uniform  
Grids with  
c        any Resolution," Applied Math. and Comp., 1993,  
vol. 53, February  
c        1993, pp. 235-249  
c        .  
c        .  
c        .  
c  
c   ... argument description  
c  
c  
c  
\*\*\*\*\*  
\*\*\*\*\*  
c \*\*\* input arguments  
\*\*\*\*\*  
c  
\*\*\*\*\*  
\*\*\*\*\*  
c  
c  
c   ... iparm  
c  
c        an integer vector of length 17 used to pass  
integer  
c        arguments. iparm is set internally and  
defined as

```
c follows:
c
c
c ... intl=iparm(1)
c
c           an initialization argument. intl=0 must be
input
c           on an initial call. in this case input
arguments will
c           be checked for errors and the elliptic
partial differential
c           equation and boundary conditions will be
discretized using
c           second order finite difference formula.
c
c ***      an approximation is not generated after an
intl=0 call!
c           mud2sa should be called with intl=1 to
approximate the elliptic
c           pde discretized by the intl=0 call. intl=1
should also
c           be input if mud2sa has been called earlier
and only the
c           values in in rhs (see below) or gbdy (see
bndyc below)
c           or phi (see below) have changed. this will
bypass
c           redundant pde discretization and argument
checking
c           and save computational time. some examples
of when
c           intl=1 calls should be used are:
c
c           (0) after a intl=0 argument checking and
discretization call
c
c           (1) mud2sa is being recalled for additional
accuracy. in
c           this case iguess=iparm(12)=1 should also
be used.
c
c           (2) mud2sa is being called every time step in
a time dependent
c           problem (see discussion below) where the
```

```
elliptic operator
c           does not depend on time.
c
c           (3) mud2sa is being used to solve the same
elliptic equation
c           for several different right hand sides
(iguess=0 should
c           probably be used for each new righthand
side).
c
c           intl = 0 must be input before calling with
intl = 1 when any
c           of the following conditions hold:
c
c           (a) the initial call to mud2sa
c
c           (b) any of the integer arguments other than
iguess=iparm(12)
c           or maxcy=iparm(13) or mgopt have changed
since the previous
c           call.
c
c           (c) any of the floating point arguments other
than tolmax=
c           fparm(5) have changed since the previous
call
c
c           (d) any of the coefficients input by coef
(see below) have
c           changed since the previous call
c
c           (e) any of the "alfa" coefficients input by
bndyc (see below)
c           have changed since the previous call.
c
c           if any of (a) through (e) are true then the
elliptic pde
c           must be discretized or rediscretized. if
none of (a)
c           through (e) holds, calls can be made with
intl=1.
c           incorrect calls with intl=1 will produce
erroneous results.
c ***      the values set in the saved work space "work"
```

```

(see below) with
c           an intl=0 call must be preserved with
subsequent intl=1 calls.
c
c           MUDPACK software performance should be
monitored for intl=1
c           calls.  The intl=0 discretization call
performance depends
c           primarily on the efficiency or lack of
efficiency of the
c           user provided subroutines for pde
coefficients and
c           boundary conditions.

c
c
c ... nxa=iparm(2)
c
c           flags boundary conditions on the edge x=xa
c
c           = 0 if p(x,y) is periodic in x on [xa,xb]
c               (i.e., p(x+xb-xa,y) = p(x,y) for all x,y
c               (if nxa=0 then nxb=0 is required, see
ierror = 2)
c
c           = 1 if p(xa,y) is specified (this must be input
thru phi(1,j))
c
c           = 2 if there are mixed derivative boundary
conditions at x=xa
c           (see bndyc)
c
c
c ... nxb=iparm(3)
c
c           flags boundary conditions on the edge x=xb
c
c           = 0 if p(x,y) is periodic in x on [xa,xb]
c               (i.e., p(x+xb-xa,y) = p(x,y) for all x,y
c               (if nxb=0 then nxa=0 is required, see
ierror = 2)
c
c           = 1 if p(xb,y) is specified (this must be input
thru phi(nx,j))
c

```

```

c      = 2 if there are mixed derivative boundary
conditions at x=xb
c          (see bndyc)
c
c
c ... nyc=iparm(4)
c
c      flags boundary conditions on the edge y=yc
c
c      = 0 if p(x,y) is periodic in y on [yc,yd]
c          (i.e., p(x,y+yd-yc) = p(x,y) for all x,y
c          (if nyc=0 then nyd=0 is required, see
ierror = 2)
c
c      = 1 if p(x,yc) is specified (this must be input
thru phi(i,1))
c
c      = 2 if there are mixed derivative boundary
conditions at y=yc
c          (see bndyc)
c
c
c ... nyd=iparm(5)
c
c      flags boundary conditions on the edge y=yd
c
c      = 0 if p(x,y) is periodic in y on [yc,yd]
c          (i.e., p(x,y+yd-yc) = p(x,y) for all x,y
c          (if nyd=0 then nyc=0 is required, see
ierror = 2)
c
c      = 1 if p(x,yd) is specified (this must be input
thru phi(i,ny))
c
c      = 2 if there are mixed derivative boundary
conditions at y=yd
c          (see bndyc)
c
c
c *** grid size arguments
c
c
c ... ixp = iparm(6)
c

```

```
c      an integer greater than one which is used in
defining the number
c      of grid points in the x direction (see nx =
iparm(10)). "ixp+1"
c      is the number of points on the coarsest x
grid visited during
c      multigrid cycling. ixp should be chosen as
small as possible.
c      recommended values are the small primes 2 or
3.
c      larger values can reduce multigrid
convergence rates considerably,
c      especially if line relaxation in the x
direction is not used.
c      if ixp > 2 then it should be 2 or a small odd
value since a power
c      of 2 factor of ixp can be removed by
increasing iex = iparm(8)
c      without changing nx = iparm(10).
c
c
c ... jyq = iparm(7)
c
c      an integer greater than one which is used in
defining the number
c      of grid points in the y direction (see ny =
iparm(11)). "jyq+1"
c      is the number of points on the coarsest y
grid visited during
c      multigrid cycling. jyq should be chosen as
small as possible.
c      recommended values are the small primes 2 or
3.
c      larger values can reduce multigrid
convergence rates considerably,
c      especially if line relaxation in the y
direction is not used.
c      if jyq > 2 then it should be 2 or a small odd
value since a power
c      of 2 factor of jyq can be removed by
increasing jey = iparm(9)
c      without changing ny = iparm(11).
c
c
```

```

c ... iex = iparm(8)
c
c           a positive integer exponent of 2 used in
defining the number
c           of grid points in the x direction (see nx =
iparm(10)).
c           iex .le. 50 is required. for efficient
multigrid cycling,
c           iex should be chosen as large as possible and
ixp=iparm(8)
c           as small as possible within grid size
constraints when
c           defining nx.

c
c
c ... jey = iparm(9)
c
c           a positive integer exponent of 2 used in
defining the number
c           of grid points in the y direction (see ny =
iparm(11)).
c           jey .le. 50 is required. for efficient
multigrid cycling,
c           jey should be chosen as large as possible and
jyq=iparm(7)
c           as small as possible within grid size
constraints when
c           defining ny.

c
c
c
c ... nx = iparm(10)
c
c           the number of equally spaced grid points in
the interval [xa,xb]
c           (including the boundaries). nx must have the
form
c
c           nx = ixp*(2** (iex-1)) + 1
c
c           where ixp = iparm(6), iex = iparm(8).
c
c
c ... ny = iparm(11)

```

```

c
c      the number of equally spaced grid points in
the interval [yc,yd]
c      (including the boundaries). ny must have the
form:
c
c      ny = jyq* (2** (jey-1)) + 1
c
c      where jyq = iparm(7), jey = iparm(9).
c
c
c *** example
c
c      suppose a solution is wanted on a 33 by 97
grid. then
c      ixp=2, jyq=6 and iex=jey=5 could be used. a
better
c      choice would be ixp=2, jyq=3, and iex=5,
jey=6.
c
c *** grid size flexibility considerations:
c
c      the hybrid multigrid/direct method code muh2
provides more grid size
c      flexibility than mud2sa by removing the constraint
that ixp and jyq are
c      2 or 3. this is accomplished by using a direct
method whenever the
c      coarsest (ixp+1) x (jyq+1) grid is encountered in
multigrid cycling.
c      if nx = ixp+1 and ny = jyq+1 then muh2 becomes a
full direct method.
c      muh2 is roughly equivalent to mud2sa in efficiency
as long as ixp and
c      jyq remain "small" (see muh2.d). if the problem
to be approximated
c      requires a grid neither mud2sa por muh2 can
exactly fit then another option
c      is to generate an approximation on a "close grid"
using mud2sa or muh2.
c      then transfer the result to the required grid
using cubic interpolation
c      via the package "regridpack" (contact john adams
about this software)

```

```

C
C *** note
C
C      let g be the nx by ny fine grid on which the
C      approximation is
C      generated and let n = max0(iex,jey).  in mudpack,
C      multigrid
C      cycling is implemented on the ascending chain of
C      grids
C
C          g(1) < ... < g(k) < ... < g(n) = g.
C
C      each g(k) (k=1,...,n) has mx(k) by my(k) grid
C      points
C      given by:
C
C          mx(k) = ixp*[2** (max0(iex+k-n,1)-1)] + 1
C
C          my(k) = jyq*[2** (max0(jey+k-n,1)-1)] + 1
C
C
C
C ... iguess=iparm(12)
C
C          = 0 if no initial guess to the pde is
C          provided
C
C          = 1 if an initial guess to the pde is at the
C          finest grid
C          level is provided in phi (see below)
C
C      comments on iguess = 0 or 1 . . .
C
C      even if iguess = 0, phi must be initialized at all
C      grid points (this
C      is not checked).  phi can be set to 0.0 at non-
C      dirchlet grid points
C      if nothing better is available.  the values set in
C      phi when iguess = 0
C      are passed down and serve as an initial guess to
C      the pde at the coarsest
C      grid level where cycling commences.  in this
C      sense, values input in
C      phi always serve as an initial guess.  setting

```

```

iguess = 0 forces full
c      multigrid cycling beginning at the coarsest and
finishing at the finest
c      grid level.
c
c      if iguess = 1 then the values input in phi are an
initial guess to the
c      pde at the finest grid level where cycling begins.
this option should
c      be used only if a "very good" initial guess is
available (as, for
c      example, when restarting from a previous iguess=0
call).
c
c      time dependent problems . . .
c
c *** assume we are solving an elliptic pde every time
step in a
c      marching problem of the form:
c
c      l(p(t)) = r(t)
c
c      where the differential operator "l" has no time
dependence,
c      "p(t)" is the solution and "r(t)" is the right
hand side at
c      current time "t". let "dt" be the increment
between time steps.
c      then p(t) can be used as an initial guess to
p(t+dt) with
c      intl = 1 when solving
c
c      l(p(t+dt)) = r(t+dt) .
c
c      after the first two time steps, rather than
continue, it would
c      be better to define the "correction" term:
c
c      e(t,dt) = p(t+dt) - p(t)
c
c      this clearly satisfies the equation
c
c      l(e(t,dt)) = r(t+dt) - r(t) .
c

```

```

c      this should be solved with iguess = 0 and intl =
1. boundary
c      conditions for e(t,dt) are obtained from the
boundary conditions
c      for p(t) by subtracting given values at t from
given values at
c      t+dt. for example if
c
c           $d(p(t))/dx = f(t)$ ,  $d(p(t+dt))/dx = f(t+dt)$ 
c
c      at some x boundary then e(t,dt) satisfies the
derivative
c      boundary condition
c
c           $d(e(t,dt))/dx = f(t+dt) - f(t)$ .
c
c      e(t,dt) can be preset to 0.0 (at nondirichlet
points) or (if p(t-dt)
c      is saved) to p(t)-p(t-dt). with iguess = 0, these
values will serve
c      as an initial guess to e(t,dt) at the coarsest
grid level. this
c      approach has the advantage that a full sequence of
multigrid cycles,
c      beginning at the coarsest grid level, is invoked
every time step in
c      solving for e(t,dt). a few digits of accuracy in
e(t,dt), which is
c      ordinarily much smaller than p(t), will yield
several more digits of
c      accuracy in the final approximation:
c
c           $p(t+dt) = p(t) + e(t,dt)$ .
c
c      using this approach to integrate in time will give
more accuracy
c      then using p(t) as an initial guess to p(t+dt) for
all time steps.
c      it does require additional storage.
c
c      if the differential operator "l" has time
dependence (either thru
c      the coefficients in the pde or the coefficients in
the derivative

```

```

c      boundary conditions) then use p(t) as an initial
guess to p(t+dt)
c      when solving
c
c      l(t+dt) (p(t+dt)) = r(t+dt)
c
c      with intl = 0 for all time steps (the
discretization must be repeated
c      for each new "t"). either iguess = 0 (p(t) will
then be an initial
c      guess at the coarsest grid level where cycles will
commence) or
c      iguess = 1 (p(t) will then be an initial guess at
the finest grid
c      level where cycles will remain fixed) can be
tried.
c
c
c ... maxcy = iparm(13)
c
c      the exact number of cycles executed between
the finest (nx by
c      ny) and the coarsest ((ixp+1) by (jyq+1))
grid levels when
c      tolmax=fparm(5)=0.0 (no error control). when
tolmax > 0.0
c      is input (error control) then maxcy is a
limit on the number
c      of cycles between the finest and coarsest
grid levels. in
c      any case, at most maxcy*(iprert+ipost)
relaxation sweeps are
c      are performed at the finest grid level (see
iprert=mgopt(2),
c      ipost=mgopt(3) below). when multigrid
iteration is working
c      "correctly" only a few are required for
convergence. large
c      values for maxcy should not be necessary.
c
c
c ... method = iparm(14) determines the method of
relaxation
c      (gauss-seidel based on alternating points

```

```

or lines)
C
C      = 0 for point relaxation
C
C      = 1 for line relaxation in the x direction
C
C      = 2 for line relaxation in the y direction
C
C      = 3 for line relaxation in both the x and y
direction
C
C
C *** choice of method. . .
C
C      let fx represent the quantity sigx(x,y)/dlx**2
over the solution region.
C
C      let fy represent the quantity sigy(x,y)/dly**2
over the solution region
C
C      if fx,fy are roughly the same size and do not vary
too much over
C      the solution region choose method = 0. if this
fails try method=3.
C
C      if fx is much greater than fy choose method = 1.
C
C      if fy is much greater than fx choose method = 2
C
C      if neither fx or fy dominates over the solution
region and they
C      both vary considerably choose method = 3.
C
C
C ... length = iparm(15)
C
C      the length of the work space provided in
vector work (see below).
C      let isx = 0 if method = 0 or method = 2
C      let isx = 3 if method = 1 or method = 3 and
nxa.ne.0
C      let isx = 5 if method = 1 or method = 3 and
nxa.eq.0
C      let jsy = 0 if method = 0 or method = 1

```

```
c           let jsy = 3 if method = 2 or method = 3 and
nyc.ne.0
c           let jsy = 5 if method = 2 or method = 3 and
nyc.eq.0
c           then . . .
c
c           length =
4*[nx*ny*(10+isxt+jsy)+8*(nx+ny+2)]/3
c
c           will suffice in most cases. the exact
minimal work space
c           length required for the current nx,ny and
method is output
c           in iparm(16) (even if iparm(15) is too
small). this will be
c           less then the value given by the simplified
formula above
c           in most cases.
c
c
c ... fparm
c
c           a floating point vector of length 6 used to
efficiently
c           pass floating point arguments. fparm is set
internally
c           in mud2sa and defined as follows . . .
c
c
c ... xa=fparm(1), xb=fparm(2)
c
c           the range of the x independent variable. xa
must
c           be less than xb
c
c
c ... yc=fparm(3), yd=fparm(4)
c
c           the range of the y independent variable. yc
must
c           be less than yd.
c
c
c ... tolmax = fparm(5)
```

```

c
c           when input positive, tolmax is a maximum
relative error tolerance
c           used to terminate the relaxation iterations.
assume phi1(i,j)
c           and phi2(i,j) are the last two computed
approximations at all
c           grid points of the finest grid level. if we
define
c
c           phdif = max(abs(phi2(i,j)-phi1(i,j))) for
all i,j
c
c           and
c
c           phmax = max(abs(phi2(i,j))) for all i,j
c
c           then "convergence" is considered to have
occurred if and only if
c
c           phdif/phmax < tolmax.
c
c
c           if tolmax=fparm(5)=0.0 is input then there is
no error control
c           and maxcy cycles from the finest grid level
are executed. maxcy
c           is a limit which cannot be exceeded even with
error control.
c           *** calls with tolmax=0.0, when appropriate
because of known
c           convergence behavior, are more efficient than
calls with tolmax
c           positive (i.e., if possible do not use error
control!).
c
c ... work
c
c           a one dimensional real saved work space (see
iparm(15) for
c           length) which must be preserved from the
previous call when
c           calling with intl=iparm(1)=1.
c

```

```

c ... bndyc
c
c           a subroutine with arguments
(kbdy,xory,alfa,gbdy) which
c           are used to input mixed boundary conditions
to mud2sa. bndyc
c           must be declared "external" in the program
calling mud2sa.
c           the boundaries are numbered one thru four and
the mixed
c           derivative boundary conditions are described
below (see the
c           sample driver code "tmud2sa.f" for an example
of how bndyc is
c           can beset up).

c
c           * * * * * * * * * * * * * * * * y=yd
c           *         kbdy=4          *
c           *
c           *
c           *
c           *
c           * kbdy=1          kbdy=2  *
c           *
c           *
c           *
c           *
c           *         kbdy=3          *
c           * * * * * * * * * * * * * * * * y=yc
c
c           x=xa          x=xb
c
c
c           (1) the kbdy=1 boundary
c
c           this is the edge x=xa where nx=iparm(2)=2
flags
c           a mixed boundary condition of the form
c
c           dp/dx + alfx(x,y)*p(xa,y) = gbdxa(y)
c
c           in this case kbdy=1,xory=y will be input to
bndyc and
c           alfa,gbdy corresponding to alfx(x,y),gbdxa(y)
must be returned.
c

```

```

C
C          (2) the kbdy=2 boundary
C
C          this is the edge x=xb where nxb=iparm(3)=2
flags      a mixed boundary condition of the form
C
C           $\frac{dp}{dx} + \text{alfxb}(y) * p(xb, y) = \text{gbdxb}(y)$ 
C
C          in this case kbdy=2, xory=y, will be input to
bndyc and
C          alfa, gbdy corresponding to alfxb(y), gbdxb(y)
must be returned.

C
C
C          (3) the kbdy=3 boundary
C
C          this is the edge y=yc where nyc=iparm(4)=2
flags      a mixed boundary condition of the form
C
C           $\frac{dp}{dy} + \text{alfyc}(x) * p(x, yc) = \text{gbdyc}(x)$ 
C
C          in this case kbdy=3, xory=x will be input to
bndyc and
C          alfa, gbdy corresponding to alfy(c)(x), gbdyc(x)
must be returned.

C
C
C          (4) the kbdy=4 boundary
C
C          this is the edge y=yd where nyd=iparm(5)=2
flags      a mixed boundary condition of the form
C
C           $\frac{dp}{dy} + \text{alfyd}(x) * p(x, yd) = \text{gbdyd}(x)$ 
C
C          in this case kbdy=4, xory=x will be input to
bndyc and
C          alfa, gbdy corresponding to alfyd(x), gbdyd(x)
must be returned.

C
C
C ***      bndyc must provide the mixed boundary

```

```
condition values
c           in correspondence with those flagged in
iparm(2) thru
c           iparm(5). if all boundaries are specified or
periodic
c           mud2sa will never call bndyc. even then it
must be entered
c           as a dummy subroutine. bndyc must be declared
"external"
c           in the routine calling mud2sa. the actual
name chosen may
c           be different.

c
c
c ... sigx,sigy
c
c           function subroutines which returns the real
value of the
c           coefficients at any point (x,y). they must be
constructed
c           to return values outside the solution region
for nonDirchlet
c           boundaries. Let dx = (xb=xa)/ixp and dy =
(yd-yc)/jyq.
c           then sigx,sigy will be invoked for x,y in the
intervals
c           [xa-0.5*dx,xa], [xb,xb+0.5*dx], [yc-
0.5*dy,yc], [yd,yd+0.5*dy]
c           whenever boundary conditions at
x=xa,x=xb,y=yc,y=yd are unspecified.
c           this is necessitated by conservative finite
differencing. sigx,
c           sigy will not be invoked outside specified
boundaries. sigx,
c           sigy should be positive for all (x,y) (see
ierror = -2). they
c           must be declared "external" in the user
constructed program calling
c           mud2sa where their names may be different.

c
c ... xlmbda
c
c           a real valued function subroutine which
returns the value
```

```

c      of "xlmbda" in the pde at any grid point
(xi,yj). xlmbda should
c      be nonnegative for any (xi,yj) (see ierror = -
4). xlmbda must be
c      declared "external" in the user constructed
program calling
c      mud2sa where its name may be different.
c
c
c ... rhs
c
c      an array dimensioned nx by ny which contains
the given
c      right hand side values on the uniform 2-d
mesh.
c
c      rhs(i,j) = r(xi,yj) for i=1,...,nx and
j=1,...,ny
c
c ... phi
c
c      an array dimensioned nx by ny. on input phi
must contain
c      specified boundary values. for example, if
nyd=iparm(5)=1
c      then phi(i,ny) must be set equal to p(xi,yd)
for i=1,...nx
c      prior to calling mud2sa. these values are
preserved by mud2sa.
c      if an initial guess is provided
(iguess=iparm(11)=1) it must
c      be input thru phi.
c
c
c ***      if no initial guess is given (iguess=0) then
phi must still
c      be initialized at all grid points (this is
not checked). these
c      values will serve as an initial guess to the
pde at the coarsest
c      grid level after a transfer from the fine
solution grid. set phi
c      equal to 0.0 at all internal and non-
specified boundaries

```

```
c           grid points if nothing better is available.  
c  
c  
c ... mgopt  
c  
c           an integer vector of length 4 which allows  
the user to select  
c           among various multigrid options. if  
mgopt(1)=0 is input then  
c           a default set of multigrid parameters  
(chosen for robustness)  
c           will be internally selected and the  
remaining values in mgopt  
c           will be ignored. if mgopt(1) is nonzero  
then the parameters  
c           in mgopt are set internally and defined as  
follows: (see the  
c           basic coarse grid correction algorithm  
below)  
c  
c  
c   kcycle = mgopt(1)  
c  
c           = 0 if default multigrid options are to be  
used  
c  
c           = 1 if v cycling is to be used (the least  
expensive per cycle)  
c  
c           = 2 if w cycling is to be used (the  
default)  
c  
c           > 2 if more general k cycling is to be used  
c           *** warning--values larger than 2 increase  
c           the execution time per cycle  
considerably and  
c           result in the nonfatal error ierror =  
-5  
c           which indicates inefficient multigrid  
cycling.  
c  
c   iprer = mgopt(2)  
c  
c           the number of "pre-relaxation" sweeps
```

```
executed before the
c           residual is restricted and cycling is
invoked at the next
c           coarser grid level (default value is 2
whenever mgopt(1)=0)
c
c   ipost = mgopt(3)
c
c           the number of "post relaxation" sweeps
executed after cycling
c           has been invoked at the next coarser grid
level and the residual
c           correction has been transferred back
(default value is 1
c           whenever mgopt(1)=0).
c
c *** if iprер, ipost, or (especially) kcycle is greater
than 2
c       than inefficient multigrid cycling has probably
been chosen and
c       the nonfatal error (see below) ierror = -5 will be
set. note
c       this warning may be overridden by any other
nonzero value
c       for ierror.
c
c   intpol = mgopt(4)
c
c           = 1 if multilinear prolongation
(interpolation) is used to
c           transfer residual corrections and the pde
approximation
c           from coarse to fine grids within full
multigrid cycling.
c
c           = 3 if multicubic prolongation
(interpolation) is used to
c           transfer residual corrections and the pde
approximation
c           from coarse to fine grids within full
multigrid cycling.
c           (this is the default value whenever
mgopt(1)=0).
c
```

```

c *** the default values (2,2,1,3) in the vector mgopt
were chosen for
c      robustness. in some cases v(2,1) cycles with
linear prolongation will
c      give good results with less computation
(especially in two-dimensions).
c      this was the default and only choice in an
earlier version of mudpack
c      (see [1]) and can be set with the integer vector
(1,2,1,1) in mgopt.
c
c *** the schedules for one full multigrid cycle
(iguess=0) using v(2,1)
c      cycles and w(2,1) cycles are depicted for a four
level grid below.
c      the number of relaxation sweeps when each grid is
visited are indicated.
c      the "*" stands for prolongation of the full
approximation and the "."
c      stands for transfer of residuals and residual
corrections within the
c      coarse grid correction algorithm (see below). all
version 5.0.1
c      mudpack solvers use only fully weighted residual
restriction
c
c      one fmg with v(2,1) cycles:
c
c
c      -----
c      ----- level 4 -----1-
c      -----2-----1-----2-----1-----1-
c      ----- level 3
c      -----2-----1-----2-----1-----2-----1-----1-
c      ----- level 2
c      -----3-----3-----3-----3-----3-----3-----3-
c      ----- level 1
c

```

```

C
C      one fmg with w(2,1) cycles:
C
C      -----
C      -----2-----
--1--      level 4
C                      * .
.
.
C      -----2-----1---2-----3-----
1----      level 3
C                      * .
.
C      -----2---1---2---3---1-----2---3---1---2---3---1-
-----      level 2
C                      * . . . . . . . . . . . . . .
C      -----6---6-----6---6-----6---6-----6---6-----
-----      level 1
C
C
C      the form of the "recursive" coarse grid correction
cycling used
C      when kcyle.ge.0 is input is described below in
pseudo-algorithmic
C      language. it is implemented non-recursively in
fortran in mudpack.
C
C      algorithm
cgc(k,l(k),u(k),r(k),kcycle,iprer,ipost,iresh,intpol)
C
C *** approximately solve l(k)*u(k) = r(k) using
multigrid iteration
C *** k is the current grid level
C *** l(k) is the discretized pde operator at level k
C *** u(k) is the initial guess at level k
C *** r(k) is the right hand side at level k
C *** i(k,k-1) is the restriction operator from level k
to level k-1
C *** (the form of i(k,k-1) depends on iresh)
C *** i(k-1,k) is the prolongation operator from level
k-1 to level k
C *** (the form of i(k-1,k) depends on intpol)
C
C      begin algorithm cgc
C
C ***      pre-relax at level k
C

```

```

C      . do (i=1,iprer)
C
C      . . relax(l(k),u(k),r(k))
C
C      . end do
C
C      . if (k > 1) then
C
C ***      restrict the residual from level k to level k-
1
C
C      . . r(k-1) = i(k,k-1) (r(k)-l(k)*u(k))
C
C      . . kount = 0
C
C      . . repeat
C
C ***      solve for the residual correction at level k-1
in u(k-1)
C ***      using algorithm cgc "kcycle" times (this is
the recursion)
C
C      . . . kount = kount+1
C
C      . . . invoke cgc(k-1,l(k-1),u(k-1),r(k-
1),kcycle,iprer,ipost,iresw)
C
C
C      . . until (kount.eq.kcycle)
C
C ***      transfer residual correction in u(k-1) to
level k
C ***      with the prolongation operator and add to u(k)
C
C      . . u(k) = u(k) + i(k-1,k) (u(k-1))
C
C      . end if
C
C ***      post relax at level k
C
C      . do (i=1,ipost)
C
C      . . relax(l(k),u(k),r(k))
C

```

```
C      . end do
C
C      . return
C
C      end algorithm cgc
C
C
C
***** *****
C *** output arguments
*****
C
*****
C ... iparm(16) *** set for intl=0 calls only
C
C          on output iparm(16) contains the actual work
space length
C          required. this will usually be less than
that given by the
C          simplified formula for length=iparm(15) (see
as input argument)
C
C
C ... iparm(17) *** set for intl=1 calls only
C
C          on output iparm(17) contains the actual
number of multigrid cycles
C          between the finest and coarsest grid levels
used to obtain the
C          approximation when error control (tolmax >
0.0) is set.
C
C
C ... fparm(6) *** set for intl=1 calls with fparm(5)
> 0. only
C
C          on output fparm(6) contains the final
computed maximum relative
C          difference between the last two iterates at
the finest grid level.
```

```

c           fparm(6) is computed only if there is error
control (tolmax > 0.0)
c           assume phi1(i,j,k) and phi2(i,j,k) are the
last two computed
c           values for phi(i,j,k) at all points of the
finest grid level.
c           if we define
c
c           phdif = max(abs(phi2(i,j)-phi1(i,j)))
over all i,j
c
c           and
c
c           phmax = max(abs(phi2(i,j))) over all i,j
c
c           then
c
c           fparm(6) = phdif/phmax
c
c           is returned whenever phmax > 0.0. in the
degenerate case
c           phmax = 0.0, fparm(6) = phdif is returned.
c
c
c ... work
c
c           on output work contains intermediate values
that must not
c           be destroyed if mud2sa is to be called again
with intl=1
c
c
c ... phi   *** for intl=1 calls only
c
c           on output phi(i,j) contains the approximation
to p(xi,yj)
c           for all mesh points i = 1,...,nx and
j=1,...,ny. the last
c           computed iterate in phi is returned even if
convergence is
c           not obtained
c
c
c ... ierror

```

```

c
c           for intl=iparm(1)=0 initialization calls,
ierror is an
c           error flag that indicates invalid input
arguments when
c           returned positive and nonfatal warnings when
returned
c           negative. argument checking and
discretization
c           is bypassed for intl=1 calls which can only
return
c           ierror = -1 or 0 or 1.

c
c
c           non-fatal warnings * * *
c
c
c           ==5 if kcycle=mgopt(1) is greater than 2. values
larger than 2 results
c           in an algorithm which probably does far more
computation than
c           necessary. kcycle = 1 (v cycles) or kcycle=2
(w cycles) should
c           suffice for most problems. ierror = -5 is
also set if either
c           iprer = mgopt(2) or ipost=mgopt(3) is greater
than 2. the
c           ierror=-5 flag is overridden by any other
fatal or non-fatal
c           error.

c
c           ==4 if xlmbda < 0 for some grid point (xi,yj)
c
c
c           ==3 if the continuous elliptic pde is singular.
this means the
c           boundary conditions are periodic or pure
derivative at all
c           boundaries and xlmbda(x,y) = 0.0 for all x,y.
a solution is still
c           attempted but convergence may not occur due
to ill-conditioning
c           of the linear system coming from the
discretization. the

```

```

c           ierror = -3 flag overrides the ierror=-4,-5
nonfatal flags.

c
c
c     ==2 if the pde is not elliptic (sigx(x,y) or
sigy(x,y) .le. 0.0).
c           in this case a solution is still attempted
although convergence
c           may not occur due to ill-conditioning of the
linear system.
c           the ierror = -2 flag overrides the ierror=-
5,-4,-3 nonfatal
c           flags.

c
c     ==1 if convergence to the tolerance specified in
tolmax=fparm(5)>0.
c           is not obtained in maxcy=iparm(13) multigrid
cycles between the
c           coarsest (ixp+1,jyq+1) and finest (nx,ny)
grid levels.
c           in this case the last computed iterate is
still returned.
c           the ierror = -1 flag overrides all other
nonfatal flags

c
c
c     no errors * * *
c
c     = 0
c
c     fatal argument errors * * *
c
c     = 1 if intl=iparm(1) is not 0 on initial call or
not 0 or 1
c           on subsequent calls
c
c     = 2 if any of the boundary condition flags
nxa,nxb,nyc,nyd
c           in iparm(2),iparm(3),iparm(4),iparm(5) are not
0,1 or 2
c           or if nxa,nxb or nyc,nyd are not pairwise
zero.
c
c     = 3 if mino(ixp,jyq) < 2 (ixp = iparm(6), jyq =

```

```

iparm(7)
c
c      = 4 if min0(iex,jey) < 1 (iex = iparm(8), jey =
iparm(9)) or
c          if max0(iex,jey) > 50
c
c      = 5 if nx.ne.ipx*2** (iex-1)+1 or
ny.ne.jyq*2** (jey-1)+1
c          (nx = iparm(10), ny = iparm(11))
c
c      = 6 if iguess = iparm(12) is not equal to 0 or 1
c
c      = 7 if maxcy = iparm(13) < 1
c
c      = 8 if method = iparm(14) is not 0,1,2, or 3
c
c      = 9 if length = iparm(15) is too small (see
iparm(16) on output
c          for minimum required work space length)
c
c      =10 if xa >= xb or yc >= yd
c
(xa=fparm(1),xb=fparm(2),yc=fparm(3),yd=fparm(4))
c
c      =11 if tolmax = fparm(5) < 0.0
c
c      errors in setting multigrid options * * * (see
also ierror=-5)
c
c      =12 if kcycle = mgopt(1) < 0 or
c          if iprer = mgopt(2) < 1 or
c          if ipost = mgopt(3) < 1 or
c          if interpol = mgopt(4) is not 1 or 3
c
c
*****
*
c
*****
*
c
c      end of mud2sa documentation
c
c

```

```
*****
**  
C  
*****  
**  
C  
C
```

---

## MUD2SP

```
C  
C      file mud2sp.d  
C  
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * * *  
C      *  
*  
C      *          copyright (c) 2008 by UCAR  
*  
C      *  
*  
C      *          University Corporation for Atmospheric  
Research           *  
C      *  
*  
C      *          all rights reserved  
*  
C      *  
*  
C      *          MUDPACK  version 5.0.1  
*  
C      *  
*  
C      *          A Fortran Package of Multigrid  
*  
C      *  
*  
C      *          Subroutines and Example Programs
```

```
*  
C      *  
*  
C      *      for Solving Elliptic Partial Differential  
Equations      *  
C      *  
*  
C      *                      by  
*  
C      *  
*  
C      *                      John Adams  
*  
C      *  
*  
C      *                      of  
*  
C      *  
*  
C      *      the National Center for Atmospheric  
Research      *  
C      *  
*  
C      *      Boulder, Colorado (80307)  
U.S.A.      *  
C      *  
*  
C      *                      which is sponsored by  
*  
C      *  
*  
C      *      the National Science Foundation  
*  
C      *  
*  
C      *      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C  
C ... file mud2sp.d  
C  
C      contains documentation for:  
C      subroutine  
mud2sp(iparm,fparm,work,cofx,cofy,bndyc,rhs,phi,mgopt,ie  
rror)
```

```
c      A sample fortran driver is file "tmud2sp.f".
c
c ... required MUDPACK files
c
c      mudcom.f
c
c ... purpose
c
c      subroutine mud2sp automatically discretizes and
attempts to compute
c      the second-order difference approximation to the
two-dimensional
c      linear separable elliptic partial differential
equation on a
c      rectangle. the approximation is generated on a
uniform grid covering
c      the rectangle (see mesh description below).
boundary conditions
c      may be specified (dirchlet), periodic, or mixed
derivative in any
c      combination. the form of the pde solved is:
c
c
c      cxx(x)*pxx + cx(x)*px + cex(x)*p(x,y) +
c
c      cyy(y)*pyy + cy(y)*py + cey(y)*p(x,y) =
r(x,y)
c
c      pxx,pyy,px,py are second and first partial
derivatives of the
c      unknown real solution function p(x,y) with respect
to the
c      independent variables x,y. cxx,cx,cex,cyy,cy,cey
are the known
c      real coefficients of the elliptic pde and r(x,y)
is the known
c      real right hand side of the equation. cxx and cyy
should be
c      positive for all x,y in the solution region. If
some of the
c      coefficients depend on both x and y then the PDE
is nonseparable.
c      In this case subroutine muh2 or mud2 must be used
instead of mud2sp
```

```
c      (see the files muh2.d or mud2.d)
c
c
c ... mesh description . . .
c
c      the approximation is generated on a uniform nx by
ny grid.  the grid
c      is superimposed on the rectangular solution region
c
c            [xa,xb] x [yc,yd].
c
c      let
c
c      dlx = (xb-xa) / (nx-1), dly = (yd-yc) / (ny-1)
c
c      be the uniform grid increments in the x,y
directions. then
c
c            xi=xa+(i-1)*dlx,   yj=yc+(j-1)*dly
c
c      for i=1,...,nx and j=1,...,ny  denote the x,y
uniform mesh points
c
c
c ... language
c
c      fortran90/fortran77
c
c
c ... portability
c
c      mudpack5.0.1 software has been compiled and tested
with Fortran77
c      and Fortran90 on a variety of platforms.
c
c ... methods
c
c      details of the methods employed by the solvers in
mudpack are given
c      in [1,9].  [1,2,9] contain performance
measurements on a variety of
c      elliptic pdes (see "references" in the file
"readme").  in summary:
c
```

```
c *** discretization and solution (second-order solvers)
(see [1])
c
c      the pde and boundary conditions are automatically
discretized at all
c      grid levels using second-order finite difference
formula. diagonal
c      dominance at coarser grid levels is maintained in
the presence of
c      nonzero first-order terms by adjusting the second-
order coefficient
c      when necessary. the resulting block tri-diagonal
linear system is
c      approximated using multigrid iteration
[10,11,13,15,16,18]. version
c      5.0.1 of mudpack uses only fully weighted residual
restriction. defaults
c      include cubic prolongation and w(2,1) cycles.
these can be overridden
c      with selected multigrid options (see "mgopt"). error
control based on
c      maximum relative differences is available. full
multigrid cycling (fmg)
c      or cycling beginning or restarting at the finest
grid level can be
c      selected. a menu of relaxation methods including
gauss-seidel point,
c      line relaxation(s) (in any combination of
directions) and planar
c      relaxation (for three-dimensional anisotropic
problems) are provided.
c      all methods use ordering based on alternating
points (red/black),
c      lines, or planes for cray vectorization and
improved convergence
c      rates [14].
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
estimate (i.e.,
c      discretization level error is reached) then this
can be improved to a
```

```
c      fourth-order estimate using the technique of
"deferred corrections."
c      the values in the solution array are used to
generate a fourth-order
c      approximation to the truncation error. second-
order finite difference
c      formula are used to approximate third and fourth
partial derivatives
c      of the solution function [3]. the truncation
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side. the default
multigrid options
c      are used to compute the fourth-order correction
term which is added
c      to the original solution array.
c
c
c ... references (partial)
c
c
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
c      Solution of Linear Elliptic Partial Differential
Equations,"
c      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
c
c      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
c      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
c      1989, pp.1-12.
c      .
c      .
c      .
c      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
Haidvogel, and V. Pizzo,
c      "Applications of Multigrid Software in the
Atmospheric Sciences,"
c      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-
1458.
c      .
```

```
C .
C .
C      [9] J. Adams, "Recent Enhancements in MUDPACK, a
Multigrid Software
C      package for Elliptic Partial Differential
Equations," Applied Math.
C      and Comp., 1991, vol. 43, May 1991, pp. 79-94.
C
C      [10]J. Adams, "MUDPACK-2: Multigrid Software for
Approximating
C      Elliptic Partial Differential Equations on Uniform
Grids with
C      any Resolution," Applied Math. and Comp., 1993,
vol. 53, February
C      1993, pp. 235-249
C .
C .
C .
C
C ... argument description
C
C
C
*****
C *** input arguments
*****
C
*****
C
C
C ... iparm
C
C      an integer vector of length 17 used to pass
integer
C      arguments. iparm is set internally and
defined as
C      follows:
C
C
C ... intl=iparm(1)
C
C      an initialization argument. intl=0 must be
```

```
input
c          on an initial call. in this case input
arguments will
c          be checked for errors and the elliptic
partial differential
c          equation and boundary conditions will be
discretized using
c          second order finite difference formula.
c
c ***      An approximation is NOT generated after an
intl=0 call!
c          mud2sp should be called with intl=1 to
approximate the elliptic
c          PDE discretized by the intl=0 call. intl=1
should also
c          be input if mud2sp has been called earlier
and only the
c          values in in rhs (see below) or gbdy (see
bndyc below)
c          or phi (see below) have changed. This will
bypass
c          redundant pde discretization and argument
checking
c          and save computational time. Some examples
of when
c          intl=1 calls should be used are:
c
c          (0) after a intl=0 argument checking and
discretization call
c
c          (1) mud2sp is being recalled for additional
accuracy. In
c          this case iguess=iparm(12)=1 should also
be used.
c
c          (2) mud2sp is being called every time step in
a time dependent
c          problem (see discussion below) where the
elliptic operator
c          does not depend on time.
c
c          (3) mud2sp is being used to solve the same
elliptic equation
c          for several different right hand sides
```

```
(iguess=0 should
c           probably be used for each new righthand
side) .
c
c           intl = 0 must be input before calling with
intl = 1 when any
c           of the following conditions hold:
c
c           (a) the initial call to mud2sp

c           (b) any of the integer arguments other than
iguess=iparm(12)
c           or maxcy=iparm(13) or mgopt have changed
since the previous
c           call.

c
c           (c) any of the floating point arguments other
than tolmax=
c           fparm(5) have changed since the previous
call
c
c           (d) any of the coefficients input by
cofx,cofy (see below) have
c           changed since the previous call
c
c           (e) any of the constant "alfa" coefficients
input by bndyc
c           (see below) have changed since the
previous call.

c
c           If any of (a) through (e) are true then the
elliptic PDE
c           must be discretized or redisccretized. If
none of (a)
c           through (e) holds, calls can be made with
intl=1.
c           Incorrect calls with intl=1 will produce
erroneous results.
c ***      The values set in the saved work space "work"
(see below) with
c           an intl=0 call must be preserved with
subsequent intl=1 calls.
c
c           MUDPACK software performance should be
```

```
monitored for intl=1
c           calls.  The intl=0 discretization call
performance depends
c           primarily on the efficiency or lack of
efficiency of the
c           user provided subroutines for pde
coefficients and
c           boundary conditions.
c
c ... nxa=iparm(2)
c
c           flags boundary conditions on the edge x=xa
c
c           = 0 if p(x,y) is periodic in x on [xa,xb]
c           (i.e., p(x+xb-xa,y) = p(x,y) for all x,y
c           (if nxa=0 then nxb=0 is required, see
ierror = 2)
c
c           = 1 if p(xa,y) is specified (this must be input
thru phi(1,j))
c
c           = 2 if there are mixed derivative boundary
conditions at x=xa
c           (see bndyc)
c
c
c ... nxn=iparm(3)
c
c           flags boundary conditions on the edge x=xb
c
c           = 0 if p(x,y) is periodic in x on [xa,xb]
c           (i.e., p(x+xb-xa,y) = p(x,y) for all x,y
c           (if nxn=0 then nxa=0 is required, see
ierror = 2)
c
c           = 1 if p(xb,y) is specified (this must be input
thru phi(nx,j))
c
c           = 2 if there are mixed derivative boundary
conditions at x=xb
c           (see bndyc)
c
c
c ... nyc=iparm(4)
```

```

C
C           flags boundary conditions on the edge y=yc
C
C           = 0 if p(x,y) is periodic in y on [yc,yd]
C           (i.e., p(x,y+yd-yc) = p(x,y) for all x,y
C           (if nyc=0 then nyd=0 is required, see
ierror = 2)
C
C           = 1 if p(x,yc) is specified (this must be input
thru phi(i,1))
C
C           = 2 if there are mixed derivative boundary
conditions at y=yc
C           (see bndyc)
C
C
C ... nyd=iparm(5)
C
C           flags boundary conditions on the edge y=yd
C
C           = 0 if p(x,y) is periodic in y on [yc,yd]
C           (i.e., p(x,y+yd-yc) = p(x,y) for all x,y
C           (if nyd=0 then nyc=0 is required, see
ierror = 2)
C
C           = 1 if p(x,yd) is specified (this must be input
thru phi(i,ny))
C
C           = 2 if there are mixed derivative boundary
conditions at y=yd
C           (see bndyc)
C
C
C *** grid size arguments
C
C
C ... ixp = iparm(6)
C
C           an integer greater than one which is used in
defining the number
C           of grid points in the x direction (see nx =
iparm(10)). "ixp+1"
C           is the number of points on the coarsest x
grid visited during

```

```
c           multigrid cycling.  ixp should be chosen as
small as possible.
c           recommended values are the small primes 2 or
3.
c           larger values can reduce multigrid
convergence rates considerably,
c           especially if line relaxation in the x
direction is not used.
c           if ixp > 2 then it should be 2 or a small odd
value since a power
c           of 2 factor of ixp can be removed by
increasing iex = iparm(8)
c           without changing nx = iparm(10).
c
c
c ... jyq = iparm(7)
c
c           an integer greater than one which is used in
defining the number
c           of grid points in the y direction (see ny =
iparm(11)). "jyq+1"
c           is the number of points on the coarsest y
grid visited during
c           multigrid cycling. jyq should be chosen as
small as possible.
c           recommended values are the small primes 2 or
3.
c           larger values can reduce multigrid
convergence rates considerably,
c           especially if line relaxation in the y
direction is not used.
c           if jyq > 2 then it should be 2 or a small odd
value since a power
c           of 2 factor of jyq can be removed by
increasing jey = iparm(9)
c           without changing ny = iparm(11).
c
c
c ... iex = iparm(8)
c
c           a positive integer exponent of 2 used in
defining the number
c           of grid points in the x direction (see nx =
iparm(10)).
```

```

c           iex .le. 50 is required.  for efficient
multigrid cycling,
c           iex should be chosen as large as possible and
ixp=iparm(8)
c           as small as possible within grid size
constraints when
c           defining nx.

c
c
c ... jey = iparm(9)
c
c           a positive integer exponent of 2 used in
defining the number
c           of grid points in the y direction (see ny =
iparm(11)).
c           jey .le. 50 is required.  for efficient
multigrid cycling,
c           jey should be chosen as large as possible and
jyq=iparm(7)
c           as small as possible within grid size
constraints when
c           defining ny.

c
c
c
c ... nx = iparm(10)
c
c           the number of equally spaced grid points in
the interval [xa,xb]
c           (including the boundaries).  nx must have the
form
c
c           nx = ixp* (2** (iex-1)) + 1
c
c           where ixp = iparm(6), iex = iparm(8).
c
c
c ... ny = iparm(11)
c
c           the number of equally spaced grid points in
the interval [yc,yd]
c           (including the boundaries).  ny must have the
form:
c

```

```

c           ny = jyq*(2** (jey-1)) + 1
c
c           where jyq = iparm(7), jey = iparm(9).
c
c
c *** example
c
c           suppose a solution is wanted on a 33 by 97
grid. then
c           ixp=2, jyq=6 and iex=jey=5 could be used. a
better
c           choice would be ixp=2, jyq=3, and iex=5,
jey=6.
c
c
c *** note
c
c           let G be the nx by ny fine grid on which the
approximation is
c           generated and let n = max0(iex,jey). in mudpack,
multigrid
c           cycling is implemented on the ascending chain of
grids
c
c           G(1) < ... < G(k) < ... < G(n) = G.
c
c           each G(k) (k=1,...,n) has mx(k) by my(k) grid
points
c           given by:
c
c           mx(k) = ixp*[2** (max0(iex+k-n,1)-1)] + 1
c
c           my(k) = jyq*[2** (max0(jey+k-n,1)-1)] + 1
c
c
c
c ... iguess=iparm(12)
c
c           = 0 if no initial guess to the pde is
provided
c
c           = 1 if an initial guess to the pde is at the
finest grid
c           level is provided in phi (see below)

```

```

c
c      comments on iguess = 0 or 1 . . .
c
c      even if iguess = 0, phi must be initialized at all
grid points (this
c      is not checked).  phi can be set to 0.0 at non-
dirchlet grid points
c      if nothing better is available.  the values set in
phi when iguess = 0
c      are passed down and serve as an initial guess to
the pde at the coarsest
c      grid level where cycling commences.  in this
sense, values input in
c      phi always serve as an initial guess.  setting
iguess = 0 forces full
c      multigrid cycling beginning at the coarsest and
finishing at the finest
c      grid level.
c
c      if iguess = 1 then the values input in phi are an
initial guess to the
c      pde at the finest grid level where cycling begins.
this option should
c      be used only if a "very good" initial guess is
available (as, for
c      example, when restarting from a previous iguess=0
call).
c
c      time dependent problems . . .
c
c *** assume we are solving an elliptic pde every time
step in a
c      marching problem of the form:
c
c          l(p(t)) = r(t)
c
c      where the differential operator "l" has no time
dependence,
c      "p(t)" is the solution and "r(t)" is the right
hand side at
c      current time "t". let "dt" be the increment
between time steps.
c      then p(t) can be used as an initial guess to
p(t+dt) with

```

```

c      intl = 1 when solving
c
c      l(p(t+dt)) = r(t+dt).
c
c      after the first two time steps, rather than
c continue, it would
c      be better to define the "correction" term:
c
c      e(t,dt) = p(t+dt) - p(t)
c
c      this clearly satisfies the equation
c
c      l(e(t,dt)) = r(t+dt) - r(t).
c
c      this should be solved with iguess = 0 and intl =
1. boundary
c      conditions for e(t,dt) are obtained from the
boundary conditions
c      for p(t) by subtracting given values at t from
given values at
c      t+dt. for example if
c
c      d(p(t))/dx = f(t), d(p(t+dt))/dx = f(t+dt)
c
c      at some x boundary then e(t,dt) satisfies the
derivative
c      boundary condition
c
c      d(e(t,dt))/dx = f(t+dt) - f(t).
c
c      e(t,dt) can be preset to 0.0 (at nondirichlet
points) or (if p(t-dt)
c      is saved) to p(t)-p(t-dt). with iguess = 0, these
values will serve
c      as an initial guess to e(t,dt) at the coarsest
grid level. this
c      approach has the advantage that a full sequence of
multigrid cycles,
c      beginning at the coarsest grid level, is invoked
every time step in
c      solving for e(t,dt). a few digits of accuracy in
e(t,dt), which is
c      ordinarily much smaller than p(t), will yield
several more digits of

```

```

c      accuracy in the final approximation:
c
c      p(t+dt) = p(t) + e(t,dt).
c
c      using this approach to integrate in time will give
more accuracy
c      then using p(t) as an initial guess to p(t+dt) for
all time steps.
c      it does require additional storage.
c
c      if the differential operator "l" has time
dependence (either thru
c      the coefficients in the pde or the coefficients in
the derivative
c      boundary conditions) then use p(t) as an initial
guess to p(t+dt)
c      when solving
c
c      l(t+dt) (p(t+dt)) = r(t+dt)
c
c      with intl = 0 for all time steps (the
discretization must be repeated
c      for each new "t"). either iguess = 0 (p(t) will
then be an initial
c      guess at the coarsest grid level where cycles will
commence) or
c      iguess = 1 (p(t) will then be an initial guess at
the finest grid
c      level where cycles will remain fixed) can be
tried.
c
c
c ... maxcy = iparm(13)
c
c      the exact number of cycles executed between
the finest (nx by
c      ny) and the coarsest ((ixp+1) by (jyq+1))
grid levels when
c      tolmax=fparm(5)=0.0 (no error control). when
tolmax > 0.0
c      is input (error control) then maxcy is a
limit on the number
c      of cycles between the finest and coarsest
grid levels. in

```

```

c      any case, at most maxcy*(iprter+ipost)
relaxation sweeps are
c      are performed at the finest grid level (see
iprter=mgopt(2),
c      ipost=mgopt(3) below). when multigrid
iteration is working
c      "correctly" only a few are required for
convergence. large
c      values for maxcy should not be necessary.
c
c
c ... method = iparm(14) determines the method of
relaxation
c      (gauss-seidel based on alternating points
or lines)
c
c      = 0 for point relaxation
c
c      = 1 for line relaxation in the x direction
c
c      = 2 for line relaxation in the y direction
c
c      = 3 for line relaxation in both the x and y
direction
c
c
c *** choice of method. . .
c
c      let fx represent the quantity cxx(x,y)/dlx**2 over
the solution region.
c
c      let fy represent the quantity cyy(x,y)/dly**2 over
the solution region
c
c      if fx,fy are roughly the same size and do not vary
too much over
c      the solution region choose method = 0. if this
fails try method=3.
c
c      if fx is much greater than fy choose method = 1.
c
c      if fy is much greater than fx choose method = 2
c
c      if neither fx or fy dominates over the solution

```

```
region and they
c      both vary considerably choose method = 3.
c
c
c ... length = iparm(15)
c
c      the length of the work space provided in
vector work (see below).
c      let isx = 0 if method = 0 or method = 2
c      let isx = 3 if method = 1 or method = 3 and
nxa.ne.0
c      let isx = 5 if method = 1 or method = 3 and
nxa.eq.0
c      let jsy = 0 if method = 0 or method = 1
c      let jsy = 3 if method = 2 or method = 3 and
nyc.ne.0
c      let jsy = 5 if method = 2 or method = 3 and
nyc.eq.0
c      then . . .
c
c      length = nx*ny* (5+3*(isx+jsy)/2) +
10* (nx+ny)
c
c      will suffice in all cases but very small nx
and ny.
c      the exact minimal work space length required
for the
c      current set of input arugments is output in
iparm(16).
c      (even if iparm(15) is too small). this will
be usually
c      be less then the value given by the
simplified formula
c      above. * Notice that mud2sp requires
considerably less
c      work space than the nonseparable solvers
muh2,mud2 if
c      and only if method=0 is chosen.
c
c ... fparm
c
c      a floating point vector of length 6 used to
efficiently
c      pass floating point arguments. fparm is set
```

```
internally
c           in mud2sp and defined as follows . . .
c
c
c ... xa=fparm(1), xb=fparm(2)
c
c           the range of the x independent variable. xa
must
c           be less than xb
c
c
c ... yc=fparm(3), yd=fparm(4)
c
c           the range of the y independent variable. yc
must
c           be less than yd.
c
c
c ... tolmax = fparm(5)
c
c           when input positive, tolmax is a maximum
relative error tolerance
c           used to terminate the relaxation iterations.
assume phi1(i,j)
c           and phi2(i,j) are the last two computed
approximations at all
c           grid points of the finest grid level. if we
define
c
c           phdif = max(abs(phi2(i,j)-phi1(i,j))) for
all i,j
c
c           and
c
c           phmax = max(abs(phi2(i,j))) for all i,j
c
c           then "convergence" is considered to have
occurred if and only if
c
c           phdif/phmax < tolmax.
c
c
c           if tolmax=fparm(5)=0.0 is input then there is
no error control
```

```
c           and maxcy cycles from the finest grid level
are executed. maxcy
c           is a limit which cannot be exceeded even with
error control.
c   *** calls with tolmax=0.0, when appropriate
because of known
c           convergence behavior, are more efficient than
calls with tolmax
c           positive (i.e., if possible DO NOT use error
control!).
c
c ... work
c
c           a one dimensional real saved work space (see
iparm(15) for
c           length) which must be preserved from the
previous call when
c           calling with intl=iparm(1)=1.
c
c ... bndyc
c
c           a subroutine with arguments
(kbdy,xory,alfa,gbdy) which
c           are used to input mixed boundary conditions
to mud2sp. bndyc
c           must be declared "external" in the program
calling mud2sp.
c           the boundaries are numbered one thru four and
the mixed
c           derivative boundary conditions are described
below (see the
c           sample driver code "tmud2sp.f" for an example
of how bndyc is
c           can beset up).
c
c           * * * * * * * * * * * * * * * * y=yd
c           *         kbdy=4          *
c           *
c           *
c           *
c           *         kbdy=1          kbdy=2  *
c           *
c           *
c           *
```

```

C           *      kbdy=3      *
C           * * * * * * * * * * *   y=yc
C
C           x=xa           x=xb
C
C
C           (1) the kbdy=1 boundary
C
C           this is the edge x=xa where nxa=iparm(2)=2
flags
C           a mixed boundary condition of the form
C
C           dp/dx + alfxa*p(xa,y) = gbdxa(y)
C
C           in this case kbdy=1,xory=y will be input to
bndyc and
C           alfa,gbdy corresponding to alfxa,gbdxa(y)
must be returned
C
C
C           (2) the kbdy=2 boundary
C
C           this is the edge x=xb where nxb=iparm(3)=2
flags
C           a mixed boundary condition of the form
C
C           dp/dx + alfxb*p(xb,y) = gbdxb(y)
C
C           in this case kbdy=2,xory=y, will be input to
bndyc and
C           alfa,gbdy corresponding to alfxb,gbdxb(y)
must be returned.
C
C
C           (3) the kbdy=3 boundary
C
C           this is the edge y=yc where nyc=iparm(4)=2
flags
C           a mixed boundary condition of the form
C
C           dp/dy + alfyc*p(x,yc) = gbdyc(x)
C
C           in this case kbdy=3,xory=x will be input to
bndyc and

```

```

c           alfa,gbdy corresponding to alfyC,gbdyC(x)
must be returned.

c
c
c           (4) the kbdy=4 boundary
c
c           this is the edge y=yd where nyd=iparm(5)=2
flags
c           a mixed boundary condition of the form
c
c           dp/dy + alfyd*p(x,yd) = gbdyD(x)
c
c           in this case kbdy=4,xory=x will be input to
bndyc and
c           alfa,gbdy corresponding to alfyd,gbdyd(x)
must be returned.

c
c
c ***      alfxA,alfxB,alfyC,alfyD must be constants for
mud2sp.
c           Use muh2 or mud2 if any of these depend on x
or y.
c           bndyc must provide the mixed boundary
condition values
c           in correspondence with those flagged in
iparm(2) thru
c           iparm(5). if all boundaries are specified or
periodic
c           mud2sp will never call bndyc. even then it
must be entered
c           as a dummy subroutine. bndyc must be declared
"external"
c           in the routine calling mud2sp the actual name
chosen may
c           be different.

c
c
c ... cofx
c
c           a subroutine with arguments (x,cxx,cx,cex)
which provides
c           the known real x dependent coefficients for
the separable
c           elliptic pde at any x grid point. the name

```

```
chosen in the calling
c           routine may be different where the coefficient
routine must be declared
c           "external."
c
c ... cofy
c
c           a subroutine with arguments (y,cyy,cy,cey)
which provides
c           the known real y dependent coefficients for
the separable
c           elliptic pde at any y grid point.  the name
chosen in the calling
c           routine may be different where the coefficient
routine must be declared
c           "external."
c
c ... rhs
c
c           an array dimensioned nx by ny which contains
the given
c           right hand side values on the uniform 2-d
mesh.
c
c           rhs(i,j) = r(xi,yj) for i=1,...,nx and
j=1,...,ny
c
c ... phi
c
c           an array dimensioned nx by ny.  on input phi
must contain
c           specified boundary values.  for example, if
nyd=iparm(5)=1
c           then phi(i,ny) must be set equal to p(xi,yd)
for i=1,...nx
c           prior to calling mud2sp.  these values are
preserved by mud2sp.
c           if an initial guess is provided
(iguess=iparm(11)=1) it must
c           be input thru phi.
c
c
c ***      if no initial guess is given (iguess=0) then
phi must still
```

```
c      be initialized at all grid points (this is
c      not checked). these
c      values will serve as an initial guess to the
pde at the coarsest
c      grid level after a transfer from the fine
solution grid. set phi
c      equal to to 0.0 at all internal and non-
specified boundaries
c      grid points if nothing better is available.
c
c
c ... mgopt
c
c      an integer vector of length 4 which allows
the user to select
c      among various multigrid options. if
mgopt(1)=0 is input then
c      a default set of multigrid arguments (chosen
for robustness)
c      will be internally selected and the
remaining values in mgopt
c      will be ignored. if mgopt(1) is nonzero
then the arguments
c      in mgopt are set internally and defined as
follows: (see the
c      basic coarse grid correction algorithm
below)
c
c
c      kcycle = mgopt(1)
c
c      = 0 if default multigrid options are to be
used
c
c      = 1 if v cycling is to be used (the least
expensive per cycle)
c
c      = 2 if w cycling is to be used (the
default)
c
c      > 2 if more general k cycling is to be used
c      *** warning--values larger than 2 increase
c      the execution time per cycle
considerably and
```

```
c           result in the nonfatal error ierror =
-5
c           which indicates inefficient multigrid
c cycling.
c
c     iprер = mgopt(2)
c
c           the number of "pre-relaxation" sweeps
c executed before the
c           residual is restricted and cycling is
c invoked at the next
c           coarser grid level (default value is 2
c whenever mgopt(1)=0)
c
c     ipost = mgopt(3)
c
c           the number of "post relaxation" sweeps
c executed after cycling
c           has been invoked at the next coarser grid
c level and the residual
c           correction has been transferred back
c (default value is 1
c           whenever mgopt(1)=0).
c
c *** if iprер, ipost, or (especially) kcyle is greater
c than 2
c           than inefficient multigrid cycling has probably
c been chosen and
c           the nonfatal error (see below) ierror = -5 will be
c set. note
c           this warning may be overridden by any other
c nonzero value
c           for ierror.
c
c     interpol = mgopt(4)
c
c           = 1 if multilinear prolongation
c (interpolation) is used to
c           transfer residual corrections and the pde
c approximation
c           from coarse to fine grids within full
c multigrid cycling.
c
c           = 3 if multicubic prolongation
```

```

(interpolation) is used to
c           transfer residual corrections and the pde
approximation
c           from coarse to fine grids within full
multigrid cycling.
c           (this is the default value whenever
mgopt(1)=0).
c
c *** the default values (2,2,1,3) in the vector mgopt
were chosen for
c   robustness. in some cases v(2,1) cycles with
linear prolongation will
c   give good results with less computation
(especially in two-dimensions).
c   this was the default and only choice in an
earlier version of mudpack
c   (see [1]) and can be set with the integer vector
(1,2,1,1) in mgopt.
c
c *** the schedules for one full multigrid cycle
(iguess=0) using v(2,1)
c   cycles and w(2,1) cycles are depicted for a four
level grid below.
c   the number of relaxation sweeps when each grid is
visited are indicated.
c   the "*" stands for prolongation of the full
approximation and the "."
c   stands for transfer of residuals and residual
corrections within the
c   coarse grid correction algorithm (see below). all
version 5.0.1
c   mudpack solvers use only fully weighted residual
restriction
c
c   one fmg with v(2,1) cycles:
c
c
c   -----
c   -----2-----1-----
c   -----      level 4
c               * . .
c               * . .
c   -----2-----1-----2-----1-----
c   -----      level 3
c               * . . . .

```

```

C          *
C          -----2-----1-----2-----1-----2-----1-----
C          level 2
C          *
C          * . .
C          * . .
C          ---3---3-----3-----3-----3-----
C          level 1
C
C
C      one fmg with w(2,1) cycles:
C
C          -----
C          -----2-----
--1--      level 4
C          *
C          .
C          -----2-----1-----2-----3-----
1----      level 3
C          *
C          -----2---1---2---3---1-----2---3---1---2---3---1-
C          level 2
C          *
C          -----6---6-----6---6-----6---6-----6---6-----
C          level 1
C
C
C      the form of the "recursive" coarse grid correction
cycling used
c      when kcycle.ge.0 is input is described below in
pseudo-algorithmic
c      language. it is implemented non-recursively in
fortran in mudpack.
c
c      algorithm
cgc(k,l(k),u(k),r(k),kcycle,iprer,ipost,iressw,intpol)
c
c *** approximately solve l(k)*u(k) = r(k) using
multigrid iteration
c *** k is the current grid level
c *** l(k) is the discretized pde operator at level k
c *** u(k) is the initial guess at level k
c *** r(k) is the right hand side at level k
c *** i(k,k-1) is the restriction operator from level k
to level k-1
c *** (the form of i(k,k-1) depends on iressw)

```

```

c *** i(k-1,k) is the prolongation operator from level
k-1 to level k
c *** (the form of i(k-1,k) depends on interpol)
c
c      begin algorithm cgc
c
c ***      pre-relax at level k
c
c      . do (i=1,iprer)
c
c      . . relax(l(k),u(k),r(k))
c
c      . end do
c
c      . if (k > 1) then
c
c ***      restrict the residual from level k to level k-
1
c
c      . . r(k-1) = i(k,k-1) (r(k)-l(k)*u(k))
c
c      . . kount = 0
c
c      . . repeat
c
c ***      solve for the residual correction at level k-1
in u(k-1)
c ***      using algorithm cgc "kcycle" times (this is
the recursion)
c
c      . . . kount = kount+1
c
c      . . . invoke cgc(k-1,l(k-1),u(k-1),r(k-
1),kcycle,iprer,ipost,iresh)
c
c
c      . . until (kount.eq.kcycle)
c
c ***      transfer residual correction in u(k-1) to
level k
c ***      with the prolongation operator and add to u(k)
c
c      . . u(k) = u(k) + i(k-1,k) (u(k-1))
c

```

```

C      . end if
C
C ***    post relax at level k
C
C      . do (i=1,ipost)
C
C      . . relax(l(k),u(k),r(k))
C
C      . end do
C
C      . return
C
C      end algorithm cgc
C
C
C
C ****
C **** output arguments
C ****
C
C ****
C
C ... iparm(16) *** set for intl=0 calls only
C
C          on output iparm(16) contains the actual work
C space length
C          required. this will usually be less than
C that given by the
C          simplified formula for length=iparm(15) (see
C as input argument)
C
C
C ... iparm(17) *** set for intl=1 calls only
C
C          on output iparm(17) contains the actual
C number of multigrid cycles
C          between the finest and coarsest grid levels
C used to obtain the
C          approximation when error control (tolmax >
C 0.0) is set.
C

```

```
c
c ... fparm(6) *** set for intl=1 calls with fparm(5)
> 0. only
c
c           on output fparm(6) contains the final
computed maximum relative
c           difference between the last two iterates at
the finest grid level.
c           fparm(6) is computed only if there is error
control (tolmax > 0.0)
c           assume phi1(i,j,k) and phi2(i,j,k) are the
last two computed
c           values for phi(i,j,k) at all points of the
finest grid level.
c           if we define
c
c           phdif = max(abs(phi2(i,j)-phi1(i,j)))
over all i,j
c
c           and
c
c           phmax = max(abs(phi2(i,j))) over all i,j
c
c           then
c
c           fparm(6) = phdif/phmax
c
c           is returned whenever phmax > 0.0. in the
degenerate case
c           phmax = 0.0, fparm(6) = phdif is returned.
c
c
c ... work
c
c           on output work contains intermediate values
that must not
c           be destroyed if mud2sp is to be called again
with intl=1
c
c
c ... phi *** for intl=1 calls only
c
c           on output phi(i,j) contains the approximation
to p(xi,yj)
```

```
c           for all mesh points i = 1,...,nx and
j=1,...,ny.  the last
c           computed iterate in phi is returned even if
convergence is
c           not obtained
c
c
c ... ierror
c
c           For intl=iparm(1)=0 initialization calls,
ierror is an
c           error flag that indicates invalid input
arguments when
c           returned positive and nonfatal warnings when
returned
c           negative. Argument checking and
discretization
c           is bypassed for intl=1 calls which can only
return
c           ierror = -1 or 0 or 1.
c
c
c     non-fatal warnings * * *
c
c
c     ==5 if kcycle=mgopt(1) is greater than 2. values
larger than 2 results
c           in an algorithm which probably does far more
computation than
c           necessary. kcycle = 1 (v cycles) or kcycle=2
(w cycles) should
c           suffice for most problems. ierror = -5 is
also set if either
c           iprер = mgopt(2) or ipost=mgopt(3) is greater
than 2. the
c           ierror=-5 flag is overridden by any other
fatal or non-fatal
c           error.
c
c     ==4 if there are dominant nonzero first order
terms in the pde which
c           make it "hyperbolic" at the finest grid level.
numerically, this
c           happens if:
```

```

c
c           abs(cx)*dlx > 2.*abs(cxx)      (dlx = (xb-
xa)/(nx-1))
c
c           (or)
c
c           abs(cy)*dly > 2.*abs(cyy)      (dly = (yd-
yc)/(ny-1))
c
c
c           at some fine grid point (xi,yj). if an
adjustment is not made the
c           condition can lead to a matrix coming from the
discretization
c           which is not diagonally dominant and
divergence is possible. since
c           the condition is "likely" at coarser grid
levels for pde's with
c           nonzero first order terms, the adjustments
(actually first order
c           approximations to the pde)
c
c
c           cxx = amax1(cxx,0.5*abs(cx)*dx)
c
c           (and)
c
c           cyy = amax1(cyy,0.5*abs(cy)*dy)
c
c
c           (here dx,dy are the x,y mesh sizes of the
subgrid)
c
c           are made to preserve convergence of multigrid
iteration. if made
c           at the finest grid level, it can lead to
convergence to an
c           erroneous solution (flagged by ierror = -4).
a possible remedy
c           is to increase resolution. the ierror = -4
flag overrides the
c           nonfatal ierror = -5 flag.
c
c

```

```

c      ==3  if the continuous elliptic pde is singular.
this means the
c          boundary conditions are periodic or pure
derivative at all
c          boundaries and ce(x,y) = 0.0 for all x,y.  a
solution is still
c          attempted but convergence may not occur due
to ill-conditioning
c          of the linear system coming from the
discretization.  the
c          ierror = -3 flag overrides the ierror=-4 and
ierror=-5 nonfatal
c          flags.

c
c
c      ==2  if the pde is not elliptic (i.e.,
cxx*cyy.le.0.0 for some (xi,yj))
c          in this case a solution is still attempted
although convergence
c          may not occur due to ill-conditioning of the
linear system.
c          the ierror = -2 flag overrides the ierror=-
5,-4,-3 nonfatal
c          flags.

c
c
c      ==1  if convergence to the tolerance specified in
tolmax=fparm(5)>0.
c          is not obtained in maxcy=iparm(13) multigrid
cycles between the
c          coarsest (ixp+1,jyq+1) and finest (nx,ny)
grid levels.
c          in this case the last computed iterate is
still returned.
c          the ierror = -1 flag overrides all other
nonfatal flags

c
c
c      no errors * * *
c
c      = 0
c
c      fatal argument errors * * *
c

```

```

c      = 1 if intl=iparm(1) is not 0 on the first call or
not 0 or 1
c          on subsequent calls
c
c      = 2 if any of the boundary condition flags
nxa,nxb,nyc,nyd
c          in iparm(2),iparm(3),iparm(4),iparm(5) are not
0,1 or 2
c          or if nxa,nxb or nyc,nyd are not pairwise
zero.
c
c      = 3 if min0(ixp,jyq) < 2 (ixp = iparm(6), jyq =
iparm(7))
c
c      = 4 if min0(iex,jey) < 1 (iex = iparm(8), jey =
iparm(9)) or
c          if max0(iex,jey) > 50
c
c      = 5 if nx.ne.ixp*2** (iex-1)+1 or
ny.ne.jyq*2** (jey-1)+1
c          (nx = iparm(10), ny = iparm(11))
c
c      = 6 if iguess = iparm(12) is not equal to 0 or 1
c
c      = 7 if maxcy = iparm(13) < 1
c
c      = 8 if method = iparm(14) is not 0,1,2, or 3
c
c      = 9 if length = iparm(15) is too small (see
iparm(16) on output
c          for minimum required work space length)
c
c      =10 if xa >= xb or yc >= yd
c
c (xa=fparm(1),xb=fparm(2),yc=fparm(3),yd=fparm(4))
c
c      =11 if tolmax = fparm(5) < 0.0
c
c      errors in setting multigrid options * * * (see
also ierror=-5)
c
c      =12 if kcycle = mgopt(1) < 0 or
c          if iprer = mgopt(1) < 1 or
c          if ipost = mgopt(3) < 1 or

```

```
C           if intpol = mgopt(4)  is not 1 or 3
C
C
*****
*
C
*****
*
C
*****
C     end of mud2sp documentation
C
C
*****
**
C
*****
**
C
C
```

---

## MUD3

```
C
C      file mud3.d
C
C      * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *               University Corporation for Atmospheric
Research          *
C      *
*
C      *               all rights reserved
```

\*  
C \*  
\*  
C \* MUDPACK version 5.0.1  
\*  
C \*  
\*  
C \* A Fortran Package of Multigrid  
\*  
C \*  
\*  
C \* Subroutines and Example Programs  
\*  
C \*  
\*  
C \* for Solving Elliptic Partial Differential  
Equations \*  
C \*  
\*  
C \* by  
\*  
C \*  
\*  
C \* John Adams  
\*  
C \*  
\*  
C \* of  
\*  
C \*  
\*  
C \* the National Center for Atmospheric  
Research \*  
C \*  
\*  
C \* Boulder, Colorado (80307)  
U.S.A. \*  
C \*  
\*  
C \* which is sponsored by  
\*  
C \*  
\*  
C \* the National Science Foundation

```

*
C      *
*
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C
C ... file mud3.d
C
C      contains documentation for:
C      subroutine
mud3(iparm,fparm,work,coef,bndyc,rhs,phi,mgopt,ierror)
C      A sample fortran driver is file "tmud3.f".
C
C ... required MUDPACK files
C
C      mudcom.f, mud3ln.f, mud3pn.f
C
C ... purpose
C
C      subroutine mud3 automatically discretizes and
attempts to compute
C      the second order finite difference approximation
to a three-
C      dimensional linear nonseparable elliptic partial
differential
C      equation on a box.  the approximation is generated
on a uniform
C      grid covering the box (see mesh description
below).  boundary
C      conditions may be any combination of mixed,
specified (Dirchlet)
C      or periodic.  the form of the pde solved is . . .
C
C      cxx(x,y,z)*pxx + cyy(x,y,z)*pyy +
czz(z,y,z)*pzz +
C
C      cx(x,y,z)*px + cy(x,y,z)*py + cz(x,y,z)*pz +
C
C      ce(x,y,z)*p(x,y,z) = r(x,y,z)
C
C      here cxx,cyy,czz,cx,cy,cz,ce are the known real
coefficients
C      of the pde; pxx,pyy,pzz,px,py,pz are the second
and first

```

```

c      partial derivatives of the unknown solution
function p(x,y,z)
c      with respect to the independent variables x,y,z;
r(x,y,z) is
c      is the known real right hand side of the elliptic
pde. cxx,cyy
c      and czz should be positive for all (x,y,z) in the
solution region.
c
c
c ... mesh description . . .
c
c      the approximation is generated on a uniform nx by
ny by nz grid.
c      the grid is superimposed on the rectangular
solution region
c
c      [xa,xb] x [yc,yd] x [ze,zf].
c
c      let
c
c      dlx = (xb-xa) / (nx-1), dly = (yd-yc) / (ny-1),
dlz = (zf-ze) / (nz-1)
c
c      be the uniform grid increments in the x,y,z
directions. then
c
c      xi=xa+(i-1)*dlx,   yj=yc+(j-1)*dly,  zk =
ze+(k-1)*dlz
c
c      for i=1,...,nx; j=1,...,ny; k=1,...,nz  denote the
x,y,z uniform
c      mesh points.
c
c
c ... language
c
c      fortran90/fortran77
c
c
c ... portability
c
c      mudpack5.0.1 software has been compiled and tested
with Fortran77

```

```
c      and Fortran90 on a variety of platforms.  
c  
c ... methods  
c  
c      details of the methods employed by the solvers in  
mudpack are given  
c      in [1,9]. [1,2,9] contain performance  
measurements on a variety of  
c      elliptic pdes (see "references" in the file  
"readme"). in summary:  
c  
c *** discretization and solution (second-order solvers)  
(see [1])  
c  
c      the pde and boundary conditions are automatically  
discretized at all  
c      grid levels using second-order finite difference  
formula. diagonal  
c      dominance at coarser grid levels is maintained in  
the presence of  
c      nonzero first-order terms by adjusting the second-  
order coefficient  
c      when necessary. the resulting block tri-diagonal  
linear system is  
c      approximated using multigrid iteration  
[10,11,13,15,16,18]. version  
c      5.0.1 of mudpack uses only fully weighted residual  
restriction. defaults  
c      include cubic prolongation and w(2,1) cycles.  
these can be overridden  
c      with selected multigrid options (see "mgopt").  
error control based on  
c      maximum relative differences is available. full  
multigrid cycling (fmg)  
c      or cycling beginning or restarting at the finest  
grid level can be  
c      selected. a menu of relaxation methods including  
gauss-seidel point,  
c      line relaxation(s) (in any combination of  
directions) and planar  
c      relaxation (for three-dimensional anisotropic  
problems) are provided.  
c      all methods use ordering based on alternating  
points (red/black),
```

```
c      lines, or planes for cray vectorization and
improved convergence
c      rates [14].
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
estimate (i.e.,
c      discretization level error is reached) then this
can be improved to a
c      fourth-order estimate using the technique of
"deferred corrections."
c      the values in the solution array are used to
generate a fourth-order
c      approximation to the truncation error. second-
order finite difference
c      formula are used to approximate third and fourth
partial derivatives
c      of the solution function [3]. the truncation
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side. the default
multigrid options
c      are used to compute the fourth-order correction
term which is added
c      to the original solution array.
c
c
c ... references (partial)
c
c
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
c      Solution of Linear Elliptic Partial Differential
Equations,"
c      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
c
c      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
c      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
```

```
c      1989, pp.1-12.  
c      .  
c      .  
c      .  
c      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.  
Haidvogel, and V. Pizzo,  
c      "Applications of Multigrid Software in the  
Atmospheric Sciences,"  
c      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-  
1458.  
c      .  
c      .  
c      .  
c      [9] J. Adams, "Recent Enhancements in MUDPACK, a  
Multigrid Software  
c      package for Elliptic Partial Differential  
Equations," Applied Math.  
c      and Comp., 1991, vol. 43, May 1991, pp. 79-94.  
c  
c      [10]J. Adams, "MUDPACK-2: Multigrid Software for  
Approximating  
c      Elliptic Partial Differential Equations on Uniform  
Grids with  
c      any Resolution," Applied Math. and Comp., 1993,  
vol. 53, February  
c      1993, pp. 235-249  
c      .  
c      .  
c      .  
c  
c ... argument description  
c  
c  
c  
*****  
*****  
c *** input arguments  
*****  
c  
*****  
*****  
c  
*****  
*****  
c  
c  
c ... iparm
```

```
c
c      an integer vector of length 23 used to
efficiently pass
c      integer arguments. iparm is set internally
in mud3
c      and defined as follows . . .
c
c
c ... intl=iparm(1)
c
c      an initialization argument. intl=0 must be
input
c      on an initial call. in this case input
arguments will
c      be checked for errors and the elliptic
partial differential
c      equation and boundary conditions will be
discretized using
c      second order finite difference formula.
c
c ***      An approximation is NOT generated after an
intl=0 call!
c      mud3 should be called with intl=1 to
approximate the elliptic
c      PDE discretized by the intl=0 call. intl=1
should also
c      be input if mud3 has been called earlier and
only the
c      values in in rhs (see below) or gbdy (see
bndyc below)
c      or phi (see below) have changed. This will
bypass
c      redundant pde discretization and argument
checking
c      and save computational time. Some examples
of when
c      intl=1 calls should be used are:
c
c      (0) after a intl=0 argument checking and
discretization call
c
c      (1) mud3 is being recalled for additional
accuracy. In
c      this case iguess=iparm(12)=1 should also
```

be used.

c

c (2) mud3 is being called every time step in a  
time dependent

c problem (see discussion below) where the  
elliptic operator

c does not depend on time.

c

c (3) mud3 is being used to solve the same  
elliptic equation

c for several different right hand sides  
(iguess=0 should

c probably be used for each new righthand  
side).

c

c intl = 0 must be input before calling with  
intl = 1 when any

c of the following conditions hold:

c

c (a) the initial call to mud3

c (b) any of the integer arguments other than  
iguess=iparm(12)

c or maxcy=iparm(13) or mgopt have changed  
since the previous

c call.

c

c (c) any of the floating point arguments other  
than tolmax=

c fparm(5) have changed since the previous  
call

c

c (d) any of the coefficients input by coef  
(see below) have

c changed since the previous call

c

c (e) any of the "alfa" coefficients input by  
bndyc (see below)

c have changed since the previous call.

c

c If any of (a) through (e) are true then the  
elliptic PDE

c must be discretized or rediscretized. If  
none of (a)

```
c           through (e) holds, calls can be made with
intl=1.
c           Incorrect calls with intl=1 will produce
erroneous results.
c ***      The values set in the saved work space "work"
(see below) with
c           an intl=0 call must be preserved with
subsequent intl=1 calls.
c
c           MUDPACK software performance should be
monitored for intl=1
c           calls.  The intl=0 discretization call
performance depends
c           primarily on the efficiency or lack of
efficiency of the
c           user provided subroutines for pde
coefficients and
c           boundary conditions.
c
c
c ... nxa=iparm(2)
c
c           flags boundary conditions on the (y,z) plane
x=xa
c
c           = 0 if p(x,y,z) is periodic in x on [xa,xb]
c           (i.e., p(x+xb-xa,y,z) = p(x,y,z) for all
x,y,z)
c
c           = 1 if p(xa,y,z) is specified (this must be
input thru phi(1,j,k))
c
c           = 2 if there are mixed derivative boundary
conditions at x=xa
c           (see "bndyc" description below where kbdy =
1)
c
c
c ... nxn=iparm(3)
c
c           flags boundary conditions on the (y,z) plane
x=xb
c
c           = 0 if p(x,y,z) is periodic in x on [xa,xb]
```

```

c           (i.e., p(x+xb-xa,y,z) = p(x,y,z) for all
x,y,z)
c
c           = 1 if p(xb,y,z) is specified (this must be
input thru phi(nx,j,k))
c
c           = 2 if there are mixed derivative boundary
conditions at x=xb
c           (see "bndyc" description below where kbdy =
2)
c
c
c ... nyc=iparm(4)
c
c           flags boundary conditions on the (x,z) plane
y=yc
c
c           = 0 if p(x,y,z) is periodic in y on [yc,yd]
c           (i.e., p(x,y+yd-yc,z) = p(x,y,z) for all
x,y,z)

c           = 1 if p(x,yc,z) is specified (this must be
input thru phi(i,1,k))

c           = 2 if there are mixed derivative boundary
conditions at y=yc
c           (see "bndyc" description below where kbdy =
3)
c
c
c ... nyd=iparm(5)
c
c           flags boundary conditions on the (x,z) plane
y=yd
c
c           = 0 if p(x,y,z) is periodic in y on [yc,yd]
c           (i.e., p(x,y+yd-yc,z) = p(x,y,z) for all
x,y,z)

c           = 1 if p(x,yd,z) is specified (this must be
input thru phi(i,ny,k))

c           = 2 if there are mixed derivative boundary
conditions at y=yd

```

```

C           (see "bndyc" description below where kbdy =
4)
C
C
C ... nze=iparm(6)
C
C           flags boundary conditions on the (x,y) plane
z=ze
C
C           = 0 if p(x,y,z) is periodic in z on [ze,zf]
C           (i.e., p(x,y,z+zf-ze) = p(x,y,z) for all
x,y,z

C           = 1 if p(x,y,ze) is specified (this must be
input thru phi(i,j,1))

C           = 2 if there are mixed derivative boundary
conditions at z=ze
C           (see "bndyc" description below where kbdy =
5)
C
C
C ... nzf=iparm(7)
C
C           flags boundary conditions on the (x,y) plane
z=zf
C
C           = 0 if p(x,y,z) is periodic in z on [ze,zf]
C           (i.e., p(x,y,z+zf-ze) = p(x,y,z) for all
x,y,z

C           = 1 if p(x,y,zf) is specified (this must be
input thru phi(i,j,nz))

C           = 2 if there are mixed derivative boundary
conditions at z=zf
C           (see "bndyc" description below where kbdy =
6)
C
C
C *** grid size arguments
C
C
C ... ixp = iparm(8)

```

```
c
c      an integer greater than one which is used in
defining the number
c      of grid points in the x direction (see nx =
iparm(14)). "ixp+1"
c      is the number of points on the coarsest x
grid visited during
c      multigrid cycling. ixp should be chosen as
small as possible.
c      recommended values are the small primes 2 or
3 or (possibly) 5.
c      larger values can reduce multigrid
convergence rates considerably,
c      especially if line relaxation in the x
direction is not used.
c      if ixp > 2 then it should be 2 or a small odd
value since a power
c      of 2 factor of ixp can be removed by
increasing iex = iparm(11)
c      without changing nx = iparm(14)
c
c
c ... jyq = iparm(9)
c
c      an integer greater than one which is used in
defining the number
c      of grid points in the y direction (see ny =
iparm(15)). "jyq+1"
c      is the number of points on the coarsest y
grid visited during
c      multigrid cycling. jyq should be chosen as
small as possible.
c      recommended values are the small primes 2 or
3 or (possibly) 5.
c      larger values can reduce multigrid
convergence rates considerably,
c      especially if line relaxation in the y
direction is not used.
c      if jyq > 2 then it should be 2 or a small odd
value since a power
c      of 2 factor of jyq can be removed by
increasing jey = iparm(12)
c      without changing ny = iparm(15)
c
```

```

c
c ... kzs = iparm(10)
c
c           an integer greater than one which is used in
defining the number
c           of grid points in the z direction (see nz =
iparm(16)). "kzs+1"
c           is the number of points on the coarsest z
grid visited during
c           multigrid cycling. kzs should be chosen as
small as possible.
c           recommended values are the small primes 2 or
3 or (possibly) 5.
c           larger values can reduce multigrid
convergence rates considerably,
c           especially if line relaxation in the z
direction is not used.
c           if kzs > 2 then it should be 2 or a small odd
value since a power
c           of 2 factor of kzs can be removed by
increasing kez = iparm(13)
c           without changing nz = iparm(16)
c
c
c ... iex = iparm(11)
c
c           a positive integer exponent of 2 used in
defining the number
c           of grid points in the x direction (see nx =
iparm(14)).
c           iex .le. 50 is required. for efficient
multigrid cycling,
c           iex should be chosen as large as possible and
ixp=iparm(8)
c           as small as possible within grid size
constraints when
c           defining nx = iparm(14).
c
c
c ... jey = iparm(12)
c
c           a positive integer exponent of 2 used in
defining the number
c           of grid points in the y direction (see ny =

```

```

iparm(15)) .
c           jey .le. 50 is required.  for efficient
multigrid cycling,
c           jey should be chosen as large as possible and
jyq=iparm(9)
c           as small as possible within grid size
constraints when
c           defining ny = iparm(15) .
c
c
c ... kez = iparm(13)
c
c           a positive integer exponent of 2 used in
defining the number
c           of grid points in the z direction (see nz =
iparm(16)) .
c           kez .le. 50 is required.  for efficient
multigrid cycling,
c           kez should be chosen as large as possible and
kzr=iparm(10)
c           as small as possible within grid size
constraints when
c           defining nz = iparm(16) .
c
c
c ... nx = iparm(14)
c
c           the number of equally spaced grid points in
the interval [xa,xb]
c           (including the boundaries).  nx must have the
form
c
c           nx = ixp* (2** (iex-1)) + 1
c
c           where ixp = iparm(8), iex = iparm(11) .
c
c
c ... ny = iparm(15)
c
c           the number of equally spaced grid points in
the interval [yc,yd]
c           (including the boundaries).  ny must have the
form:
c
```

```

c           ny = jyq*(2** (jey-1)) + 1
c
c           where jyq = iparm(9), jey = iparm(12).
c
c
c ... nz = iparm(16)
c
c           the number of equally spaced grid points in
c           the interval [ze,zf]
c           (including the boundaries). nz must have the
c           form
c
c           nz = ksr*(2** (kez-1)) + 1
c
c           where ksr = iparm(10), kez = iparm(13)
c
c
c *** example
c
c           suppose a solution is wanted on a 33 by 65 by
c           97 grid. then
c           ixp=2, jyq=4, ksr=6 and iex=jey=kez=5 could be
c           used. a better
c           choice would be ixp=jyq=2, ksr=3, and iex=5,
c           jey=kez=6.
c
c *** note
c
c           let G be the nx by ny by nz fine grid on which the
c           approximation is
c           generated and let n = max0(iex,jey,kez). in
c           mudpack, multigrid
c           cycling is implemented on the ascending chain of
c           grids
c
c           G(1) < ... < G(k) < ... < G(n) = G.
c
c           each g(k) (k=1,...,n) has mx(k) by my(k) by mz(k)
c           grid points
c           given by:
c
c           mx(k) = ixp*[2** (max0(iex+k-n,1)-1)] + 1
c
c           my(k) = jyq*[2** (max0(jey+k-n,1)-1)] + 1

```

```
C
C      mz(k) = kzs* [2** (max0(kez+k-n,1)-1)] + 1
C
C
C
C ... iguess=iparm(17)
C
C           = 0 if no initial guess to the pde is
provided
C           and/or full multigrid cycling beginning
at the
C           coarsest grid level is desired.
C
C           = 1 if an initial guess to the pde at the
finest grid
C           level is provided in phi (see below). in
this case
C           cycling beginning or restarting at the
finest grid
C           is initiated.
C
C *** comments on iguess = 0 or 1 . . .
C
C
C     setting iguess=0 forces full multigrid or "fmg"
cycling. phi
C     must be initialized at all grid points. it can be
set to zero at
C     non-Dirchlet grid points if nothing better is
available. the
C     values set in phi when iguess = 0 are passed and
down and serve
C     as an initial guess to the pde at the coarsest
grid level where
C     multigrid cycling commences.
C
C     if iguess = 1 then the values input in phi are an
initial guess to the
C     pde at the finest grid level where cycling begins.
this option should
C     be used only if a "very good" initial guess is
available (as, for
C     example, when restarting from a previous iguess=0
call).
```

```

c
c *** time dependent problems . . .
c
c      assume we are solving an elliptic pde every time
step in a
c      marching problem of the form:
c
c          l(p(t)) = r(t)
c
c      where the differential operator "l" has no time
dependence,
c      "p(t)" is the solution and "r(t)" is the right
hand side at
c      current time "t". let "dt" be the increment
between time steps.
c      then p(t) can be used as an initial guess to
p(t+dt) with
c      intl = 1 when solving
c
c          l(p(t+dt)) = r(t+dt).
c
c      after the first two time steps, rather than
continue, it would
c      be better to define the "correction" term:
c
c          e(t,dt) = p(t+dt) - p(t)
c
c      this clearly satisfies the equation
c
c          l(e(t,dt)) = r(t+dt) - r(t).
c
c      this should be solved with iguess = 0 and intl =
1. boundary
c      conditions for e(t,dt) are obtained from the
boundary conditions
c      for p(t) by subtracting given values at t from
given values at
c      t+dt. for example if
c
c          d(p(t))/dx = f(t), d(p(t+dt))/dx = f(t+dt)
c
c      at some x boundary then e(t,dt) satisfies the
derivative
c      boundary condition

```

```

c
c      d(e(t,dt))/dx = f(t+dt) - f(t).
c
c      e(t,dt) can be preset to 0.0 (at non-Dirchlet
points) or (if p(t-dt)
c      is saved) to p(t)-p(t-dt). with iguess = 0, these
values will serve
c      as an initial guess to e(t,dt) at the coarsest
grid level. this
c      approach has the advantage that a full sequence of
multigrid cycles,
c      beginning at the coarsest grid level, is invoked
every time step in
c      solving for e(t,dt). a few digits of accuracy in
e(t,dt), which is
c      ordinarily much smaller than p(t), will yield
several more digits of
c      accuracy in the final approximation:
c
c      p(t+dt) = p(t) + e(t,dt).
c
c      using this approach to integrate in time will give
more accuracy
c      then using p(t) as an initial guess to p(t+dt) for
all time steps.
c      it does require additional storage.
c
c      if the differential operator "l" has time
dependence (either thru
c      the coefficients in the pde or the coefficients in
the derivative
c      boundary conditions) then use p(t) as an initial
guess to p(t+dt)
c      when solving
c
c      l(t+dt)(p(t+dt)) = r(t+dt)
c
c      with intl = 0 for all time steps (the
discretization must be repeated
c      for each new "t"). either iguess = 0 (p(t) will
then be an initial
c      guess at the coarsest grid level where cycles will
commence) or
c      iguess = 1 (p(t) will then be an initial guess at

```

```
the finest grid
c      level where cycles will remain fixed) can be
tried.

c
c
c ... maxcy = iparm(18)
c
c           the exact number of cycles executed between
the finest
c           (nx by ny by nz) and the coarsest ((ixp+1) by
(jyq+1) by
c           (kzr+1)) grid levels when tolmax=fparm(7)=0.0
(no error
c           control). when tolmax=fparm(7).gt.0.0 is
input (error control)
c           then maxcy is a limit on the number of cycles
between the
c           finest and coarsest grid levels. in any
case, at most
c           maxcy*(iprert+ipost) relaxation sweeps are
performed at the
c           finest grid level (see
iprer=mgopt(2),ipost=mgopt(3) below)
c           when multigrid iteration is working
"correctly" only a few
c           cycles are required for convergence. large
values for maxcy
c           should not be required.

c
c
c ... method = iparm(19)
c
c           this sets the method of relaxation (all
relaxation
c           schemes in mudpack use red/black type
ordering)
c
c           = 0 for gauss-seidel pointwise relaxation
c
c           = 1 for line relaxation in the x direction
c
c           = 2 for line relaxation in the y direction
c
c           = 3 for line relaxation in the z direction
```

```
c
c           = 4 for line relaxation in the x and y
direction
c
c           = 5 for line relaxation in the x and z
direction
c
c           = 6 for line relaxation in the y and z
direction
c
c           = 7 for line relaxation in the x,y and z
direction
c
c           = 8 for x,y planar relaxation
c
c           = 9 for x,z planar relaxation
c
c           =10 for y,z planar relaxation
c
c *** if nxa = 0 and nx = 3 at a grid level where line
relaxation in the x
c      direction is flagged then it will be replaced by
gauss-seidel point
c      relaxation at that grid level.
c
c *** if nyc = 0 and ny = 3 at a grid level where line
relaxation in the y
c      direction is flagged then it will be replaced by
gauss-seidel point
c      relaxation at that grid level.
c
c *** if nze = 0 and nz = 3 at a grid level where line
relaxation in the z
c      direction is flagged then it will be replaced by
gauss-seidel point
c      relaxation at that grid level.
c
c      these adjustments are necessary since the
simultaneous tri-diagonal
c      solvers used with line periodic relaxation must
have n > 2 where n
c      is number of unknowns (excluding the periodic
point).
```

```
c *** choice of method
c
c      this is very important for efficient convergence.
in some cases
c      experimentation may be required.
c
c      let fx represent the quantity cxx(x,y,z)/dlx**2
over the solution box
c
c      let fy represent the quantity cyy(x,y,z)/dly**2
over the solution box
c
c      let fz represent the quantity czz(x,y,z)/dlz**2
over the solution box
c
c      (0) if fx,fy,fz are roughly the same size and do
not vary too
c          much choose method = 0.  if this fails try
method = 7.
c
c      (1) if fx is much greater then fy,fz and fy,fz
are roughly the same
c          size choose method = 1
c
c      (2) if fy is much greater then fx,fz and fx,fz
are roughly the same
c          size choose method = 2
c
c      (3) if fz is much greater then fx,fy and fx,fy
are roughly the same
c          size choose method = 3
c
c      (4) if fx,fy are roughly the same and both are
much greater then fz
c          try method = 4.  if this fails try method = 8
c
c      (5) if fx,fz are roughly the same and both are
much greater then fy
c          try method = 5.  if this fails try method = 9
c
c      (6) if fy,fz are roughly the same and both are
much greater then fx
c          try method = 6.  if this fails try method =
10
```

```

c
c      (7) if fx,fy,fz vary considerably with none
dominating try method = 7
c
c      (8) if fx and fy are considerably greater then fz
but not necessarily
c          the same size (e.g., fx=1000.,fy=100.,fz=1.)
try method = 8
c
c      (9) if fx and fz are considerably greater then fy
but not necessarily
c          the same size (e.g., fx=10.,fy=1.,fz=1000.)
try method = 9
c
c      (10)if fy and fz are considerably greater then fx
but not necessarily
c          the same size (e.g., fx=1.,fy=100.,fz=10.)
try method = 10
c
c
c ... meth2 = iparm(20) determines the method of
relaxation used in the planes
c          when method = 8 or 9 or 10.
c
c
c          as above, let fx,fy,fz represent the
quantities cxx/dlx**2,
c          cyy/dly**2,czz/dlz**2 over the box.
c
c          (if method = 8)
c
c          = 0 for gauss-seidel pointwise relaxation
c              in the x,y plane for each fixed z
c          = 1 for line relaxation in the x direction
c              in the x,y plane for each fixed z
c          = 2 for line relaxation in the y direction
c              in the x,y plane for each fixed z
c          = 3 for line relaxation in the x and y
direction
c              in the x,y plane for each fixed z
c
c          (1) if fx,fy are roughly the same and vary
little choose meth2 = 0
c          (2) if fx is much greater then fy choose

```

```

meth2 = 1
c           (3) if fy is much greater than fx choose
meth2 = 2
c           (4) if none of the above or meth2 = 0 fails
choose meth2 = 3
c
c           (if method = 9)
c
c           = 0 for gauss-seidel pointwise relaxation
with red/black ordering
c           in the x,z plane for each fixed y
c           = 1 for simultaneous line relaxation in the x
direction
c           of the x,z plane for each fixed y
c           = 2 for simultaneous line relaxation in the z
direction
c           of the x,z plane for each fixed y
c           = 3 for simultaneous line relaxation in the x
and z direction
c           of the x,z plane for each fixed y
c
c           (1) if fx,fz are roughly the same and vary
little choose meth2 = 0
c           (2) if fx is much greater than fz choose
meth2 = 1
c           (3) if fz is much greater than fx choose
meth2 = 2
c           (4) if none of the above or meth2 = 0 fails
choose meth2 = 3
c
c           (if method = 10)
c
c           = 0 for gauss-seidel pointwise relaxation
with red/black ordering
c           in the y,z plane for each fixed x
c           = 1 for simultaneous line relaxation in the y
direction
c           of the y,z plane for each fixed x
c           = 2 for simultaneous line relaxation in the z
direction
c           of the y,z plane for each fixed x
c           = 3 for simultaneous line relaxation in the y
and z direction
c           of the y,z plane for each fixed x

```

```

c
c           (1) if fy,fz are roughly the same and vary
little choose meth2 = 0
c           (2) if fy is much greater then fz choose
meth2 = 1
c           (3) if fz is much greater then fy choose
meth2 = 2
c           (4) if none of the above or meth2 = 0 fails
choose meth2 = 3
c
c
c ... length = iparm(21)
c
c           the length of the work space provided in
vector work.
c
c           let isx = 3 if method = 1,4,5 or 7 and
nx.a.ne.0
c           let isx = 5 if method = 1,4,5 or 7 and
nx.a.eq.0
c           let isx = 0 if method has any other value
c
c           let jsy = 3 if method = 2,4,6 or 7 and
nyc.ne.0
c           let jsy = 5 if method = 2,4,6 or 7 and
nyc.eq.0
c           let jsy = 0 if method has any other value
c
c           let ksz = 3 if method = 3,5,6 or 7 and
nze.ne.0
c           let ksz = 5 if method = 3,5,6 or 7 and
nze.eq.0
c           let ksz = 0 if method has any other value
c
c
c           then (for method .le.7)
c
c           (1)   length =
(nx+2)* (ny+2)* (nz+2)* (10+isx+jsy+ksz)
c
c           or (for method.gt.7)
c
c           (2)   length = 14* (nx+2)* (ny+2)* (nz+2)
c

```

```
c           will usually but not always suffice.  The
c           exact minimal length depends,
c           in a complex way, on the grid size arguments
c           and method chosen.
c ***      It can be predetermined for the current input
c           arguments by calling
c           mud3 with length=iparm(21)=0 and printing
c           iparm(22) or (in f90)
c           dynamically allocating the work space using
c           the value in iparm(22)
c           in a subsequent mud3 call.
c
c
c ... fparm
c
c           a floating point vector of length 8 used to
c           efficiently
c           pass floating point arguments.  fparm is set
c           internally
c           in mud3 and defined as follows . . .
c
c
c
c ... xa=fparm(1), xb=fparm(2)
c
c           the range of the x independent variable.  xa
c           must
c           be less than xb
c
c
c
c ... yc=fparm(3), yd=fparm(4)
c
c           the range of the y independent variable.  yc
c           must
c           be less than yd.
c
c
c
c ... ze=fparm(5), zf=fparm(6)
c
c           the range of the z independent variable.  ze
c           must
c           be less than zf.
c
c
c
c ... tolmax = fparm(7)
c
```

```

c      when input positive, tolmax is a maximum
relative error tolerance
c      used to terminate the relaxation iterations.
assume phi1(i,j,k)
c      and phi2(i,j,k) are the last two computed
approximations at all
c      grid points of the finest grid level. if we
define
c
c      phdif = max(abs(phi2(i,j,k)-phi1(i,j,k)))
for all i,j,k
c
c      and
c
c      phmax = max(abs(phi2(i,j,k))) for all
i,j,k
c
c      then "convergence" is considered to have
occurred if and only if
c
c      phdif/phmax < tolmax.
c
c
c      if tolmax=fparm(7)=0.0 is input then there is
no error control
c      and maxcy cycles from the finest grid level
are executed. maxcy
c      is a limit which cannot be exceeded even with
error control.
c      *** calls with tolmax=0.0, when appropriate
because of known
c      convergence behavior, are more efficient than
calls with tolmax
c      positive (i.e., if possible DO NOT use error
control!).
c
c
c ... work
c
c      a one dimensional array that must be provided
for work space.
c      see length = iparm(21). the values in work
must be preserved
c      if mud3 is called again with

```

```

intl=iparm(1).ne.0 or if mud34
c           is called to improve accuracy.
c
c
c ... bndyc
c
c           a subroutine with arguments
(kbdy,xory,yorz,alfa,gbdy).
c           which are used to input mixed boundary
conditions to mud3.
c           the boundaries are numbered one thru six and
the form of
c           conditions are described below.
c
c
c           (1) the kbdy=1 boundary
c
c           this is the (y,z) plane x=xa where nxa=iparm(2) =
2 flags
c           a mixed boundary condition of the form
c
c           dp/dx + alfxa(y,z)*p(xa,y,z) = gbdxa(y,z)
c
c           in this case kbdy=1,xory=y,yorz=z will be input to
bndyc and
c           alfa,gbdy corresponding to alfxa(y,z),gbdxa(y,z)
must be returned.
c
c
c           (2) the kbdy=2 boundary
c
c           this is the (y,z) plane x=xb where nxb=iparm(3) =
2 flags
c           a mixed boundary condition of the form
c
c           dp/dx + alfxb(y,z)*p(xb,y,z) = gbdxb(y,z)
c
c           in this case kbdy=2,xory=y,yorz=z will be input to
bndyc and
c           alfa,gbdy corresponding to alfxb(y,z),gbdxb(y,z)
must be returned.
c
c
c           (3) the kbdy=3 boundary

```

```

C
C      this is the (x,z) plane y=yc where nyc=iparm(4) =
2 flags
C      a mixed boundary condition of the form
C
C      dp/dy + alfyC(x,z)*p(x,yc,z) = gbdyc(x,z)
C
C      in this case kbdy=3,xory=x,yorz=z will be input to
bndyc and
C      alfa,gbdy corresponding to alfyC(x,z),gbdyc(x,z)
must be returned.
C
C
C      (4) the kbdy=4 boundary
C
C      this is the (x,z) plane y=yd where nyd=iparm(5) =
2 flags
C      a mixed boundary condition of the form
C
C      dp/dy + alfyd(x,z)*p(x,yd,z) = gbdyd(x,z)
C
C      in this case kbdy=4,xory=x,yorz=z will be input to
bndyc and
C      alfa,gbdy corresponding to alfyd(x,z),gbdyd(x,z)
must be returned.
C
C
C      (5) the kbdy=5 boundary
C
C      this is the (x,y) plane z=ze where nze=iparm(6) =
2 flags
C      a mixed boundary condition of the form
C
C      dp/dz + alfze(x,y)*p(x,y,ze) = gbdze(x,y)
C
C      in this case kbdy=5,xory=x,yorz=y will be input to
bndyc and
C      alfa,gbdy corresponding to alfze(x,y),gbdze(x,y)
must be returned.
C
C
C      (6) the kbdy=6 boundary
C
C      this is the (x,y) plane z=zf where nzf=iparm(7) =

```

```

2 flags
c      a mixed boundary condition of the form
c
c      dp/dz + alfzf(x,y)*p(x,y,zf) = gbdzf(x,y)
c
c      in this case kbdy=6,xory=x,yorz=y will be input to
bndyc and
c      alfa,gbdy corresponding to alfzf(x,y),gbdzf(x,y)
must be returned.
c
c
c *** alfxa,alfyc,alfze nonpositive and
alfxb,alfyd,alfze nonnegative
c      will help maintain matrix diagonal dominance
during discretization
c      aiding convergence.
c
c *** bndyc must provide the mixed boundary condition
c      values in correspondence with those flagged in
iparm(2)
c      thru iparm(7). if all boundaries are specified
then
c      mud3 will never call bndyc. even then it must be
entered
c      as a dummy subroutine. bndyc must be declared
c      external in the routine calling mud3. the actual
c      name chosen may be different.
c
c
c ... coef
c
c      a subroutine with arguments
(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
c      which provides the known real coefficients for
the elliptic pde
c      at any grid point (x,y,z). the name chosen in
the calling routine
c      may be different where the coefficient routine
must be declared
c      external.
c
c ... rhs
c
c      an array dimensioned nx by ny by nz which

```

```
contains
c           the given right hand side values on the
uniform 3-d mesh.
c           rhs(i,j,k) = r(xi,yj,zk) for i=1,...,nx and
j=1,...,ny
c           and k=1,...,nz.
c
c ... phi
c
c           an array dimensioned nx by ny by nz .  on
input phi must
c           contain specified boundary values and an
initial guess
c           to the solution if flagged (see
iguess=iparm(17)=1).  for
c           example, if nyd=iparm(5)=1 then phi(i,ny,k)
must be set
c           equal to p(xi,yd,zk) for i=1,...,nx and
k=1,...,nz prior to
c           calling mud3.  the specified values are
preserved by mud3.
c
c ***      if no initial guess is given (iguess=0) then
phi must still
c           be initialized at non-Dirchlet grid points
(this is not
c           checked).  these values are projected down and
serve as an initial
c           guess to the pde at the coarsest grid level.
set phi to 0.0 at
c           nonDirchlet grid points if nothing better is
available.
c
c
c ... mgopt
c
c           an integer vector of length 4 which allows
the user to select
c           among various multigrid options.  if
mgopt(1)=0 is input then
c           a default set of multigrid arguments (chosen
for robustness)
c           will be internally selected and the
remaining values in mgopt
```

```
c           will be ignored. if mgopt(1) is nonzero
then the arguments
c           in mgopt are set internally and defined as
follows: (see the
c           basic coarse grid correction algorithm
below)
c
c
c      kcycle = mgopt(1)
c
c      = 0 if default multigrid options are to be
used
c
c      = 1 if v cycling is to be used (the least
expensive per cycle)
c
c      = 2 if w cycling is to be used (the
default)
c
c      > 2 if more general k cycling is to be used
c          *** warning--values larger than 2 increase
c              the execution time per cycle
considerably and
c          result in the nonfatal error ierror =
-5
c          which indicates inefficient multigrid
cycling.
c
c      iprer = mgopt(2)
c
c          the number of "pre-relaxation" sweeps
executed before the
c          residual is restricted and cycling is
invoked at the next
c          coarser grid level (default value is 2
whenever mgopt(1)=0)
c
c      ipost = mgopt(3)
c
c          the number of "post relaxation" sweeps
executed after cycling
c          has been invoked at the next coarser grid
level and the residual
c          correction has been transferred back
```

```
(default value is 1
c           whenever mgopt(1)=0) .
c
c *** if iprер, ipost, or (especially) kcycle is greater
than 2
c       than inefficient multigrid cycling has probably
been chosen and
c       the nonfatal error (see below) ierror = -5 will be
set. note
c       this warning may be overridden by any other
nonzero value
c       for ierror.
c
c   interpol = mgopt(4)
c
c           = 1 if multilinear prolongation
(interpolation) is used to
c           transfer residual corrections and the pde
approximation
c           from coarse to fine grids within full
multigrid cycling.
c
c           = 3 if multicubic prolongation
(interpolation) is used to
c           transfer residual corrections and the pde
approximation
c           from coarse to fine grids within full
multigrid cycling.
c           (this is the default value whenever
mgopt(1)=0).
c
c *** the default values (2,2,1,3) in the vector mgopt
were chosen for
c       robustness. in some cases v(2,1) cycles with
linear prolongation will
c       give good results with less computation
(especially in two-dimensions).
c       this was the default and only choice in an
earlier version of mudpack
c       (see [1]) and can be set with the integer vector
(1,2,1,1) in mgopt.
c
c *** the schedules for one full multigrid cycle
(iguess=0) using v(2,1)
```

```

c      cycles and w(2,1) cycles are depicted for a four
level grid below.
c      the number of relaxation sweeps when each grid is
visited are indicated.
c      the "*" stands for prolongation of the full
approximation and the "."
c      stands for transfer of residuals and residual
corrections within the
c      coarse grid correction algorithm (see below).
c
c      one fmg with v(2,1) cycles:
c
c
c      -----
c      -----2-----1-----1-
c      level 4
c
c      *
c      .
c
c      -----
c      -----2-----1-----2-----1-----1-
c      level 3
c
c      *
c      .
c
c      -----
c      -----2-----1-----2-----1-----2-----1-----
c      level 2
c
c      *
c      .
c
c      -----
c      -----3-----3-----3-----3-----
c      level 1
c
c
c      one fmg with w(2,1) cycles:
c
c
c      -----
c      -----2-----
c      --1-- level 4
c
c      *
c      .
c
c      -----
c      -----2-----1-----2-----3-----
c      1---- level 3
c
c      *
c      .
c
c      -----
c      -----2-----1-----2-----3-----1-----2-----3-----1-
c      level 2
c
c      *
c      .
c
c      -----
c      -----6-----6-----6-----6-----6-----6-----6-----
c      level 1
c

```

```

c
c      the form of the "recursive" coarse grid correction
cycling used
c      when kcycle.ge.0 is input is described below in
pseudo-algorithmic
c      language. it is implemented non-recursively in
fortran in mudpack.
c
c      algorithm
cgc(k,l(k),u(k),r(k),kcycle,iprer,ipost,ireshape,intpol)
c
c *** approximately solve l(k)*u(k) = r(k) using
multigrid iteration
c *** k is the current grid level
c *** l(k) is the discretized pde operator at level k
c *** u(k) is the initial guess at level k
c *** r(k) is the right hand side at level k
c *** i(k,k-1) is the restriction operator from level k
to level k-1
c *** (the form of i(k,k-1) depends on ireshape)
c *** i(k-1,k) is the prolongation operator from level
k-1 to level k
c *** (the form of i(k-1,k) depends on intpol)
c
c      begin algorithm cgc
c
c *** pre-relax at level k
c
c      . do (i=1,iprer)
c
c      . . relax(l(k),u(k),r(k))
c
c      . end do
c
c      . if (k > 1) then
c
c ***      restrict the residual from level k to level k-
1
c
c      . . r(k-1) = i(k,k-1) (r(k)-l(k)*u(k) )
c
c      . . kount = 0
c
c      . . repeat

```

```

C
C ***      solve for the residual correction at level k-1
in u(k-1)
C ***      using algorithm cgc "kcycle" times (this is
the recursion)
C
C      . . . kount = kount+1
C
C      . . . invoke cgc(k-1,l(k-1),u(k-1),r(k-
1),kcycle,iprер,ipost,iresw)
C
C
C      . . until (kount.eq.kcycle)
C
C ***      transfer residual correction in u(k-1) to
level k
C ***      with the prolongation operator and add to u(k)
C
C      . . u(k) = u(k) + i(k-1,k) (u(k-1))
C
C      . end if
C
C ***      post relax at level k
C
C      . do (i=1,ipost)
C
C      . . relax(l(k),u(k),r(k))
C
C      . end do
C
C      . return
C
C      end algorithm cgc
C
C
*****
*****output
arguments*****
***
```

\*\*\*\*\*

```

C
```

```

c
c ... iparm(22)
c
c          on output iparm(22) contains the actual work
space length
c          required for the current grid sizes and
method. This value
c          will be computed and returned even if
iparm(21) is less than
c          iparm(22) (see ierror=9).
c
c
c ... iparm(23)
c
c          if error control is selected (tolmax =
fparm(7) .gt. 0.0) then
c          on output iparm(23) contains the actual
number of cycles executed
c          between the coarsest and finest grid levels
in obtaining the
c          approximation in phi. the quantity
(iprter+ipost)*iparm(23) is
c          the number of relaxation sweeps performed at
the finest grid level.
c
c
c ... fparm(8)
c
c          on output fparm(8) contains the final
computed maximum relative
c          difference between the last two iterates at
the finest grid level.
c          fparm(8) is computed only if there is error
control (tolmax.gt.0.)
c          assume phi1(i,j,k) and phi2(i,j,k) are the
last two computed
c          values for phi(i,j,k) at all points of the
finest grid level.
c          if we define
c
c          phdif = max (abs(phi2(i,j,k)-phi1(i,j,k)))
for all i,j,k
c
c          and

```

```

C
C           phmax = max (abs (phi2 (i,j,k) )) for all
i,j,k
C
C           then
C
C           fparm(8) = phdif/phmax
C
C           is returned whenever phmax.gt.0.0.  in the
degenerate case
C           phmax = 0.0, fparm(8) = phdif is returned.
C
C
C
C ... work
C
C           on output work contains intermediate values
that must not be
C           destroyed if mud3 is to be called again with
iparm(1)=1 or
C           if mud34 is to be called to improve the
estimate to fourth
C           order.
C
C
C ... phi
C
C           on output phi(i,j,k) contains the
approximation to
C           p(xi,yj,zk) for all mesh points i=1,...,nx;
j=1,...,ny;
C           k=1,...,nz.  the last computed iterate in phi
is returned
C           even if convergence is not obtained (ierror=-
1)
C
C
C ... ierror
C
C           For intl=iparm(1)=0 initialization calls,
ierror is an
C           error flag that indicates invalid input
arguments when
C           returned positive and nonfatal warnings when
returned
C           negative. Argument checking and

```

```

discretization
c           is bypassed for intl=1 calls which can only
return
c           ierror = -1 or 0 or 1.
c
c
c     non-fatal warnings * * *
c
c
c     ==5 if kcycle=mgopt(1) is greater than 2. values
larger than 2 results
c           in an algorithm which probably does far more
computation than
c           necessary.  kcycle = 1 (v cycles)  or kcycle=2
(w cycles) should
c           suffice for most problems.  ierror = -5 is
also set if either
c           iprer = mgopt(2) or ipost=mgopt(3) is greater
than 2.  the
c           ierror=-5 flag is overridden by any other
fatal or non-fatal
c           error.
c
c     ==4 if there are dominant nonzero first order
terms in the pde which
c           make it "hyperbolic" at the finest grid level.
numerically, this
c           happens if:
c
c           abs(cx)*dlx > 2.*abs(cxx)      (dlx = (xb-
xa) / (nx-1))
c
c                           (or)
c
c           abs(cy)*dly > 2.*abs(cyy)      (dly = (yd-
yc) / (ny-1))
c
c
c           at some fine grid point (xi,yj).  if an
adjustment is not made the
c           condition can lead to a matrix coming from the
discretization
c           which is not diagonally dominant and
divergence is possible. since

```

```

c      the condition is "likely" at coarser grid
levels for pde's with
c      nonzero first order terms, the adjustments
(actually first order
c      approximations to the pde)
c
c
c      cxx = amax1(cxx,0.5*abs(cx)*dx)
c
c      (and)
c
c      cyy = amax1(cyy,0.5*abs(cy)*dy)
c
c
c      (here dx,dy are the x,y mesh sizes of the
subgrid)
c
c      are made to preserve convergence of multigrid
iteration. if made
c      at the finest grid level, it can lead to
convergence to an
c      erroneous solution (flagged by ierror = -4).
a possible remedy
c      is to increase resolution. the ierror = -4
flag overrides the
c      nonfatal ierror = -5 flag.
c
c
c      ==3 if the continuous elliptic pde is singular.
this means the
c      boundary conditions are periodic or pure
derivative at all
c      boundaries and ce(x,y) = 0.0 for all x,y. a
solution is still
c      attempted but convergence may not occur due
to ill-conditioning
c      of the linear system coming from the
discretization. the
c      ierror = -3 flag overrides the ierror=-4 and
ierror=-5 nonfatal
c      flags.
c
c
c      ==2 if the pde is not elliptic (i.e.,

```

```

cxxx*cyy.le.0.0 for some (xi,yj))
c           in this case a solution is still attempted
although convergence
c           may not occur due to ill-conditioning of the
linear system.
c           the ierror = -2 flag overrides the ierror=-
5,-4,-3 nonfatal
c           flags.

c
c
c     ==1 if convergence to the tolerance specified in
tolmax=fparm(5)>0.
c           is not obtained in maxcy=iparm(13) multigrid
cycles between the
c           coarsest (ixp+1,jyq+1) and finest (nx,ny)
grid levels.
c           in this case the last computed iterate is
still returned.
c           the ierror = -1 flag overrides all other
nonfatal flags

c
c
c     no errors * * *
c
c     = 0
c
c     fatal argument errors * * *
c
c     = 1 if intl=iparm(1) is not 0 on initial call or
not 0 or 1
c           on subsequent calls
c
c     = 2 if any of the boundary condition flags
nxa,nxb,nyc,nyd,nze,nzf
c           in iparm(2) through iparm(7) is not 0,1 or 2 or
if
c           (nxa,nxb) or (nyc,nyd) or (nze,nzf) are not
pairwise zero.
c
c     = 3 if mino(ixp,jyq,kzr) < 2
(ixp=iparm(8),jyq=iparm(9),kzr=iparm(10))
c
c     = 4 if min0(iex,jey,kez) < 1
(iex=iparm(11),jey=iparm(12),kez=iparm(13))

```

```

C      or if max0(iex,jey,kez) > 50
C
C      = 5 if nx.ne.ipx*2** (iex-1)+1 or if
ny.ne.jyq*2** (jey-1)+1 or
C          if nz.ne.kzr*2** (kez-1)+1
(nx=iparm(14),ny=iparm(15),nz=iparm(16))
C
C      = 6 if iguess = iparm(17) is not equal to 0 or 1
C
C      = 7 if maxcy = iparm(18) < 1 (large values for
maxcy should not be used)
C
C      = 8 if method = iparm(19) is less than 0 or
greater than 10 or
C          if meth2 = iparm(20) is not 0 or 1 or 2 or 3
when method > 7.
C
C      = 9 if length = iparm(20) is too small (see
iparm(21) on output
C          for minimum required work space length)
C
C      =10 if xa >= xb or yc >= yd or ze >= zf
C
C      (xa=fparm(1),xb=fparm(2),yc=fparm(3),yd=fparm(4),ze=fpar
m(5),zf=fparm(6))
C
C      =11 if tolmax = fparm(7) < 0.0
C
C      errors in setting multigrid options * * * (see
also ierror=-5)
C
C      =12 if kcycle = mgopt(1) < 0 or
C          if iprer = mgopt(2) < 1 or
C          if ipost = mgopt(3) < 1 or
C          if interpol = mgopt(4) is not 1 or 3
C
C
*****
*
C
*****
*
C
C      end of mud3 documentation

```

```
C
C
*****
** 
C
*****
** 
C
C
```

---

## MUD34

```
C
C      file mud34.d
C
C      * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *           copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *           all rights reserved
*
C      *
*
C      *           MUDPACK version 5.0.1
*
C      *
*
C      *           A Fortran Package of Multigrid
*
C      *
```

```
*  
C      *          Subroutines and Example Programs  
*  
C      *  
*  
C      *      for Solving Elliptic Partial Differential  
Equations      *  
C      *  
*  
C      *          by  
*  
C      *  
*  
C      *          John Adams  
*  
C      *  
*  
C      *          of  
*  
C      *  
*  
C      *          the National Center for Atmospheric  
Research      *  
C      *  
*  
C      *          Boulder, Colorado (80307)  
U.S.A.      *  
C      *  
*  
C      *          which is sponsored by  
*  
C      *  
*  
C      *          the National Science Foundation  
*  
C      *  
*  
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C  
C ... file mud34.d  
C  
C      contains documentation for subroutine  
mud34 (work,phi,ierror)
```

```
c      A sample fortran driver is file "tmud34.f".
c
c ... required MUDPACK files
c
c      mud3.f, mudcom.f, mud3ln.f, mud3pn.f
c
c ... purpose
c
c      mud34 attempts to improve the estimate in phi,
c obtained by calling
c      mud3, from second to fourth order accuracy. see
c the file "mud3.d"
c      for a detailed discussion of the elliptic pde
c approximated and
c      arguments "work,phi" which are also part of the
c argument list for
c      mud3.
c
c ... assumptions
c
c      * phi contains a second-order approximation from
c an earlier mud3 call
c
c      * arguments "work,phi" are the same used in
c calling mud3
c
c      * "work,phi" have not changed since the last call
c to mud3
c
c      * the finest grid level contains at least 6
c points in each direction
c
c
c *** warning
c
c      if the first assumption is not true then a fourth
c order approximation
c      cannot be produced in phi. the last assumption
c (adequate grid size)
c      is the only one checked. failure in any of the
c others can result in
c      in an undetectable error.
c
c ... language
```

```
c
c      fortran90/fortran77
c
c ... portability
c
c      mudpack5.0.1 software has been compiled and tested
with fortran77
c      and fortran90 on a variety of platforms.
c
c ... methods
c
c      details of the methods employed by the solvers in
mudpack are given
c      in [1,9]. [1,2,9] contain performance
measurements on a variety of
c      elliptic pdes (see "references" in the file
"readme"). in summary:
c
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
estimate (i.e.,
c      discretization level error is reached) then this
can be improved to a
c      fourth-order estimate using the technique of
"deferred corrections"
c      the values in the solution array are used to
generate a fourth-order
c      approximation to the truncation error. second-
order finite difference
c      formula are used to approximate third and fourth
partial derivatives
c      of the solution function [3]. the truncation
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side. the default
multigrid options
c      are used to compute the fourth-order correction
term which is added
c      to the original solution array.
c
```

```
C
C ... references (partial)
C
C
C      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
C      Solution of Linear Elliptic Partial Differential
Equations,"
C      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
C
C      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
C      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
C      1989, pp.1-12.
C      .
C      .
C      .
C      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
Haidvogel, and V. Pizzo,
C      "Applications of Multigrid Software in the
Atmospheric Sciences,"
C      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-
1458.
C      .
C      .
C      .
C      [9] J. Adams, "Recent Enhancements in MUDPACK, a
Multigrid Software
C      package for Elliptic Partial Differential
Equations," Applied Math.
C      and Comp., 1991, vol. 43, May 1991, pp. 79-94.
C
C      [10]J. Adams, "MUDPACK-2: Multigrid Software for
Approximating
C      Elliptic Partial Differential Equations on Uniform
Grids with
C      any Resolution," Applied Math. and Comp., 1993,
vol. 53, February
C      1993, pp. 235-249
C      .
C      .
C      .
```

```
c ... error argument
c
c      = 0 if no error is detected
c
c      = 30 if min0(nx,ny) < 6 where nx,ny are the fine
grid sizes
c          in the x,y directions.
c
c
c
*****
*****
```

C

```
*****
```

C

```
*****
```

C

```
*****
```

C end of mud34 documentation

C

```
*****
```

C

```
*****
```

C

```
*****
```

C

```
*****
```

C

MUD34SP

```
C  
C      file mud34sp.d  
C  
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * * *  
C      *  
*  
C      *          copyright (c) 2008 by UCAR  
*
```

```
C      *
*
C      *      University Corporation for Atmospheric
Research          *
C      *
*
C      *      all rights reserved
*
C      *
*
C      *      MUDPACK  version 5.0.1
*
C      *
*
C      *      A Fortran Package of Multigrid
*
C      *
*
C      *      Subroutines and Example Programs
*
C      *
*
C      *      for Solving Elliptic Partial Differential
Equations      *
C      *
*
C      *      by
*
C      *
*
C      *      John Adams
*
C      *
*
C      *      of
*
C      *
*
C      *      the National Center for Atmospheric
Research          *
C      *
*
C      *      Boulder, Colorado (80307)
U.S.A.          *
```

```
C      *
*
C      *                               which is sponsored by
*
C      *
*
C      *                               the National Science Foundation
*
C      *
*
C      * * * * * * * * * * * * * * * *
* * * * * * * *
C
C ... file mud34sp.d
C
C     contains documentation for subroutine
mud34sp(work,phi,ierror)
C     A sample fortran driver is file "tmud34sp.f".
C
C ... required MUDPACK files
C
C     mud3sp.f, mudcom.f
C
C ... purpose
C
C     mud34sp attempts to improve the estimate in phi,
obtained by calling
C     mud3sp, from second to fourth order accuracy.
see the file "mud3sp.d"
C     for a detailed discussion of the elliptic pde
approximated and
C     arguments "work,phi" which are also part of the
argument list for
C     mud3sp.
C
C ... assumptions
C
C      * phi contains a second-order approximation from
an earlier mud3sp call
C
C      * arguments "work,phi" are the same used in
calling mud3sp
C
C      * "work,phi" have not changed since the last call
```

```
to mud3sp
c
c      * the finest grid level contains at least 6
points in each direction
c
c
c *** warning
c
c      if the first assumption is not true then a fourth
order approximation
c      cannot be produced in phi. the last assumption
(adequate grid size)
c      is the only one checked. failure in any of the
others can result in
c      in an undetectable error.
c
c ... language
c
c      fortran90/fortran77
c
c ... portability
c
c      mudpack5.0.1 software has been compiled and tested
with fortran77
c      and fortran90 on a variety of platforms.
c
c ... methods
c
c      details of the methods employed by the solvers in
mudpack are given
c      in [1,9]. [1,2,9] contain performance
measurements on a variety of
c      elliptic pdes (see "references" in the file
"readme"). in summary:
c
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
estimate (i.e.,
c      discretization level error is reached) then this
can be improved to a
c      fourth-order estimate using the technique of
```

```
"deferred corrections"
c      the values in the solution array are used to
generate a fourth-order
c      approximation to the truncation error. second-
order finite difference
c      formula are used to approximate third and fourth
partial derivatives
c      of the solution function [3]. the truncation
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side. the default
multigrid options
c      are used to compute the fourth-order correction
term which is added
c      to the original solution array.

c
c
c ... references (partial)

c
c
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
c      Solution of Linear Elliptic Partial Differential
Equations,"
c      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.

c
c      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
c      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
c      1989, pp.1-12.

c      .
c      .
c      .

c      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
Haidvogel, and V. Pizzo,
c      "Applications of Multigrid Software in the
Atmospheric Sciences,"
c      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-
1458.

c      .
c      .
```

```
C .
C      [9] J. Adams, "Recent Enhancements in MUDPACK, a
Multigrid Software
C      package for Elliptic Partial Differential
Equations," Applied Math.
C      and Comp., 1991, vol. 43, May 1991, pp. 79-94.
C
C      [10]J. Adams, "MUDPACK-2: Multigrid Software for
Approximating
C      Elliptic Partial Differential Equations on Uniform
Grids with
C      any Resolution," Applied Math. and Comp., 1993,
vol. 53, February
C      1993, pp. 235-249
C .
C .
C .
C ...
C ... error argument
C
C      = 0 if no error is detected
C
C      = 30 if min0(nx,ny) < 6 where nx,ny are the fine
grid sizes
C          in the x,y directions.
C
C
C ****
*****
C ****
*****
C
C ****
*****
C
C      end of mud34sp documentation
C
C ****
*****
C ****
*****
C
C ****
*****
```

---

## MUD3CR

```
C
C      file mud3cr.d
C
C      * * * * * * * * * * * * * * * *
* * * * * * *
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *               University Corporation for Atmospheric
Research           *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK version 5.0.1
*
C      *
*
C      *               A Fortran Package of Multigrid
*
C      *
*
C      *               Subroutines and Example Programs
*
C      *
*
C      *               for Solving Elliptic Partial Differential
Equations        *
C      *
*
C      *               by
*
C      *
```

```
*  
C      *                                John Adams  
*  
C      *  
*  
C      *                                of  
*  
C      *  
*  
C      *          the National Center for Atmospheric  
Research           *  
C      *  
*  
C      *          Boulder, Colorado (80307)  
U.S.A.           *  
C      *  
*  
C      *          which is sponsored by  
*  
C      *  
*  
C      *          the National Science Foundation  
*  
C      *  
*  
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C  
C ... file mud3cr.d  
C  
C     contains documentation for:  
C     subroutine  
mud3cr(iparm,fparm,work,coef,bnd3cr,rhs,phi,mgopt,  
C  
+icros,crsxy,crsxz,crsyz,tol,maxit,iouter,rmax,ierror)  
C     A sample fortran driver is file "tmud3cr.f".  
C  
C ... required MUDPACK files  
C  
C     mudcom.f  
C  
C  
C ... purpose  
C
```

```

c      subroutine mud3cr automatically discretizes and
attempts to compute
c      the second order finite difference approximation
to a three-
c      dimensional linear nonseparable elliptic partial
differential
c      equation with cross derivative terms on a box.
the approximation
c      is generated on a uniform grid covering the box
(see mesh description
c      below). boundary conditions may be any
combination of oblique mixed
c      derivative (see bnd3cr description below),
specified (Dirchlet) or
c      periodic. the form of the pde in operator
notation is
c
c      l(p) + lxxyz(p) = r(x,y,z)
c
c      where
c
c      l(p) = cxx(x,y,z)*pxx + cyy(x,y,z)*pyy +
czz(z,y,z)*pzz +
c
c                  cx(x,y,z)*px + cy(x,y,z)*py +
cz(x,y,z)*pz +
c
c                  ce(x,y,z)*p(x,y,z) = r(x,y,z)
c
c      and
c
c      lxxyz(p) = cxy(x,y,z)*pxy + cxz(x,y,z)*pxz +
cyz(x,y,z)*pyz
c
c      here cxx,cyy,czz,cx,cy,cz,ce,cxy,cxz,cyz are the
known real
c      coefficients of the pde; pxx,pyy,pzz,px,py,pz are
the second and
c      first partial derivatives of the unknown solution
function p(x,y,z)
c      with respect to the independent variables x,y,z;
pxy,pxz, and pyz
c      are the second order mixed partial derivatives of
p with respect

```

```

c      to xy,xz, and yz.  r(x,y,z) is the known right
hand side of the pde.
c
c
c ... mesh description . . .
c
c      the approximation is generated on a uniform nx by
ny by nz grid.
c      the grid is superimposed on the rectangular
solution region
c
c            [xa,xb] x [yc,yd] x [ze,zf].
c
c      let
c
c      dlx = (xb-xa) / (nx-1), dly = (yd-yc) / (ny-1),
dlz = (zf-ze) / (nz-1)
c
c      be the uniform grid increments in the x,y,z
directions. then
c
c            xi=xa+(i-1)*dlx,   yj=yc+(j-1)*dly,  zk =
ze+(k-1)*dlz
c
c      for i=1,...,nx; j=1,...,ny; k=1,...,nz  denote the
x,y,z uniform
c      mesh points.
c
c
c
c ... methods
c
c
c      subroutine mud3cr is a recent addition to mudpack.
details
c      of the methods employed by the other solvers in
mudpack are in
c      [1,9,10].  [1,2,7,9,10] contain performance
measurements on a variety
c      of elliptic pdes (see "references" in the file
"readme").  the multi-
c      grid methods are described in documentation for
the other solvers.
c *** mud3cr differs fundamentally from the other

```

solvers in mudpack.  
 c        the full pde including cross derivative terms is  
 discretized on  
 c        the INTERIOR of the solution region:  
 c  
 c         $xa < x < xb, yc < y < yd, ze < z < zf$   
 c  
 c        however, on nonspecified (nondirichlet) boundaries  
 only  $l(p)$  is  
 c        discretized and the cross derivative term  $lxyz(p)$   
 is moved to the  
 c        right hand side of the pde and approximated by  
 second order finite  
 c        finite difference formula applied to a previous  
 estimate in  $p(k-1)$ .  
 c        similarly, oblique mixed derivative boundary  
 conditions (see bnd3cr)  
 c        are converted to a "mud3" type mixed normal form  
 using second-order  
 c        finite difference formula applied to a previous  
 estimate  $p(k-1)$  to  
 c        approximate non-normal derivative components. for  
 example if  
 c        the mixed derivative condition  
 c  
 c         $py + a(x, z)*px + b(x, z)*pz + c(x, z)*p(x, yd, z) =$   
 $gyd(y, z)$   
 c  
 c        is specified on the  $(x, z)$  plane of the upper  $y=yd$   
 boundary (see  
 c        bnd3cr for  $kbdy=4$  below) then mud3cr converts this  
 to the mixed  
 c        normal derivative form  
 c  
 c         $py + c(x, z)*p(x, yd, z) = h(k, x, z)$   
 c  
 c        where the modified right hand side  $h(k, x, z)$  is  
 given by  
 c  
 c         $h(k, x, z) = gyd(x, z) - [a(x, z)*dx(p(k-1)) +$   
 $b(x, z)*dz(p(k-1))]$ .  
 c  
 c         $dx(p(k-1))$  and  $dz(p(k-1))$  are second order finite  
 difference

```

c      approximations to the nonnormal partial
derivatives px,pz using the
c      previous estimate in p(k-1).
c
c      the result of full discretization on interior grid
points and partial
c      discretization with right hand side modifications
on boundaries,
c      is a linear system which we denote by
c
c      D(p(k)) = r - Dxyz(p(k-1)).
c
c      D is the coefficient matrix coming from the
discretization and
c      Dxyz(p(k-1)) stands for the right hand side
modification obtained
c      by approximating boundary cross derivative terms
and/or nonnormal
c      derivative components from mixed derivative
boundary conditions
c      with second order finite difference formula
applied to p(k-1).
c      with this notation, we formally describe the outer
iteration employeed
c      by mud3cr:
c
c      algorithm mud3cr
c      .
c      set k = 0
c      .
c      set p(0) = 0.0 for all nonspecified grid points
c      .
c      repeat
c
c      .. k = k+1
c
c      .. solve D(p(k)) = r - Dxyz(p(k-1)) using
multigrid iteration
c
c      .. set rmax(k) = ||p(k) - p(k-1)|| / ||p(k) ||
c
c      until (rmax(k) < tol or k = maxit)
c      .
c      end mud3cr

```

```
c
c      tol is an error tolerance for convergence and
maxit is a limit on
c      the number of outer iterations. both are user
prescribed input
c      arguments to mud3cr. the maximum vector norm |||
||| is used in
c      computing the relative difference between
successive estimates in
c      rmax(k). large values for maxit should not be
used.

c
c *** note
c
c      originally a code, mud3cr0, was designed by moving
all cross terms
c      to the right hand side and solving
c
c      l(p(k)) = r - lxyz(p(k-1)) for k=1, ...
c
c      over the entire solution region including the
interior. in this
c      relatively straightforward approach, the standard
mudpack solver
c      mud3 is used iteratively at each step. however,
convergence with
c      mud3cro is slow and unreliable. an attempt was
then made to
c      discretize the complete pde including cross terms
and boundary
c      conditions over the entire region like the other
solvers in mudpack.
c      undoubtedly, this would have the most efficient
and robust convergence
c      properties. the main difficulty with this
approach is the
c      unmanageable code complexity required to
discretize all possible
c      combinations of nonzero cross derivative terms and
oblique derivative
c      conditions. for example, in the presence of
nonzero cross terms,
c      corners which are at the intersections of oblique
derivative boundary
```

c        conditions become too complex (for this person) to  
discretize.

c        detection of combinations of oblique derivative  
conditions at corners

c        which are singular (i.e., for which discretization  
leads to division  
c        by 0.0) is a logical nightmare.

c

c        the present mud3cr is a middle ground between  
these two approaches.

c        it is an attempt to include as much of the pde  
cross terms as possible

c        in the discretization while bypassing the code  
complexity required to

c        include all possible boundary situations in the  
discretization.

c

c        by including the (manageable) discretization of  
cross terms on the

c        interior, the unfavorable convergence properties  
of the first approach

c        are (hopefully) avoided and the favorable  
convergence properties

c        of the second approach are (hopefully) obtained.  
by moving nonzero

c        boundary cross terms and nonnormal components of  
mixed derivative

c        boundary conditions to the right hand side, the  
problems of too much

c        code complexity with discretization in the second  
approach are bypassed.

c

c        extensive numerical testing indicates for most  
problems mud3cr is more

c        robust than mud3cro. convergence is more "iffy"  
and computationally

c        expensive than with the other solvers in mudpack.

c

c        ... language

c

c        fortran90/fortran77

c

c

c        ... portability

```
C
C      mudpack5.0.1 software has been compiled and tested
with Fortran77
C      and Fortran90 on a variety of platforms.
C
C
C ... references (partial list)
C
C      for a complete list see "references" in the
mudpack information and
C      directory file "readme"
C
C      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
C      Solution of Linear Elliptic Partial Differential
Equations,"
C      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
C
C      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
C      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
C      1989, pp.1-12.
C      .
C      .
C      .
C      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
Haidvogel, and V. Pizzo,
C      "Applications of Multigrid Software in the
Atmospheric Sciences,"
C      Mon. Wea. Rev.,vol. 120 # 7, July 1992, pp. 1447-
1458.
C      .
C      .
C      .
C      [9] J. Adams, "Recent Enhancements in MUDPACK, a
Multigrid Software
C      package for Elliptic Partial Differential
Equations," Applied Math.
C      and Comp., 1991, vol. 43, May 1991, pp. 79-94.
C
C      [10]J. Adams, "MUDPACK-2: Multigrid Software for
Approximating
```

```

c      Elliptic Partial Differential Equations on Uniform
Grids with
c      any Resolution," Applied Math. and Comp., 1993,
vol. 53, February
c      1993, pp. 235-249
c      .
c      .
c      .
c      .
c ****
*****
c *** arguments
*****
c
*****
c      arguments iparm, fparm, work, rhs, phi, coef, mgopt are
the same as
c      those input to mud3 (see mud3.d for a detailed
description) with the
c      following provisions:
c
c      (1) the minimum required work space length for
mud3cr is increased
c          by approximately
c
c
nx*ny*nz*(1+8*(icros(1)+icros(2)+icros(3))/7 +
c
c
2*(icros(1)+icros(2)+icros(3))*(nx*ny+nx*nz+ny*nz)
c
c          words over the minimum work space required by
mud3 (see icros
c          description below). the exact minimal work
space required
c          by mud3cr for the current set of input
arguments is output
c          in iparm(22). * The exact minimal work length
required
c          for the current method and grid size arguments
can be

```

```
c      predetermined by calling mud3cr with
c      iparm(21)=0 and
c      printout of iparm(22) or (in fortran 90 codes)
c      dynamically
c      allocating work using the the value in
c      iparm(22) in subsequent
c      calls to mud3cr.
c
c      (2) at least two calls to mud3cr are necessary to
c      generate an
c      approximation. intl=iparm(1)=0 is required on
c      the first
c      call. this call will do "once only"
c      discretization, and
c      set intermediate values in work which must be
c      preserved
c      for noninitial calls.
c
c      (3) maxcy = iparm(18) must be 1 or 2 (see ierror =
c      13).
c
c      (4) tolmax = fparm(5) = 0.0 is required. no
c      "internal" error control
c      is allowed within multigrid cycling (see
mud3.d)
c
c      (5) mgopt(1) = 0 is required. only the default
multigrid
c      options (W(2,1) cycles with cubic
prolongation) can be used
c      with mud3cr
c
c *** new arguments
c
c      the arguments:
bnd3cr,icros,crsxy,crsxz,crsyz,tol,maxit,iouter,rmax
c      are all new to mud3cr. the error argument,
ierror, has been expanded.
c      these are all described below:
c
c
c ... bnd3cr(kbdy,xory,yorz,a,b,c,g)
c
c      a subroutine with input arguments kbdy,xory,yorz
```

```

and output
c      arguments a,b,c,g. bnd3cr inputs OBLIQUE mixed
derivative
c      conditions at any of the six x,y,z boundaries to
mud3cr as
c      described below:
c
c          (1) the kbdy=1 boundary
c
c          this is the (y,z) plane x=xa where
nxa=iparm(2)=2 flags
c          an oblique mixed boundary condition of the
form
c
c          px + axa(y,z)*py + bxa(y,z)*pz
+cxa(y,z)*p(xa,y,z) = gxa(y,z)
c
c          in this case kbdy=1,xory=y,yorz=z will be
input to bnd3cr and
c          a,b,c,g corresponding to the known
coefficients axa(y,z),bxa(y,z),
c          cxa(y,z),gxa(y,z) must be returned
c
c
c          (2) the kbdy=2 boundary
c
c          this is the (y,z) plane x=xb where
nxb=iparm(3)=2 flags
c          an oblique mixed boundary condition of the
form
c
c          px + axb(y,z)*py + bxb(y,z)*pz
+cxb(y,z)*p(xb,y,z) = gxb(y,z)
c
c          in this case kbdy=2,xory=y,yorz=z will be
input to bnd3cr and
c          a,b,c,g corresponding to the known
coefficients axb(y,z),bxb(y,z),
c          cxb(y,z),gxb(y,z) must be returned
c
c          (3) the kbdy=3 boundary
c
c          this is the (x,z) plane y=yc where
nyc=iparm(4)=2 flags

```

```

c      an oblique mixed boundary condition of the
form

c
c      py + ayc(x,z)*px + byc(x,z)*pz
+cyc(x,z)*p(x,yc,z) = gyc(x,z)
c
c      in this case kbdy=3,xory=x,yorz=z will be
input to bnd3cr and
c      a,b,c,g corresponding to the known
coefficients ayc(x,z),byc(x,z),
c      cyc(x,z),gyc(x,z) must be returned
c
c
c      (4) the kbdy=4 boundary
c
c      this is the (x,z) plane y=yd where
nyd=iparm(5)=2 flags
c      an oblique mixed boundary condition of the
form

c
c      py + ayd(x,z)*px + byd(x,z)*pz
+cyd(x,z)*p(x,yd,z) = gyd(x,z)
c
c      in this case kbdy=4,xory=x,yorz=z will be
input to bnd3cr and
c      a,b,c,g corresponding to the known
coefficients ayd(x,z),byd(x,z),
c      cyd(x,z),gyd(x,z) must be returned
c
c      (5) the kbdy=5 boundary
c
c      this is the (x,y) plane z=ze where
nze=iparm(6)=2 flags
c      an oblique mixed boundary condition of the
form

c
c      pz + aze(x,y)*px + bze(x,y)*py +
cze(x,y)*p(x,y,ze) = gze(x,y)
c
c      in this case kbdy=5,xory=x,yorz=y will be
input to bnd3cr and
c      a,b,c,g corresponding to the known
coefficients aze(x,y),bze(x,y),
c      cze(x,y),gze(x,y) must be returned

```

```

c
c          (6) the kbdy=6 boundary
c
c          this is the (x,y) plane z=zf where
nzf=iparm(7)=2 flags
c          an oblique mixed boundary condition of the
form
c
c          pz + azf(x,y)*px + bzf(x,y)*py +
czf(x,y)*p(x,y,zf) = gzf(x,y)
c
c          in this case kbdy=6,xory=x,yorz=y will be
input to bnd3cr and
c          a,b,c,g corresponding to the known
coefficients azf(x,y),bzf(x,y),
c          czf(x,y),gzf(x,y) must be returned
c
c
c          bnd3cr must be declared "external" in the routine
calling mud3cr
c          where its name may be different. bnd3cr must be
entered as a
c          dummy subroutine even if there are no derivative
boundary conditions.
c          for an example of how to set up a subroutine to
input derivative
c          boundary conditions, see the test program
tmud3cr.f
c
c ... icros
c
c          an integer vector argument dimensioned 3 which
flags the presence
c          or absence of cross derivative terms in the pde as
follows:
c
c          icros(1) = 1 if cxy(x,y,z) is nonzero for any
grid point (x,y,z)
c          icros(1) = 0 if cxy(x,y,z) = 0.0 for all grid
points (x,y,z)
c
c          icros(2) = 1 if cxz(x,y,z) is nonzero for any
grid point (x,y,z)
c          icros(2) = 0 if cxz(x,y,z) = 0.0 for all grid

```

```
points (x,y,z)
c
c      icros(3) = 1 if cyz(x,y,z) is nonzero for any
grid point (x,y,z)
c      icros(3) = 0 if cyz(x,y,z) = 0.0 for all grid
points (x,y,z)
c
c
c ... crsxy(x,y,z,cxy)
c
c      if icros(1) = 1 then crsxy is a subroutine with
arguments
c      (x,y,z,cxy) which supplies the xy cross derivative
coefficient
c      cxy at the grid point (x,y,z).  if icros(1) = 0
then crsxy
c      is a dummy subroutine argument (i.e., it must be
provided but
c      will not be invoked).
c
c
c ... crsxz(x,y,z,cxz)
c
c      if icros(2) = 1 then crsxz is a subroutine with
arguments
c      (x,y,z,cxz) which supplies the xz cross derivative
coefficient
c      cxz at the grid point (x,y,z).  if icros(2) = 0
then crsxz
c      is a dummy subroutine argument (i.e., it must be
provided but
c      will not be invoked).
c
c
c ... crsyz(x,y,z,cyz)
c
c      if icros(3) = 1 then crsyz is a subroutine with
arguments
c      (x,y,z,cyz) which supplies the yz cross derivative
coefficient
c      cxy at the grid point (x,y,z).  if icros(3) = 0
then crsyz
c      is a dummy subroutine argument (i.e., it must be
provided but
```

```
c      will not be invoked) .  
c  
c      crsxy,crsxz,crsyz must be declared "external" in  
the routine  
c      calling mud3cr. the names chosen for these  
routines can be  
c      different (see tmud3cr.f for an example)  
c  
c ... tol  
c  
c      tol is an error control argument for the outer  
iteration employed  
c      by mud3cr (see "methods" description above). if  
tol > 0.0 is input  
c      then tol is a relative error tolerance for  
convergence. the outer  
c      iteration terminates and convergence is deemed to  
have occurred at the  
c      k(th) iterate if the maximum relative difference,  
rmax(k), satisfies  
c  
c            def  
c            rmax(k) = ||p(k) - p(k-1)|| / ||p(k)|| < tol.  
c  
c      the last approximation p(maxit) is returned in phi  
even if  
c      convergence does not occur. the maximum norm ||  
|| is used.  
c      when tol = 0.0 is input, error control is not  
implemented and  
c      exactly maxit (see below) outer iterations are  
executed in mud3cr.  
c      the tol = 0.0 option eliminates unnecessary  
computation when  
c      the user is certain of the required value for  
maxit.  
c  
c  
c ... maxit  
c  
c      a limit on the outer iteration loop (see "method"  
description)  
c      used to approximate the 3-d pde with cross  
derivative terms when
```

```
c      tol > 0.0.  if tol = 0.0 is entered then exactly
maxit outer
c      iterations are performed and only rmax(maxit) is
computed.  the
c      total number of relaxation sweeps performed at the
finest grid
c      level is bounded by 3*maxcy*maxit.  large values
for maxit should
c      not be used.

C
C
C
*****
*****output
arguments*****
***

C
*****
*****iparm(22)
C
c      on output iparm(22) contains the actual work
space length
c      required by mud3cr for the current grid sizes
and method.
c      this will be approximately

C
nx*ny*nz*(1+8*(icros(1)+icros(2)+icros(3))/7 +
C
C
2*(icros(1)+icros(2)+icros(3))*(nx*ny+nx*nz+ny*nz)
C
c      words longer than the space required by mud3
(see mud3.d)
C
C
c ... work
C
c      on output work contains intermediate values
that must not be
```

```

c           destroyed if mud3cr is to be called again
with iparm(1)=1
c           and iparm(17)=1.
c
c
c ... phi
c
c           on output phi(i,j,k) contains the
approximation to
c           p(xi,yj,zk) for all mesh points i=1,...,nx;
j=1,...,ny;
c           k=1,...,nz. the last computed iterate in phi
is returned
c           even if convergence is not obtained.
c
c
c ... iouter
c
c           the number of outer iterations (see "method"
description above)
c           executed by mud3cr for the current call.
maxit is an upper bound
c           for iouter
c
c
c ... rmax (see tol,maxit descriptions above)
c
c           a maxit dimensioned real vector. if tol >
0.0 is input then
c           rmax(k) for k=1,...,iouter contain the
maximum relative
c           difference between successive estimates.
rmax(k) is
c           given by
c
c           rmax(k) = ||p(k) - p(k-1)|| / ||p(k) ||
c
c           for k=1,...,iouter. the maximum norm || ||
is used. either
c           iouter < maxit (convergence) or iouter =
maxit is possible.
c           if tol = 0.0 input then exactly maxit outer
iterations are
c           executed and only rmax(maxit) is computed.

```

```
in this case
c           rmax(1),...,rmax(maxit-1) are set to 0.0.
the tol = 0.0
c           option eliminates unnecessary computation
when the user is
c           certain of the required value for maxit.
c
c
c ... ierror
c
c           an integer error argument which indicates
fatal errors when
c           returned positive. the negative values -5,-
4,-3,-2,-1 and
c           ierror = 2,3,4,5,6,9,10 have the same meaning
as described for
c           for mud3 (see mud3.d). in addition:
c
c ***      new nonfatal error
c
c           ierror = -10 if tol > 0.0 is input (error
control) and convergence
c           fails in maxit outer iterations. in this
case the latest
c           approximation p(maxit) is returned in phi
(mud3cr can be recalled
c           with iparm(1)=iparm(17)=1 to improve the
approximation as long
c           as all other arguments are unchanged)
c
c ***      new fatal errors
c
c ... ierror
c
c     = 1 if intl=iparm(1) is not 0 on initial call or
not 0 or 1
c           on subsequent calls of if intl=0 and
iguess=iparm(17)=1
c
c     = 7 if maxcy = iparm(18) is not 1 or 2
c
c     = 8 if method = iparm(19) is less than 0 or
greater than 7
c           mud3cr does not allow planar relaxation.
```

```
meth2=iparm(20)
c           is not used or checked.
c
c       =11 if tolmax = fparm(7) is not 0.0
c
c       =12 if kcycle = mgopt(1) is not 0
c
c       =13 if icros(1) or icros(2) or icros(3) is not 0
or 1
c
c       =14 if tol < 0.0
c
c       =15 if maxit < 1
c
c
*****
*****
```

C

```
*****
*****
```

C

```
*****
*****
```

C

```
end of mud3cr documentation
C
C
*****
*****
```

C

```
*****
*****
```

C

```
*****
*****
```

C

```
*****
*****
```

MUD3SA

```
* * * * * *
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK version 5.0.1
*
C      *
*
C      *               A Fortran Package of Multigrid
*
C      *
*
C      *               Subroutines and Example Programs
*
C      *
*
C      *               for Solving Elliptic Partial Differential
Equations        *
C      *
*
C      *               by
*
C      *
*
C      *               John Adams
*
C      *
*
C      *               of
*
C      *
*
C      *               the National Center for Atmospheric
```

```
Research          *
C      *
*
C      *                      Boulder, Colorado (80307)
U.S.A.          *
C      *
*
C      *                      which is sponsored by
*
C      *
*
C      *                      the National Science Foundation
*
C      *
*
C      *      *      *      *      *      *      *      *      *      *      *
*      *      *      *      *      *      *      *      *      *      *      *
C
C ... file mud3sa.d
C
C     contains documentation for:
C     subroutine
mud3sa(iparm,fparm,work,sgx,sgy,sgz,xlmbda,bndyc,rhs,phi
,
C     +                  mgopt,ierror)
C     A sample fortran driver is file "tmud3sa.f".
C
C ... required MUDPACK files
C
C     mudcom.f, mud3ln.f, mud3pn.f
C
C ... purpose
C
C     subroutine mud3sa automatically discretizes and
attempts to compute
C     the second order conservative finite difference
approximation
C     to a three dimensional linear nonseparable "self
adjoint" elliptic
C     partial differential equation on a rectangle. the
approximation
C     is generated on a uniform grid covering the
rectangle. boundary
C     conditions may be specified (Dirichlet), periodic,
```

```

or mixed.

c      the form of the pde solved is:
c
c      d(sgx(x,y,z)*dp/dx)/dx +
d(sgy(x,y,z)*dp/dy)/dy +
c
c      d(sgz(x,y,z)*dp/dz)/dz -
xlmbda(x,y,z)*p(x,y,z) = r(x,y,z)
c
c      where sgx(x,y,z),sgy(x,y,z),sgz(x,y,z) (all
positive), xlmbda(x,y,z)
c      (nonnegative), r(x,y,z) (the given right hand
side) and p(x,y,z) (the
c      unknown solution function) are all real valued
functions of the real
c      independent variables x,y,z. the use of the
variable names "x,y,z"
c      does not imply the cartesian coordinate system
underlies the pde.
c      for example, any pde in divergence form in
cartesian coordinates can
c      be put in a self-adjoint form suitable for mud3sa
after a curvilinear
c      coordinate transform (see tmud3sa.f)
c
c
c ... mesh description . . .
c
c      the approximation is generated on a uniform nx by
ny by nz grid.
c      the grid is superimposed on the rectangular
solution region
c
c      [xa,xb] x [yc,yd] x [ze,zf].
c
c      let
c
c      dlx = (xb-xa) / (nx-1), dly = (yd-yc) / (ny-1),
dlz = (zf-ze) / (nz-1)
c
c      be the uniform grid increments in the x,y,z
directions. then
c
c      xi=xa+(i-1)*dlx,   yj=yc+(j-1)*dly, zk =

```

```
ze+ (k-1) *dlz
c
c      for i=1,...,nx; j=1,...,ny; k=1,...,nz  denote the
x,y,z uniform
c      mesh points.
c
c
c ... language
c
c      fortran90/fortran77
c
c
c ... portability
c
c      mudpack5.0.1 software has been compiled and tested
with Fortran77
c      and Fortran90 on a variety of platforms.
c
c ... methods
c
c      details of the methods employed by the solvers in
mudpack are given
c      in [1,9]. [1,2,9] contain performance
measurements on a variety of
c      elliptic pdes (see "references" in the file
"readme"). in summary:
c
c *** discretization and solution (second-order solvers)
(see [1])
c
c      the pde and boundary conditions are automatically
discretized at all
c      grid levels using second-order finite difference
formula. diagonal
c      dominance at coarser grid levels is maintained in
the presence of
c      nonzero first-order terms by adjusting the second-
order coefficient
c      when necessary. the resulting block tri-diagonal
linear system is
c      approximated using multigrid iteration
[10,11,13,15,16,18]. version
c      5.0.1 of mudpack uses only fully weighted residual
restriction. defaults
```

```
c      include cubic prolongation and w(2,1) cycles.  
these can be overridden  
c      with selected multigrid options (see "mgopt").  
error control based on  
c      maximum relative differences is available. full  
multigrid cycling (fmg)  
c      or cycling beginning or restarting at the finest  
grid level can be  
c      selected. a menu of relaxation methods including  
gauss-seidel point,  
c      line relaxation(s) (in any combination of  
directions) and planar  
c      relaxation (for three-dimensional anisotropic  
problems) are provided.  
c      all methods use ordering based on alternating  
points (red/black),  
c      lines, or planes for cray vectorization and  
improved convergence  
c      rates [14].  
c  
c *** higher order solution (fourth-order solvers) (see  
[9,19,21])  
c  
c      if the multigrid cycling results in a second-order  
estimate (i.e.,  
c      discretization level error is reached) then this  
can be improved to a  
c      fourth-order estimate using the technique of  
"deferred corrections."  
c      the values in the solution array are used to  
generate a fourth-order  
c      approximation to the truncation error. second-  
order finite difference  
c      formula are used to approximate third and fourth  
partial derivatives  
c      of the solution function [3]. the truncation  
error estimate is  
c      transferred down to all grid levels using weighted  
averaging where  
c      it serves as a new right hand side. the default  
multigrid options  
c      are used to compute the fourth-order correction  
term which is added  
c      to the original solution array.
```

```
C
C
C ... references (partial)
C
C
C      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
C      Solution of Linear Elliptic Partial Differential
Equations,"
C      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
C
C      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
C      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
C      1989, pp.1-12.
C      .
C      .
C      .
C      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
Haidvogel, and V. Pizzo,
C      "Applications of Multigrid Software in the
Atmospheric Sciences,"
C      Mon. Wea. Rev.,vol. 120 # 7, July 1992, pp. 1447-
1458.
C      .
C      .
C      .
C      [9] J. Adams, "Recent Enhancements in MUDPACK, a
Multigrid Software
C      package for Elliptic Partial Differential
Equations," Applied Math.
C      and Comp., 1991, vol. 43, May 1991, pp. 79-94.
C
C      [10]J. Adams, "MUDPACK-2: Multigrid Software for
Approximating
C      Elliptic Partial Differential Equations on Uniform
Grids with
C      any Resolution," Applied Math. and Comp., 1993,
vol. 53, February
C      1993, pp. 235-249
C      .
C      .
```

```
C .
C
C ... argument description
C
C
C
*****
*****  

C *** input arguments
*****
C
*****
*****  

C
C
C ... iparm
C
C           an integer vector of length 23 used to
efficiently pass
C           integer parameters. iparm is set internally
in mud3sa
C           and defined as follows . . .
C
C
C ... intl=iparm(1)
C
C           an initialization argument. intl=0 must be
input
C           on an initial call. in this case input
arguments will
C           be checked for errors and the elliptic
partial differential
C           equation and boundary conditions will be
discretized using
C           second order finite difference formula.
C
C ***      An approximation is NOT generated after an
intl=0 call!
C           mud3sa should be called with intl=1 to
approximate the elliptic
C           PDE discretized by the intl=0 call. intl=1
should also
C           be input if mud3sa has been called earlier
and only the
```

```
c           values in in rhs (see below) or gbdy (see
bndyc below)
c           or phi (see below) have changed. This will
bypass
c           redundant pde discretization and argument
checking
c           and save computational time. Some examples
of when
c           intl=1 calls should be used are:
c
c           (0) after a intl=0 argument checking and
discretization call
c
c           (1) mud3sa is being recalled for additional
accuracy. In
c               this case iguess=iparm(12)=1 should also
be used.
c
c           (2) mud3sa is being called every time step in
a time dependent
c               problem (see discussion below) where the
elliptic operator
c               does not depend on time.
c
c           (3) mud3sa is being used to solve the same
elliptic equation
c               for several different right hand sides
(igueess=0 should
c               probably be used for each new righthand
side).
c
c           intl = 0 must be input before calling with
intl = 1 when any
c               of the following conditions hold:
c
c           (a) the initial call to mud3sa
c
c           (b) any of the integer arguments other than
igueess=iparm(12)
c               or maxcy=iparm(13) or mgopt have changed
since the previous
c               call.
c
c           (c) any of the floating point arguments other
```

```
than tolmax=
c                      fparm(5) have changed since the previous
call
c
c                      (d) any of the coefficients input by coef
(see below) have
c                      changed since the previous call
c
c                      (e) any of the "alfa" coefficients input by
bndyc (see below)
c                      have changed since the previous call.
c
c                      If any of (a) through (e) are true then the
elliptic PDE
c                      must be discretized or rediscretized.  If
none of (a)
c                      through (e) holds, calls can be made with
intl=1.
c                      Incorrect calls with intl=1 will produce
erroneous results.
c ***      The values set in the saved work space "work"
(see below) with
c                      an intl=0 call must be preserved with
subsequent intl=1 calls.
c
c                      MUDPACK software performance should be
monitored for intl=1
c                      calls.  The intl=0 discretization call
performance depends
c                      primarily on the efficiency or lack of
efficiency of the
c                      user provided subroutines for pde
coefficients and
c                      boundary conditions.

c
c
c ... nxa=iparm(2)
c
c                      flags boundary conditions on the (y,z) plane
x=xa
c
c                      = 0 if p(x,y,z) is periodic in x on [xa,xb]
c                      (i.e., p(x+xb-xa,y,z) = p(x,y,z) for all
x,y,z)
```

```

C
C           = 1 if p(xa,y,z) is specified (this must be
input thru phi(1,j,k))
C
C           = 2 if there are mixed derivative boundary
conditions at x=xa
C           (see "bndyc" description below where kbdy =
1)
C
C
C ... nxb=iparm(3)
C
C           flags boundary conditions on the (y,z) plane
x=xb
C
C           = 0 if p(x,y,z) is periodic in x on [xa,xb]
C           (i.e., p(x+xb-xa,y,z) = p(x,y,z) for all
x,y,z)
C
C           = 1 if p(xb,y,z) is specified (this must be
input thru phi(nx,j,k))
C
C           = 2 if there are mixed derivative boundary
conditions at x=xb
C           (see "bndyc" description below where kbdy =
2)
C
C
C ... nyc=iparm(4)
C
C           flags boundary conditions on the (x,z) plane
y=yc
C
C           = 0 if p(x,y,z) is periodic in y on [yc,yd]
C           (i.e., p(x,y+yd-yc,z) = p(x,y,z) for all
x,y,z)
C
C           = 1 if p(x,yc,z) is specified (this must be
input thru phi(i,1,k))
C
C           = 2 if there are mixed derivative boundary
conditions at y=yc
C           (see "bndyc" description below where kbdy =
3)

```

```

C
C
C ... nyd=iparm(5)
C
C           flags boundary conditions on the (x,z) plane
y=yd
C
C           = 0 if p(x,y,z) is periodic in y on [yc,yd]
C           (i.e., p(x,y+yd-yc,z) = p(x,y,z) for all
x,y,z)

C           = 1 if p(x,yd,z) is specified (this must be
input thru phi(i,ny,k))

C           = 2 if there are mixed derivative boundary
conditions at y=yd
C           (see "bndyc" description below where kbdy =
4)
C
C
C ... nze=iparm(6)
C
C           flags boundary conditions on the (x,y) plane
z=ze
C
C           = 0 if p(x,y,z) is periodic in z on [ze,zf]
C           (i.e., p(x,y,z+zf-ze) = p(x,y,z) for all
x,y,z

C           = 1 if p(x,y,ze) is specified (this must be
input thru phi(i,j,1))

C           = 2 if there are mixed derivative boundary
conditions at z=ze
C           (see "bndyc" description below where kbdy =
5)
C
C
C ... nzf=iparm(7)
C
C           flags boundary conditions on the (x,y) plane
z=zf
C
C           = 0 if p(x,y,z) is periodic in z on [ze,zf]

```

```

c           (i.e., p(x,y,z+zf-ze) = p(x,y,z) for all
x,y,z

c           = 1 if p(x,y,zf) is specified (this must be
input thru phi(i,j,nz))

c           = 2 if there are mixed derivative boundary
conditions at z=zf
c           (see "bndyc" description below where kbdy =
6)

c
c
c *** grid size parameters
c
c
c ... ixp = iparm(8)
c
c           an integer greater than one which is used in
defining the number
c           of grid points in the x direction (see nx =
iparm(14)). "ixp+1"
c           is the number of points on the coarsest x
grid visited during
c           multigrid cycling. ixp should be chosen as
small as possible.
c           recommended values are the small primes 2 or
3 or (possibly) 5.
c           larger values can reduce multigrid
convergence rates considerably,
c           especially if line relaxation in the x
direction is not used.
c           if ixp > 2 then it should be 2 or a small odd
value since a power
c           of 2 factor of ixp can be removed by
increasing iex = iparm(11)
c           without changing nx = iparm(14)

c
c
c ... jyq = iparm(9)
c
c           an integer greater than one which is used in
defining the number
c           of grid points in the y direction (see ny =
iparm(15)). "jyq+1"

```

```
c           is the number of points on the coarsest y
grid visited during
c           multigrid cycling.  jyq should be chosen as
small as possible.
c           recommended values are the small primes 2 or
3 or (possibly) 5.
c           larger values can reduce multigrid
convergence rates considerably,
c           especially if line relaxation in the y
direction is not used.
c           if jyq > 2 then it should be 2 or a small odd
value since a power
c           of 2 factor of jyq can be removed by
increasing jey = iparm(12)
c           without changing ny = iparm(15)
c
c
c ... kzs = iparm(10)
c
c           an integer greater than one which is used in
defining the number
c           of grid points in the z direction (see nz =
iparm(16)). "kzs+1"
c           is the number of points on the coarsest z
grid visited during
c           multigrid cycling.  kzs should be chosen as
small as possible.
c           recommended values are the small primes 2 or
3 or (possibly) 5.
c           larger values can reduce multigrid
convergence rates considerably,
c           especially if line relaxation in the z
direction is not used.
c           if kzs > 2 then it should be 2 or a small odd
value since a power
c           of 2 factor of kzs can be removed by
increasing kez = iparm(13)
c           without changing nz = iparm(16)
c
c
c ... iex = iparm(11)
c
c           a positive integer exponent of 2 used in
defining the number
```

```
c           of grid points in the x direction (see nx =
iparm(14)).  
c           iex .le. 50 is required. for efficient  
multigrid cycling,  
c           iex should be chosen as large as possible and  
ixp=iparm(8)  
c           as small as possible within grid size  
constraints when  
c           defining nx = iparm(14).  
c  
c  
c ... jey = iparm(12)  
c  
c           a positive integer exponent of 2 used in  
defining the number  
c           of grid points in the y direction (see ny =
iparm(15)).  
c           jey .le. 50 is required. for efficient  
multigrid cycling,  
c           jey should be chosen as large as possible and  
jyq=iparm(9)  
c           as small as possible within grid size  
constraints when  
c           defining ny = iparm(15).  
c  
c  
c ... kez = iparm(13)  
c  
c           a positive integer exponent of 2 used in  
defining the number  
c           of grid points in the z direction (see nz =
iparm(16)).  
c           kez .le. 50 is required. for efficient  
multigrid cycling,  
c           kez should be chosen as large as possible and  
kzr=iparm(10)  
c           as small as possible within grid size  
constraints when  
c           defining nz = iparm(16).  
c  
c  
c ... nx = iparm(14)  
c  
c           the number of equally spaced grid points in
```

```

the interval [xa,xb]
c           (including the boundaries).  nx must have the
form
c
c           nx = ixp*(2** (iex-1)) + 1
c
c           where ixp = iparm(8), iex = iparm(11).
c
c
c ... ny = iparm(15)
c
c           the number of equally spaced grid points in
the interval [yc,yd]
c           (including the boundaries).  ny must have the
form:
c
c           ny = jyq*(2** (jey-1)) + 1
c
c           where jyq = iparm(9), jey = iparm(12).
c
c
c ... nz = iparm(16)
c
c           the number of equally spaced grid points in
the interval [ze,zf]
c           (including the boundaries).  nz must have the
form
c
c           nz = kzs*(2** (kez-1)) + 1
c
c           where kzs = iparm(10), kez = iparm(13)
c
c
c *** example
c
c           suppose a solution is wanted on a 33 by 65 by
97 grid.  then
c           ixp=2, jyq=4, kzs=6 and iex=jey=kez=5 could be
used.  a better
c           choice would be ixp=jyq=2, kzs=3, and iex=5,
jey=kez=6.
c
c *** note
c
```

```

c      let G be the nx by ny by nz fine grid on which the
approximation is
c      generated and let n = max0(iex,jey,kez).  in
mudpack, multigrid
c      cycling is implemented on the ascending chain of
grids
c
c          G(1) < ... < G(k) < ... < G(n) = G.
c
c      each g(k) (k=1,...,n) has mx(k) by my(k) by mz(k)
grid points
c      given by:
c
c          mx(k) = ixp*[2** (max0(iex+k-n,1)-1)] + 1
c
c          my(k) = jyq*[2** (max0(jey+k-n,1)-1)] + 1
c
c          mz(k) = kxr*[2** (max0(kez+k-n,1)-1)] + 1
c
c
c
c ... iguess=iparm(17)
c
c          = 0 if no initial guess to the pde is
provided
c          and/or full multigrid cycling beginning
at the
c          coarsest grid level is desired.
c
c          = 1 if an initial guess to the pde at the
finest grid
c          level is provided in phi (see below).  in
this case
c          cycling beginning or restarting at the
finest grid
c          is initiated.
c
c *** comments on iguess = 0 or 1 . . .
c
c
c      setting iguess=0 forces full multigrid or "fmg"
cycling.  phi
c      must be initialized at all grid points.  it can be
set to zero at

```

```

c      non-Dirchlet grid points if nothing better is
available.  the
c      values set in phi when iguess = 0 are passed and
down and serve
c      as an initial guess to the pde at the coarsest
grid level where
c      multigrid cycling commences.
c
c      if iguess = 1 then the values input in phi are an
initial guess to the
c      pde at the finest grid level where cycling begins.
this option should
c      be used only if a "very good" initial guess is
available (as, for
c      example, when restarting from a previous iguess=0
call).
c
c *** time dependent problems . . .
c
c      assume we are solving an elliptic pde every time
step in a
c      marching problem of the form:
c
c          l(p(t)) = r(t)
c
c      where the differential operator "l" has no time
dependence,
c      "p(t)" is the solution and "r(t)" is the right
hand side at
c      current time "t". let "dt" be the increment
between time steps.
c      then p(t) can be used as an initial guess to
p(t+dt) with
c      intl = 1 when solving
c
c          l(p(t+dt)) = r(t+dt) .
c
c      after the first two time steps, rather than
continue, it would
c      be better to define the "correction" term:
c
c          e(t,dt) = p(t+dt) - p(t)
c
c      this clearly satisfies the equation

```

```

c
c           l(e(t,dt)) = r(t+dt) - r(t).
c
c   this should be solved with iguess = 0 and intl =
1. boundary
c   conditions for e(t,dt) are obtained from the
boundary conditions
c   for p(t) by subtracting given values at t from
given values at
c   t+dt. for example if
c
c           d(p(t))/dx = f(t), d(p(t+dt))/dx = f(t+dt)
c
c   at some x boundary then e(t,dt) satisfies the
derivative
c   boundary condition
c
c           d(e(t,dt))/dx = f(t+dt) - f(t).
c
c   e(t,dt) can be preset to 0.0 (at non-Dirchlet
points) or (if p(t-dt)
c   is saved) to p(t)-p(t-dt). with iguess = 0, these
values will serve
c   as an initial guess to e(t,dt) at the coarsest
grid level. this
c   approach has the advantage that a full sequence of
multigrid cycles,
c   beginning at the coarsest grid level, is invoked
every time step in
c   solving for e(t,dt). a few digits of accuracy in
e(t,dt), which is
c   ordinarily much smaller than p(t), will yield
several more digits of
c   accuracy in the final approximation:
c
c           p(t+dt) = p(t) + e(t,dt).
c
c   using this approach to integrate in time will give
more accuracy
c   then using p(t) as an initial guess to p(t+dt) for
all time steps.
c   it does require additional storage.
c
c   if the differential operator "l" has time

```

```

dependence (either thru
c      the coefficients in the pde or the coefficients in
the derivative
c      boundary conditions) then use p(t) as an initial
guess to p(t+dt)
c      when solving
c
c          l(t+dt) (p(t+dt)) = r(t+dt)
c
c      with intl = 0 for all time steps (the
discretization must be repeated
c      for each new "t"). either iguess = 0 (p(t) will
then be an initial
c      guess at the coarsest grid level where cycles will
commence) or
c      iguess = 1 (p(t) will then be an initial guess at
the finest grid
c      level where cycles will remain fixed) can be
tried.
c
c
c ... maxcy = iparm(18)
c
c      the exact number of cycles executed between
the finest
c      (nx by ny by nz) and the coarsest ((ixp+1) by
(jyq+1) by
c      (kzr+1)) grid levels when tolmax=fparm(7)=0.0
(no error
c      control). when tolmax=fparm(7).gt.0.0 is
input (error control)
c      then maxcy is a limit on the number of cycles
between the
c      finest and coarsest grid levels. in any
case, at most
c      maxcy*(iprert+ipost) relaxation sweeps are
performed at the
c      finest grid level (see
iprert=mgopt(2),ipost=mgopt(3) below)
c      when multigrid iteration is working
"correctly" only a few
c      cycles are required for convergence. large
values for maxcy
c      should not be required.

```

```
c
c
c ... method = iparm(19)
c
c           this sets the method of relaxation (all
relaxation
c           schemes in mudpack use red/black type
ordering)
c
c           = 0 for gauss-seidel pointwise relaxation
c
c           = 1 for line relaxation in the x direction
c
c           = 2 for line relaxation in the y direction
c
c           = 3 for line relaxation in the z direction
c
c           = 4 for line relaxation in the x and y
direction
c
c           = 5 for line relaxation in the x and z
direction
c
c           = 6 for line relaxation in the y and z
direction
c
c           = 7 for line relaxation in the x,y and z
direction
c
c           = 8 for x,y planar relaxation
c
c           = 9 for x,z planar relaxation
c
c           =10 for y,z planar relaxation
c
c *** if nxa = 0 and nx = 3 at a grid level where line
relaxation in the x
c       direction is flagged then it will be replaced by
gauss-seidel point
c       relaxation at that grid level.
c
c *** if nyc = 0 and ny = 3 at a grid level where line
relaxation in the y
c       direction is flagged then it will be replaced by
```

```
gauss-seidel point
c      relaxation at that grid level.
c
c *** if nze = 0 and nz = 3 at a grid level where line
relaxation in the z
c      direction is flagged then it will be replaced by
gauss-seidel point
c      relaxation at that grid level.
c
c      these adjustments are necessary since the
simultaneous tri-diagonal
c      solvers used with line periodic relaxation must
have n > 2 where n
c      is number of unknowns (excluding the periodic
point).

c *** choice of method
c
c      this is very important for efficient convergence.
in some cases
c      experimentation may be required.
c
c      let fx represent the quantity sgx(x,y,z)/dlx**2
over the solution box
c
c      let fy represent the quantity sgy(x,y,z)/dly**2
over the solution box
c
c      let fz represent the quantity sgz(x,y,z)/dlz**2
over the solution box
c
c      (0) if fx,fy,fz are roughly the same size and do
not vary too
c          much choose method = 0.  if this fails try
method = 7.
c
c      (1) if fx is much greater then fy,fz and fy,fz
are roughly the same
c          size choose method = 1
c
c      (2) if fy is much greater then fx,fz and fx,fz
are roughly the same
c          size choose method = 2
c
```

```

c      (3) if fz is much greater then fx,fy and fx,fy
are roughly the same
c          size choose method = 3
c
c      (4) if fx,fy are roughly the same and both are
much greater then fz
c          try method = 4.  if this fails try method = 8
c
c      (5) if fx,fz are roughly the same and both are
much greater then fy
c          try method = 5.  if this fails try method = 9
c
c      (6) if fy,fz are roughly the same and both are
much greater then fx
c          try method = 6.  if this fails try method =
10
c
c      (7) if fx,fy,fz vary considerably with none
dominating try method = 7
c
c      (8) if fx and fy are considerably greater than fz
but not necessarily
c          the same size (e.g., fx=1000.,fy=100.,fz=1.)
try method = 8
c
c      (9) if fx and fz are considerably greater than fy
but not necessarily
c          the same size (e.g., fx=10.,fy=1.,fz=1000.)
try method = 9
c
c      (10)if fy and fz are considerably greater than fx
but not necessarily
c          the same size (e.g., fx=1.,fy=100.,fz=10.)
try method = 10
c
c
c ... meth2 = iparm(20) determines the method of
relaxation used in the planes
c          when method = 8 or 9 or 10.
c
c
c      (if method = 8)
c
c          = 0 for gauss-seidel pointwise relaxation

```

```

c           in the x,y plane for each fixed z
c           = 1 for line relaxation in the x direction
c           in the x,y plane for each fixed z
c           = 2 for line relaxation in the y direction
c           in the x,y plane for each fixed z
c           = 3 for line relaxation in the x and y
direction
c           in the x,y plane for each fixed z
c
c           (1) if fx,fy are roughly the same and vary
little choose meth2 = 0
c           (2) if fx is much greater than fy choose
meth2 = 1
c           (3) if fy is much greater than fx choose
meth2 = 2
c           (4) if none of the above or meth2 = 0 fails
choose meth2 = 3
c
c           (if method = 9)
c
c           = 0 for gauss-seidel pointwise relaxation
with red/black ordering
c           in the x,z plane for each fixed y
c           = 1 for simultaneous line relaxation in the x
direction
c           of the x,z plane for each fixed y
c           = 2 for simultaneous line relaxation in the z
direction
c           of the x,z plane for each fixed y
c           = 3 for simultaneous line relaxation in the x
and z direction
c           of the x,z plane for each fixed y
c
c           (1) if fx,fz are roughly the same and vary
little choose meth2 = 0
c           (2) if fx is much greater than fz choose
meth2 = 1
c           (3) if fz is much greater than fx choose
meth2 = 2
c           (4) if none of the above or meth2 = 0 fails
choose meth2 = 3
c
c           (if method = 10)
c

```

```

c           = 0 for gauss-seidel pointwise relaxation
with red/black ordering
c           in the y,z plane for each fixed x
c           = 1 for simultaneous line relaxation in the y
direction
c           of the y,z plane for each fixed x
c           = 2 for simultaneous line relaxation in the z
direction
c           of the y,z plane for each fixed x
c           = 3 for simultaneous line relaxation in the y
and z direction
c           of the y,z plane for each fixed x
c
c           (1) if fy,fz are roughly the same and vary
little choose meth2 = 0
c           (2) if fy is much greater than fz choose
meth2 = 1
c           (3) if fz is much greater than fy choose
meth2 = 2
c           (4) if none of the above or meth2 = 0 fails
choose meth2 = 3
c
c
c ... length = iparm(21)
c
c           the length of the work space provided in
vector work.
c
c           let isx = 3 if method = 1,4,5 or 7 and
nxa.ne.0
c           let isx = 5 if method = 1,4,5 or 7 and
nxa.eq.0
c           let isx = 0 if method has any other value
c
c           let jsy = 3 if method = 2,4,6 or 7 and
nyc.ne.0
c           let jsy = 5 if method = 2,4,6 or 7 and
nyc.eq.0
c           let jsy = 0 if method has any other value
c
c           let ksz = 3 if method = 3,5,6 or 7 and
nze.ne.0
c           let ksz = 5 if method = 3,5,6 or 7 and
nze.eq.0

```

```
c           let ksz = 0 if method has any other value
c
c
c           then (for method .le.7)
c
c               (1)    length =
c           (nx+2)*(ny+2)*(nz+2)*(10+isx+jsy+ksz)
c
c               or (for method.gt.7)
c
c               (2)    length = 14*(nx+2)*(ny+2)*(nz+2)
c
c           will usually but not always suffice.  The
c exact minimal length depends,
c           in a complex way, on the grid size arguments
c and method chosen.
c ***      It can be predetermined for the current input
c arguments by calling
c           mud3sa with length=iparm(21)=0 and printing
c iparm(22) or (in f90)
c           dynamically allocating the work space using
c the value in iparm(22)
c           in a subsequent mud3sa call.
c
c
c ... fparm
c
c           a floating point vector of length 8 used to
c efficiently
c           pass floating point parameters.  fparm is set
c internally
c           in mud3sa and defined as follows . . .
c
c
c ... xa=fparm(1), xb=fparm(2)
c
c           the range of the x independent variable. xa
c must
c           be less than xb
c
c
c ... yc=fparm(3), yd=fparm(4)
c
c           the range of the y independent variable. yc
```

```
must
c           be less than yd.
c
c
c ... ze=fparm(5), zf=fparm(6)
c
c           the range of the z independent variable. ze
must
c           be less than zf.
c
c
c ... tolmax = fparm(7)
c
c           when input positive, tolmax is a maximum
relative error tolerance
c           used to terminate the relaxation iterations.
assume phil(i,j,k)
c           and phi2(i,j,k) are the last two computed
approximations at all
c           grid points of the finest grid level. if we
define
c
c           phdif = max(abs(phi2(i,j,k)-phil(i,j,k)))
for all i,j,k
c
c           and
c
c           phmax = max(abs(phi2(i,j,k))) for all
i,j,k
c
c           then "convergence" is considered to have
occurred if and only if
c
c           phdif/phmax < tolmax.
c
c
c           if tolmax=fparm(7)=0.0 is input then there is
no error control
c           and maxcy cycles from the finest grid level
are executed. maxcy
c           is a limit which cannot be exceeded even with
error control.
c   *** calls with tolmax=0.0, when appropriate
because of known
```

```

c           convergence behavior, are more efficient than
calls with tolmax
c           positive (i.e., if possible DO NOT use error
control!).
c
c
c ... work
c
c           a one dimensional array that must be provided
for work space.
c           see length = iparm(21). the values in work
must be preserved
c           if mud3sa is called again with
intl=iparm(1).ne.0
c
c
c ... bndyc
c
c           a subroutine with parameters
(kbdy,xory,yorz,alfa,gbdy).
c           which are used to input mixed boundary
conditions to mud3sa.
c           the boundaries are numbered one thru six and
the form of
c           conditions are described below.
c
c
c           (1) the kbdy=1 boundary
c
c           this is the (y,z) plane x=xa where nxz=iparm(2) =
2 flags
c           a mixed boundary condition of the form
c
c           dp/dx + alfxz(y,z)*p(xa,y,z) = gbdxz(y,z)
c
c           in this case kbdy=1,xory=y,yorz=z will be input to
bndyc and
c           alfa,gbdy corresponding to alfxz(y,z),gbdxz(y,z)
must be returned.
c
c
c           (2) the kbdy=2 boundary
c
c           this is the (y,z) plane x=xb where nxz=iparm(3) =

```

```

2 flags
c      a mixed boundary condition of the form
c
c      dp/dx + alfxb(y,z)*p(xb,y,z) = gbdxb(y,z)
c
c      in this case kbdy=2,xory=y,yorz=z will be input to
bndyc and
c      alfa,gbdy corresponding to alfxb(y,z),gbdxb(y,z)
must be returned.
c
c
c      (3) the kbdy=3 boundary
c
c      this is the (x,z) plane y=yc where nyc=iparm(4) =
2 flags
c      a mixed boundary condition of the form
c
c      dp/dy + alfyc(x,z)*p(x,yc,z) = gbdyc(x,z)
c
c      in this case kbdy=3,xory=x,yorz=z will be input to
bndyc and
c      alfa,gbdy corresponding to alfyc(x,z),gbdyc(x,z)
must be returned.
c
c
c      (4) the kbdy=4 boundary
c
c      this is the (x,z) plane y=yd where nyd=iparm(5) =
2 flags
c      a mixed boundary condition of the form
c
c      dp/dy + alfyd(x,z)*p(x,yd,z) = gbdyd(x,z)
c
c      in this case kbdy=4,xory=x,yorz=z will be input to
bndyc and
c      alfa,gbdy corresponding to alfyd(x,z),gbdyd(x,z)
must be returned.
c
c
c      (5) the kbdy=5 boundary
c
c      this is the (x,y) plane z=ze where nze=iparm(6) =
2 flags
c      a mixed boundary condition of the form

```

```

C
C      dp/dz + alfze(x,y)*p(x,y,ze) = gbdze(x,y)
C
C      in this case kbdy=5,xory=x,yorz=y will be input to
C      bndyc and
C      alfa,gbdy corresponding to alfze(x,y),gbdze(x,y)
C      must be returned.
C
C
C      (6) the kbdy=6 boundary
C
C      this is the (x,y) plane z=zf where nzf=iparm(7) =
C      2 flags
C      a mixed boundary condition of the form
C
C      dp/dz + alfzf(x,y)*p(x,y,zf) = gbdzf(x,y)
C
C      in this case kbdy=6,xory=x,yorz=y will be input to
C      bndyc and
C      alfa,gbdy corresponding to alfzf(x,y),gbdzf(x,y)
C      must be returned.
C
C
C      *** alfxa,alfyc,alfze nonpositive and
C      alfxb,alfyd,alfze nonnegative
C      will help maintain matrix diagonal dominance
C      during discretization
C      aiding convergence.
C
C
C      *** bndyc must provide the mixed boundary condition
C      values in correspondence with those flagged in
C      iparm(2)
C      thru iparm(7). if all boundaries are specified
C      then
C      mud3sa will never call bndyc. even then it must
C      be entered
C      as a dummy subroutine. bndyc must be declared
C      external in the routine calling mud3sa. the
C      actual
C      name chosen may be different.
C
C
C      ... sgx,sgy,sgz
C

```

```

c      function subroutines which returns the real
value of the
c      coefficients at any point (x,y,z). they must
be constructed
c      to return values outside the solution region.
let dx=(xb-xa)/ixp,
c      dy = (yd-yc)/jyq, dz=(zf-ze)/kzr (coarse grid
increments).
c      then sgx,sgy,sgz will be invoked for x,y,z in
the intervals
c
c      [xa-0.5*dx,xa], [xb,xb+0.5*dx]
c      [yc-0.5*dy,yc], [yd,yd+0.5*dy]
c      [ze-0.5*dz,ze], [zf,zf+0.5*dz].
c
c      this is necessitated by conservative finite
differencing. values
c      outside specified (Dirichlet) boundaries will
not be used in
c      the final discretization (e.g., if nxa = 1
then sgx will be
c      invoked for x.lt.xa but these values will not
used). on the other-
c      hand such values are required outside mixed
derivative boundaries.
c      sgx,sgy,sgz should be positive for all (x,y,z)
(see ierror = -5). they
c      must be declared "external" in the user
constructed program calling
c      mud3sa where their names may be different.
c
c ... xlmbda
c
c      a real valued function subroutine which
returns the value of
c      "xlmbda" in the pde at any grid point
(xi,yj,zk). xlmbda should
c      be nonnegative for any (xi,yj,zk) (see ierror
= -4). xlmbda must be
c      declared "external" in the user constructed
program calling
c      mud3sa where its name may be different.
c
c

```

```

c ... rhs
c
c           an array dimensioned nx by ny by nz which
contains
c           the given right hand side values on the
uniform 3-d mesh.
c           rhs(i,j,k) = r(xi,yj,zk) for i=1,...,nx and
j=1,...,ny
c           and k=1,...,nz.
c
c ... phi
c
c           an array dimensioned nx by ny by nz .  on
input phi must
c           contain specified boundary values and an
initial guess
c           to the solution if flagged (see
iguess=iparm(17)=1).  for
c           example, if nyd=iparm(5)=1 then phi(i,ny,k)
must be set
c           equal to p(xi,yd,zk) for i=1,...,nx and
k=1,...,nz prior to
c           calling mud3sa.  the specified values are
preserved by mud3sa.
c
c ***      if no initial guess is given (iguess=0) then
phi must still
c           be initialized at non-Dirchlet grid points
(this is not
c           checked).  these values are projected down and
serve as an initial
c           guess to the pde at the coarsest grid level.
set phi to 0.0 at
c           nonDirchlet grid points if nothing better is
available.
c
c
c ... mgopt
c
c           an integer vector of length 4 which allows
the user to select
c           among various multigrid options.  if
mgopt(1)=0 is input then
c           a default set of multigrid parameters

```

```
(chosen for robustness)
c           will be internally selected and the
remaining values in mgopt
c           will be ignored. if mgopt(1) is nonzero
then the parameters
c           in mgopt are set internally and defined as
follows: (see the
c           basic coarse grid correction algorithm
below)

c
c
c      kcycle = mgopt(1)
c
c      = 0 if default multigrid options are to be
used
c
c      = 1 if v cycling is to be used (the least
expensive per cycle)
c
c      = 2 if w cycling is to be used (the
default)
c
c      > 2 if more general k cycling is to be used
c      *** warning--values larger than 2 increase
c          the execution time per cycle
considerably and
c          result in the nonfatal error ierror =
-5
c          which indicates inefficient multigrid
cycling.

c
c      iprer = mgopt(2)
c
c          the number of "pre-relaxation" sweeps
executed before the
c          residual is restricted and cycling is
invoked at the next
c          coarser grid level (default value is 2
whenever mgopt(1)=0)
c
c      ipost = mgopt(3)
c
c          the number of "post relaxation" sweeps
executed after cycling
```

```
c           has been invoked at the next coarser grid
level and the residual
c           correction has been transferred back
(default value is 1
c           whenever mgopt(1)=0) .
c
c *** if iprer, ipost, or (especially) kcycle is greater
than 2
c       than inefficient multigrid cycling has probably
been chosen and
c       the nonfatal error (see below) ierror = -5 will be
set. note
c       this warning may be overridden by any other
nonzero value
c       for ierror.
c
c   intpol = mgopt(4)
c
c       = 1 if multilinear prolongation
(interpolation) is used to
c       transfer residual corrections and the pde
approximation
c       from coarse to fine grids within full
multigrid cycling.
c
c       = 3 if multicubic prolongation
(interpolation) is used to
c       transfer residual corrections and the pde
approximation
c       from coarse to fine grids within full
multigrid cycling.
c           (this is the default value whenever
mgopt(1)=0) .
c
c *** the default values (2,2,1,3) in the vector mgopt
were chosen for
c       robustness. in some cases v(2,1) cycles with
linear prolongation will
c       give good results with less computation
(especially in two-dimensions).
c       this was the default and only choice in an
earlier version of mudpack
c       (see [1]) and can be set with the integer vector
(1,2,1,1) in mgopt.
```

```

c
c *** the schedules for one full multigrid cycle
(iguess=0) using v(2,1)
c      cycles and w(2,1) cycles are depicted for a four
level grid below.
c      the number of relaxation sweeps when each grid is
visited are indicated.
c      the "*" stands for prolongation of the full
approximation and the "."
c      stands for transfer of residuals and residual
corrections within the
c      coarse grid correction algorithm (see below).
c
c      one fmg with v(2,1) cycles:
c
c
c      -----
c      -----2-----1-----1-----2-----1-----1-
c      -----      level 4
c          *
c          .
c          *
c          .
c      -----2-----1-----2-----1-----2-----1-----1-
c      -----      level 3
c          *
c          .
c          *
c          .
c      -----2-----1-----2-----1-----2-----1-----1-
c      -----      level 2
c          *
c          .
c          *
c          .
c      ---3---3---3-----3-----3-----3-----
c      -----      level 1
c
c
c      one fmg with w(2,1) cycles:
c
c
c      -----
c      -----2-----1-----1-----2-----3-----1-
c      -----      level 4
c          *
c          .
c          .
c      -----2-----1-----2-----3-----1-----2-----3-----1-
c      -----      level 3
c          *
c          .
c          .
c          *
c          .
c          .
c      -----2-----1-----2-----3-----1-----2-----3-----1-----2-----3-----1-
c      -----      level 2
c          *
c          .
c          .
c          .
c          .
c          .
c          .

```

```

c      --6---6-----6---6-----6---6-----6---6---
c      -----      level 1
c
c
c      the form of the "recursive" coarse grid correction
c      cycling used
c      when kcycle.ge.0 is input is described below in
c      pseudo-algorithmic
c      language. it is implemented non-recursively in
c      fortran in mudpack.
c
c      algorithm
cgc(k,l(k),u(k),r(k),kcycle,iprer,ipost,iresw,intpol)
c
c *** approximately solve l(k)*u(k) = r(k) using
c      multigrid iteration
c *** k is the current grid level
c *** l(k) is the discretized pde operator at level k
c *** u(k) is the initial guess at level k
c *** r(k) is the right hand side at level k
c *** i(k,k-1) is the restriction operator from level k
c      to level k-1
c *** (the form of i(k,k-1) depends on iresw)
c *** i(k-1,k) is the prolongation operator from level
c      k-1 to level k
c *** (the form of i(k-1,k) depends on intpol)
c
c      begin algorithm cgc
c
c *** pre-relax at level k
c
c      . do (i=1,iprer)
c
c      . . relax(l(k),u(k),r(k))
c
c      . end do
c
c      . if (k > 1) then
c
c ***      restrict the residual from level k to level k-
c      1
c
c      . . r(k-1) = i(k,k-1) (r(k)-l(k)*u(k))
c

```

```

C      . . kount = 0
C
C      . . repeat
C
C ***      solve for the residual correction at level k-1
in u(k-1)
C ***      using algorithm cgc "kcycle" times (this is
the recursion)
C
C      . . . kount = kount+1
C
C      . . . invoke cgc(k-1,l(k-1),u(k-1),r(k-
1),kcycle,ipr,r,ipost,iressw)
C
C
C      . . until (kount.eq.kcycle)
C
C ***      transfer residual correction in u(k-1) to
level k
C ***      with the prolongation operator and add to u(k)
C
C      . . u(k) = u(k) + i(k-1,k) (u(k-1))
C
C      . end if
C
C ***      post relax at level k
C
C      . do (i=1,ipost)
C
C      . . relax(l(k),u(k),r(k))
C
C      . end do
C
C      . return
C
C      end algorithm cgc
C
C
C ****output
parameters*****
C

```

```
*****
*****
C
C
C ... iparm(22)
C
C           on output iparm(22) contains the actual work
space length
C           required for the current grid sizes and
method.
C
C
C ... iparm(23)
C
C           if error control is selected (tolmax =
fparm(7) .gt. 0.0) then
C           on output iparm(23) contains the actual
number of cycles executed
C           between the coarsest and finest grid levels
in obtaining the
C           approximation in phi.  the quantity
(iprter+ipost)*iparm(23) is
C           the number of relaxation sweeps performed at
the finest grid level.
C
C
C ... fparm(8)
C
C           on output fparm(8) contains the final
computed maximum relative
C           difference between the last two iterates at
the finest grid level.
C           fparm(8) is computed only if there is error
control (tolmax.gt.0.)
C           assume phi1(i,j,k) and phi2(i,j,k) are the
last two computed
C           values for phi(i,j,k) at all points of the
finest grid level.
C           if we define
C
C           phdif = max (abs(phi2(i,j,k)-phi1(i,j,k)))
for all i,j,k
C
C           and
```

```
C
C           phmax = max (abs (phi2 (i,j,k) )) for all
i,j,k
C
C           then
C
C           fparm(8) = phdif/phmax
C
C           is returned whenever phmax.gt.0.0.  in the
degenerate case
C           phmax = 0.0, fparm(8) = phdif is returned.
C
C
C
C ... work
C
C           on output work contains intermediate values
that must not be
C           destroyed if mud3sa is to be called again
with intl = iparm(1)=1
C
C ... phi
C
C           on output phi(i,j,k) contains the
approximation to
C           p(xi,yj,zk) for all mesh points i=1,...,nx;
j=1,...,ny;
C           k=1,...,nz.  the last computed iterate in phi
is returned
C           even if convergence is not obtained (ierror=-1)
C
C ... ierror
C
C           For intl=iparm(1)=0 initialization calls,
ierror is an
C           error flag that indicates invalid input
arguments when
C           returned positive and nonfatal warnings when
returned
C           negative. Argument checking and
discretization
C           is bypassed for intl=1 calls which can only
return
```

```

c         ierror = -1 or 0 or 1.
c
c
c         non-fatal warnings * * *
c
c
c         ==5 if kcycle=mgopt(1) is greater than 2. values
lager than 2 results
c             in an algorithm which probably does far more
computation than
c             necessary.  kcycle = 1 (v cycles)  or kcycle=2
(w cycles) should
c             suffice for most problems.  ierror = -5 is
also set if either
c             iprer = mgopt(2) or ipost=mgopt(3) is greater
than 2.  the
c             ierror=-5 flag is overridden by any other
fatal or non-fatal
c             error.
c
c         ==4 if there are dominant nonzero first order
terms in the pde which
c             make it "hyperbolic" at the finest grid level.
numerically, this
c             happens if:
c
c                 abs(cx)*dlx > 2.*abs(cxx)      (dlx = (xb-
xa) / (nx-1))
c
c                         (or)
c
c                 abs(cy)*dly > 2.*abs(cyy)      (dly = (yd-
yc) / (ny-1))
c
c
c             at some fine grid point (xi,yj).  if an
adjustment is not made the
c             condition can lead to a matrix coming from the
discretization
c             which is not diagonally dominant and
divergence is possible. since
c             the condition is "likely" at coarser grid
levels for pde's with
c             nonzero first order terms, the adjustments

```

```

(actually first order
c           approximations to the pde)
c
c
c           cxx = amax1(cxx,0.5*abs(cx)*dx)
c
c           (and)
c
c           cyy = amax1(cyy,0.5*abs(cy)*dy)
c
c
c           (here dx,dy are the x,y mesh sizes of the
subgrid)
c
c           are made to preserve convergence of multigrid
iteration. if made
c           at the finest grid level, it can lead to
convergence to an
c           erroneous solution (flagged by ierror = -4).
a possible remedy
c           is to increase resolution. the ierror = -4
flag overrides the
c           nonfatal ierror = -5 flag.
c
c
c           ==3 if the continuous elliptic pde is singular.
this means the
c           boundary conditions are periodic or pure
derivative at all
c           boundaries and ce(x,y) = 0.0 for all x,y. a
solution is still
c           attempted but convergence may not occur due
to ill-conditioning
c           of the linear system coming from the
discretization. the
c           ierror = -3 flag overrides the ierror=-4 and
ierror=-5 nonfatal
c           flags.
c
c
c           ==2 if the pde is not elliptic (i.e.,
cxx*cyy.le.0.0 for some (xi,yj))
c           in this case a solution is still attempted
although convergence

```

```

c      may not occur due to ill-conditioning of the
linear system.
c      the ierror = -2 flag overrides the ierror=-
5,-4,-3 nonfatal
c      flags.
c
c
c      ==1 if convergence to the tolerance specified in
tolmax=fparm(5)>0.
c      is not obtained in maxcy=iparm(13) multigrid
cycles between the
c      coarsest (ixp+1,jyq+1) and finest (nx,ny)
grid levels.
c      in this case the last computed iterate is
still returned.
c      the ierror = -1 flag overrides all other
nonfatal flags
c
c
c      no errors * * *
c
c      = 0
c
c      fatal argument errors * * *
c
c      = 1 if intl=iparm(1) is not 0 on initial call or
not 0 or 1
c      on subsequent calls
c
c      = 2 if any of the boundary condition flags
nxa,nxb,nyc,nyd,nze,nzf
c      in iparm(2) through iparm(7) is not 0,1 or 2 or
if
c      (nxa,nxb) or (nyc,nyd) or (nze,nzf) are not
pairwise zero.
c
c      = 3 if min0(ixp,jyq,kzr) < 2
(ixp=iparm(8),jyq=iparm(9),kzr=iparm(10))
c
c      = 4 if min0(iex,jey,kez) < 1
(iex=iparm(11),jey=iparm(12),kez=iparm(13))
c      or if max0(iex,jey,kez) > 50
c
c      = 5 if nx.ne.ixp*2** (iex-1)+1 or if

```

```

ny.ne.jyq*2** (jey-1)+1 or
c           if nz.ne.kzr*2** (kez-1)+1
(nx=iparm(14),ny=iparm(15),nz=iparm(16))
c
c      = 6 if iguess = iparm(17) is not equal to 0 or 1
c
c      = 7 if maxcy = iparm(18) < 1 (large values for
maxcy should not be used)
c
c      = 8 if method = iparm(19) is less than 0 or
greater than 10 or
c          if meth2 = iparm(20) is not 0 or 1 or 2 or 3
when method > 7.
c
c      = 9 if length = iparm(20) is too small (see
iparm(21) on output
c          for minimum required work space length)
c
c      =10 if xa >= xb or yc >= yd or ze >= zf
c
(xa=fparm(1),xb=fparm(2),yc=fparm(3),yd=fparm(4),ze=fpar
m(5),zf=fparm(6))
c
c      =11 if tolmax = fparm(7) < 0.0
c
c      errors in setting multigrid options * * * (see
also ierror=-5)
c
c      =12 if kcycle = mgopt(1) < 0 or
c          if iprer = mgopt(2) < 1 or
c          if ipost = mgopt(3) < 1 or
c          if interpol = mgopt(4) is not 1 or 3
c
c
*****
*
c
*****
*
c
c      end of mud3sa documentation
c
c
*****

```

```
**  
C  
*****  
**  
C  
C
```

---

## MUD3SP

```
C  
C      file mud3sp.d  
C  
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C      *  
*  
C      *           copyright (c) 2008 by UCAR  
*  
C      *  
*  
C      *           University Corporation for Atmospheric  
Research          *  
C      *  
*  
C      *           all rights reserved  
*  
C      *  
*  
C      *           MUDPACK  version 5.0.1  
*  
C      *  
*  
C      *           A Fortran Package of Multigrid  
*  
C      *  
*  
C      *           Subroutines and Example Programs  
*
```

```
C      *
*
C      *      for Solving Elliptic Partial Differential
Equations      *
C      *
*
C      *                      by
*
C      *
*
C      *                      John Adams
*
C      *
*
C      *                      of
*
C      *
*
C      *      the National Center for Atmospheric
Research      *
C      *
*
C      *      Boulder, Colorado (80307)
U.S.A.      *
C      *
*
C      *                      which is sponsored by
*
C      *
*
C      *                      the National Science Foundation
*
C      *
*
C      *      *      *      *      *      *      *      *      *      *      *
*      *      *      *      *      *      *      *      *      *      *      *
*
C
C ... file mud3sp.d
C
C      contains documentation for:
C      subroutine
mud3sp(iparm,fparm,work,cofx,cofy,cofz,bndyc,rhs,phi,
C      +                  mgopt,ierror)
C      A sample fortran driver is file "tmud3sp.f".
```

```

c
c ... required MUDPACK files
c
c      mudcom.f
c
c ... purpose
c
c      subroutine mud3sp automatically discretizes and
attempts to compute
c      the second order finite difference approximation
to a three-
c      dimensional linear SEPARABLE elliptic partial
differential
c      equation on a box.  the approximation is generated
on a uniform
c      grid covering the box (see mesh description
below). boundary
c      conditions may be any combination of mixed,
specified (Dirchlet)
c      or periodic.  the form of the pde solved is . . .
c
c      cxx(x)*pxx + cx(x)*px + cex(x)*p(x,y,z) +
c
c      cyy(y)*pyy + cy(y)*py + cey(y)*p(x,y,z) +
c
c      czz(z)*pzz + cz(z)*pz + cez(z)*p(x,y,z) =
r(x,y,z)
c
c      here cxx,cx,cex,cyy,cy,cey,czz,cz,cez are the
known real coefficients
c      of the pde; pxx,pyy,pzz,px,py,pz are the second
and first
c      partial derivatives of the unknown solution
function p(x,y,z)
c      with respect to the independent variables x,y,z;
r(x,y,z) is
c      is the known real right hand side of the elliptic
pde. cxx,cyy
c      and czz should be positive for all (x,y,z) in the
solution region.
c
c      SEPARABILITY means:
c
c      cxx,cx,cex depend only on x

```

```

c      cyy, cy, cey depend only on y
c      czz, cz, cez depend only on z
c
c      For example, LaPlace's equation in Cartesian
c      coordinates is separable.
c      Nonseparable elliptic PDEs can be approximated
c      with muh3 or mud3.
c      mud3sp requires considerably less work space than
c      muh3 or mud3.
c
c ... mesh description . . .
c
c      the approximation is generated on a uniform nx by
c      ny by nz grid.
c      the grid is superimposed on the rectangular
c      solution region
c
c      [xa, xb] x [yc, yd] x [ze, zf].
c
c      let
c
c      dlx = (xb-xa) / (nx-1), dly = (yd-yc) / (ny-1),
c      dlz = (zf-ze) / (nz-1)
c
c      be the uniform grid increments in the x,y,z
c      directions. then
c
c      xi=xa+(i-1)*dlx,   yj=yc+(j-1)*dly,  zk =
c      ze+(k-1)*dlz
c
c      for i=1,...,nx; j=1,...,ny; k=1,...,nz  denote the
c      x,y,z uniform
c      mesh points.
c
c
c ... language
c
c      fortran90/fortran77
c
c
c ... portability
c
c      mudpack5.0.1 software has been compiled and tested
c      with Fortran77

```

```
c      and Fortran90 on a variety of platforms.  
c  
c ... methods  
c  
c      details of the methods employed by the solvers in  
mudpack are given  
c      in [1,9]. [1,2,9] contain performance  
measurements on a variety of  
c      elliptic pdes (see "references" in the file  
"readme"). in summary:  
c  
c *** discretization and solution (second-order solvers)  
(see [1])  
c  
c      the pde and boundary conditions are automatically  
discretized at all  
c      grid levels using second-order finite difference  
formula. diagonal  
c      dominance at coarser grid levels is maintained in  
the presence of  
c      nonzero first-order terms by adjusting the second-  
order coefficient  
c      when necessary. the resulting block tri-diagonal  
linear system is  
c      approximated using multigrid iteration  
[10,11,13,15,16,18]. version  
c      5.0.1 of mudpack uses only fully weighted residual  
restriction. defaults  
c      include cubic prolongation and w(2,1) cycles.  
these can be overridden  
c      with selected multigrid options (see "mgopt").  
error control based on  
c      maximum relative differences is available. full  
multigrid cycling (fmg)  
c      or cycling beginning or restarting at the finest  
grid level can be  
c      selected. a menu of relaxation methods including  
gauss-seidel point,  
c      line relaxation(s) (in any combination of  
directions) and planar  
c      relaxation (for three-dimensional anisotropic  
problems) are provided.  
c      all methods use ordering based on alternating  
points (red/black),
```

```
c      lines, or planes for cray vectorization and
improved convergence
c      rates [14].
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
estimate (i.e.,
c      discretization level error is reached) then this
can be improved to a
c      fourth-order estimate using the technique of
"deferred corrections."
c      the values in the solution array are used to
generate a fourth-order
c      approximation to the truncation error. second-
order finite difference
c      formula are used to approximate third and fourth
partial derivatives
c      of the solution function [3]. the truncation
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side. the default
multigrid options
c      are used to compute the fourth-order correction
term which is added
c      to the original solution array.
c
c
c ... references (partial)
c
c
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
c      Solution of Linear Elliptic Partial Differential
Equations,"
c      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
c
c      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
c      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
```

```
c      1989, pp.1-12.  
c      .  
c      .  
c      .  
c      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.  
Haidvogel, and V. Pizzo,  
c      "Applications of Multigrid Software in the  
Atmospheric Sciences,"  
c      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-  
1458.  
c      .  
c      .  
c      .  
c      [9] J. Adams, "Recent Enhancements in MUDPACK, a  
Multigrid Software  
c      package for Elliptic Partial Differential  
Equations," Applied Math.  
c      and Comp., 1991, vol. 43, May 1991, pp. 79-94.  
c  
c      [10]J. Adams, "MUDPACK-2: Multigrid Software for  
Approximating  
c      Elliptic Partial Differential Equations on Uniform  
Grids with  
c      any Resolution," Applied Math. and Comp., 1993,  
vol. 53, February  
c      1993, pp. 235-249  
c      .  
c      .  
c      .  
c  
c ... argument description  
c  
c  
c  
*****  
*****  
c *** input arguments  
*****  
c  
*****  
*****  
c  
*****  
*****  
c  
c  
c ... iparm
```

```
c
c      an integer vector of length 22 used to
efficiently pass
c      integer arguments. iparm is set internally
in mud3sp
c      and defined as follows . . .
c
c
c ... intl=iparm(1)
c
c      an initialization argument. intl=0 must be
input
c      on an initial call. in this case input
arguments will
c      be checked for errors and the elliptic
partial differential
c      equation and boundary conditions will be
discretized using
c      second order finite difference formula.
c
c ***      An approximation is NOT generated after an
intl=0 call!
c      mud3sp should be called with intl=1 to
approximate the elliptic
c      PDE discretized by the intl=0 call. intl=1
should also
c      be input if mud3sp has been called earlier
and only the
c      values in in rhs (see below) or gbdy (see
bndyc below)
c      or phi (see below) have changed. This will
bypass
c      redundant pde discretization and argument
checking
c      and save computational time. Some examples
of when
c      intl=1 calls should be used are:
c
c      (0) after a intl=0 argument checking and
discretization call
c
c      (1) mud3sp is being recalled for additional
accuracy. In
c      this case iguess=iparm(12)=1 should also
```

be used.

c

c               (2) mud3sp is being called every time step in  
a time dependent

c                      problem (see discussion below) where the  
elliptic operator

c                      does not depend on time.

c

c               (3) mud3sp is being used to solve the same  
elliptic equation

c                      for several different right hand sides  
(iguess=0 should

c                      probably be used for each new righthand  
side).

c

c                intl = 0 must be input before calling with  
intl = 1 when any

c                      of the following conditions hold:

c

c               (a) the initial call to mud3sp

c               (b) any of the integer arguments other than  
iguess=iparm(12)

c                      or maxcy=iparm(13) or mgopt have changed  
since the previous

c                      call.

c

c               (c) any of the floating point arguments other  
than tolmax=

c                      fparm(5) have changed since the previous  
call

c

c               (d) any of the coefficients input by coef  
(see below) have

c                      changed since the previous call

c

c               (e) any of the "alfa" coefficients input by  
bndyc (see below)

c                      have changed since the previous call.

c

c               If any of (a) through (e) are true then the  
elliptic PDE

c                      must be discretized or rediscretized. If  
none of (a)

```

c           through (e) holds, calls can be made with
intl=1.
c           Incorrect calls with intl=1 will produce
erroneous results.
c ***      The values set in the saved work space "work"
(see below) with
c           an intl=0 call must be preserved with
subsequent intl=1 calls.
c
c           MUDPACK software performance should be
monitored for intl=1
c           calls.  The intl=0 discretization call
performance depends
c           primarily on the efficiency or lack of
efficiency of the
c           user provided subroutines for pde
coefficients and
c           boundary conditions.
c
c
c ... nxa=iparm(2)
c
c           flags boundary conditions on the (y,z) plane
x=xa
c
c           = 0 if p(x,y,z) is periodic in x on [xa,xb]
c           (i.e., p(x+xb-xa,y,z) = p(x,y,z) for all
x,y,z)
c
c           = 1 if p(xa,y,z) is specified (this must be
input thru phi(1,j,k))
c
c           = 2 if there are mixed derivative boundary
conditions at x=xa
c           (see "bndyc" description below where kbdy =
1)
c
c
c ... nxn=iparm(3)
c
c           flags boundary conditions on the (y,z) plane
x=xb
c
c           = 0 if p(x,y,z) is periodic in x on [xa,xb]

```

```

c           (i.e., p(x+xb-xa,y,z) = p(x,y,z) for all
x,y,z)
c
c           = 1 if p(xb,y,z) is specified (this must be
input thru phi(nx,j,k))
c
c           = 2 if there are mixed derivative boundary
conditions at x=xb
c           (see "bndyc" description below where kbdy =
2)
c
c
c ... nyc=iparm(4)
c
c           flags boundary conditions on the (x,z) plane
y=yc
c
c           = 0 if p(x,y,z) is periodic in y on [yc,yd]
c           (i.e., p(x,y+yd-yc,z) = p(x,y,z) for all
x,y,z)

c           = 1 if p(x,yc,z) is specified (this must be
input thru phi(i,1,k))

c           = 2 if there are mixed derivative boundary
conditions at y=yc
c           (see "bndyc" description below where kbdy =
3)
c
c
c ... nyd=iparm(5)
c
c           flags boundary conditions on the (x,z) plane
y=yd
c
c           = 0 if p(x,y,z) is periodic in y on [yc,yd]
c           (i.e., p(x,y+yd-yc,z) = p(x,y,z) for all
x,y,z)

c           = 1 if p(x,yd,z) is specified (this must be
input thru phi(i,ny,k))

c           = 2 if there are mixed derivative boundary
conditions at y=yd

```

```

C           (see "bndyc" description below where kbdy =
4)
C
C
C ... nze=iparm(6)
C
C           flags boundary conditions on the (x,y) plane
z=ze
C
C           = 0 if p(x,y,z) is periodic in z on [ze,zf]
C           (i.e., p(x,y,z+zf-ze) = p(x,y,z) for all
x,y,z

C           = 1 if p(x,y,ze) is specified (this must be
input thru phi(i,j,1))

C           = 2 if there are mixed derivative boundary
conditions at z=ze
C           (see "bndyc" description below where kbdy =
5)
C
C
C ... nzf=iparm(7)
C
C           flags boundary conditions on the (x,y) plane
z=zf
C
C           = 0 if p(x,y,z) is periodic in z on [ze,zf]
C           (i.e., p(x,y,z+zf-ze) = p(x,y,z) for all
x,y,z

C           = 1 if p(x,y,zf) is specified (this must be
input thru phi(i,j,nz))

C           = 2 if there are mixed derivative boundary
conditions at z=zf
C           (see "bndyc" description below where kbdy =
6)
C
C
C *** grid size arguments
C
C
C ... ixp = iparm(8)

```

```
c
c      an integer greater than one which is used in
defining the number
c      of grid points in the x direction (see nx =
iparm(14)). "ixp+1"
c      is the number of points on the coarsest x
grid visited during
c      multigrid cycling. ixp should be chosen as
small as possible.
c      recommended values are the small primes 2 or
3 or (possibly) 5.
c      larger values can reduce multigrid
convergence rates considerably,
c      especially if line relaxation in the x
direction is not used.
c      if ixp > 2 then it should be 2 or a small odd
value since a power
c      of 2 factor of ixp can be removed by
increasing iex = iparm(11)
c      without changing nx = iparm(14)
c
c
c ... jyq = iparm(9)
c
c      an integer greater than one which is used in
defining the number
c      of grid points in the y direction (see ny =
iparm(15)). "jyq+1"
c      is the number of points on the coarsest y
grid visited during
c      multigrid cycling. jyq should be chosen as
small as possible.
c      recommended values are the small primes 2 or
3 or (possibly) 5.
c      larger values can reduce multigrid
convergence rates considerably,
c      especially if line relaxation in the y
direction is not used.
c      if jyq > 2 then it should be 2 or a small odd
value since a power
c      of 2 factor of jyq can be removed by
increasing jey = iparm(12)
c      without changing ny = iparm(15)
c
```

```

c
c ... kzs = iparm(10)
c
c           an integer greater than one which is used in
defining the number
c           of grid points in the z direction (see nz =
iparm(16)). "kzs+1"
c           is the number of points on the coarsest z
grid visited during
c           multigrid cycling. kzs should be chosen as
small as possible.
c           recommended values are the small primes 2 or
3 or (possibly) 5.
c           larger values can reduce multigrid
convergence rates considerably,
c           especially if line relaxation in the z
direction is not used.
c           if kzs > 2 then it should be 2 or a small odd
value since a power
c           of 2 factor of kzs can be removed by
increasing kez = iparm(13)
c           without changing nz = iparm(16)
c
c
c ... iex = iparm(11)
c
c           a positive integer exponent of 2 used in
defining the number
c           of grid points in the x direction (see nx =
iparm(14)).
c           iex .le. 50 is required. for efficient
multigrid cycling,
c           iex should be chosen as large as possible and
ixp=iparm(8)
c           as small as possible within grid size
constraints when
c           defining nx = iparm(14).
c
c
c ... jey = iparm(12)
c
c           a positive integer exponent of 2 used in
defining the number
c           of grid points in the y direction (see ny =

```

```
iparm(15)).  
c           jey .le. 50 is required.  for efficient  
multigrid cycling,  
c           jey should be chosen as large as possible and  
jyq=iparm(9)  
c           as small as possible within grid size  
constraints when  
c           defining ny = iparm(15).  
c  
c  
c ... kez = iparm(13)  
c  
c           a positive integer exponent of 2 used in  
defining the number  
c           of grid points in the z direction (see nz =  
iparm(16)).  
c           kez .le. 50 is required.  for efficient  
multigrid cycling,  
c           kez should be chosen as large as possible and  
kzr=iparm(10)  
c           as small as possible within grid size  
constraints when  
c           defining nz = iparm(16).  
c  
c  
c ... nx = iparm(14)  
c  
c           the number of equally spaced grid points in  
the interval [xa,xb]  
c           (including the boundaries).  nx must have the  
form  
c  
c           nx = ixp*(2** (iex-1)) + 1  
c  
c           where ixp = iparm(8), iex = iparm(11).  
c  
c  
c ... ny = iparm(15)  
c  
c           the number of equally spaced grid points in  
the interval [yc,yd]  
c           (including the boundaries).  ny must have the  
form:  
c
```

```

c           ny = jyq*(2** (jey-1)) + 1
c
c           where jyq = iparm(9), jey = iparm(12).
c
c
c ... nz = iparm(16)
c
c           the number of equally spaced grid points in
c           the interval [ze,zf]
c           (including the boundaries). nz must have the
c           form
c
c           nz = ksr*(2** (kez-1)) + 1
c
c           where ksr = iparm(10), kez = iparm(13)
c
c
c *** example
c
c           suppose a solution is wanted on a 33 by 65 by
c           97 grid. then
c           ixp=2, jyq=4, ksr=6 and iex=jey=kez=5 could be
c           used. a better
c           choice would be ixp=jyq=2, ksr=3, and iex=5,
c           jey=kez=6.
c
c *** note
c
c           let G be the nx by ny by nz fine grid on which the
c           approximation is
c           generated and let n = max0(iex,jey,kez). in
c           mudpack, multigrid
c           cycling is implemented on the ascending chain of
c           grids
c
c           G(1) < ... < G(k) < ... < G(n) = G.
c
c           each g(k) (k=1,...,n) has mx(k) by my(k) by mz(k)
c           grid points
c           given by:
c
c           mx(k) = ixp*[2** (max0(iex+k-n,1)-1)] + 1
c
c           my(k) = jyq*[2** (max0(jey+k-n,1)-1)] + 1

```

```
C
C      mz(k) = kzs* [2** (max0(kez+k-n,1)-1)] + 1
C
C
C
C ... iguess=iparm(17)
C
C           = 0 if no initial guess to the pde is
provided
C           and/or full multigrid cycling beginning
at the
C           coarsest grid level is desired.
C
C           = 1 if an initial guess to the pde at the
finest grid
C           level is provided in phi (see below). in
this case
C           cycling beginning or restarting at the
finest grid
C           is initiated.
C
C *** comments on iguess = 0 or 1 . . .
C
C
C     setting iguess=0 forces full multigrid or "fmg"
cycling. phi
C     must be initialized at all grid points. it can be
set to zero at
C     non-Dirchlet grid points if nothing better is
available. the
C     values set in phi when iguess = 0 are passed and
down and serve
C     as an initial guess to the pde at the coarsest
grid level where
C     multigrid cycling commences.
C
C     if iguess = 1 then the values input in phi are an
initial guess to the
C     pde at the finest grid level where cycling begins.
this option should
C     be used only if a "very good" initial guess is
available (as, for
C     example, when restarting from a previous iguess=0
call).
```

```

c
c *** time dependent problems . . .
c
c      assume we are solving an elliptic pde every time
step in a
c      marching problem of the form:
c
c          l(p(t)) = r(t)
c
c      where the differential operator "l" has no time
dependence,
c      "p(t)" is the solution and "r(t)" is the right
hand side at
c      current time "t". let "dt" be the increment
between time steps.
c      then p(t) can be used as an initial guess to
p(t+dt) with
c      intl = 1 when solving
c
c          l(p(t+dt)) = r(t+dt).
c
c      after the first two time steps, rather than
continue, it would
c      be better to define the "correction" term:
c
c          e(t,dt) = p(t+dt) - p(t)
c
c      this clearly satisfies the equation
c
c          l(e(t,dt)) = r(t+dt) - r(t).
c
c      this should be solved with iguess = 0 and intl =
1. boundary
c      conditions for e(t,dt) are obtained from the
boundary conditions
c      for p(t) by subtracting given values at t from
given values at
c      t+dt. for example if
c
c          d(p(t))/dx = f(t), d(p(t+dt))/dx = f(t+dt)
c
c      at some x boundary then e(t,dt) satisfies the
derivative
c      boundary condition

```

```

c
c      d(e(t,dt))/dx = f(t+dt) - f(t).
c
c      e(t,dt) can be preset to 0.0 (at non-Dirchlet
points) or (if p(t-dt)
c      is saved) to p(t)-p(t-dt). with iguess = 0, these
values will serve
c      as an initial guess to e(t,dt) at the coarsest
grid level. this
c      approach has the advantage that a full sequence of
multigrid cycles,
c      beginning at the coarsest grid level, is invoked
every time step in
c      solving for e(t,dt). a few digits of accuracy in
e(t,dt), which is
c      ordinarily much smaller than p(t), will yield
several more digits of
c      accuracy in the final approximation:
c
c      p(t+dt) = p(t) + e(t,dt).
c
c      using this approach to integrate in time will give
more accuracy
c      then using p(t) as an initial guess to p(t+dt) for
all time steps.
c      it does require additional storage.
c
c      if the differential operator "l" has time
dependence (either thru
c      the coefficients in the pde or the coefficients in
the derivative
c      boundary conditions) then use p(t) as an initial
guess to p(t+dt)
c      when solving
c
c      l(t+dt)(p(t+dt)) = r(t+dt)
c
c      with intl = 0 for all time steps (the
discretization must be repeated
c      for each new "t"). either iguess = 0 (p(t) will
then be an initial
c      guess at the coarsest grid level where cycles will
commence) or
c      iguess = 1 (p(t) will then be an initial guess at

```

```
the finest grid
c      level where cycles will remain fixed) can be
tried.

c
c
c ... maxcy = iparm(18)
c
c           the exact number of cycles executed between
the finest
c           (nx by ny by nz) and the coarsest ((ixp+1) by
(jyq+1) by
c           (kzr+1)) grid levels when tolmax=fparm(7)=0.0
(no error
c           control). when tolmax=fparm(7).gt.0.0 is
input (error control)
c           then maxcy is a limit on the number of cycles
between the
c           finest and coarsest grid levels. in any
case, at most
c           maxcy*(iprert+ipost) relaxation sweeps are
performed at the
c           finest grid level (see
iprer=mgopt(2),ipost=mgopt(3) below)
c           when multigrid iteration is working
"correctly" only a few
c           cycles are required for convergence. large
values for maxcy
c           should not be required.

c
c
c ... method = iparm(19)
c
c
c           = 0 for gauss-seidel pointwise relaxation
with red/black ordering
C
C           This is the only relaxation method offered
with mud3sp. Line
C           or planar relaxation would "lose" the
significant savings in
C           work space length defeating the purpose of
mud3sp. If line
C           or planar relaxation is required then use
muh3 or mud3.
```

```
C           method is used as an argument only to focus
attention on the
C           purpose of mud3sp.
C
c ... length = iparm(20)
C
C           the length of the work space provided in
vector work.
C           This is considerably less then the work space
required by
C           the nonseparable solvers muh3 or mud3.
C
C           length = 7*(nx+2)*(ny+2)*(nz+2)/2
C
C           will usually but not always suffice.  The
exact minimal length
C           depends on the grid size arguments.  It can
be predetermined
C ***      for the current input arguments by calling
mud3sp with iparm(20)
C           set equal to zero and printing iparm(21) or
(in f90) dynamically
C           allocating the work space using the value in
iparm(21) in a
C           subsequent mud3sp call.
C
C ... fparm
C
C           a floating point vector of length 8 used to
efficiently
C           pass floating point arguments.  fparm is set
internally
C           in mud3sp and defined as follows . . .
C
C
C ... xa=fparm(1), xb=fparm(2)
C
C           the range of the x independent variable.  xa
must
C           be less than xb
C
C
C ... yc=fparm(3), yd=fparm(4)
```

```

c           the range of the y independent variable.  yc
must
c           be less than yd.
c
c
c ... ze=fparm(5), zf=fparm(6)
c
c           the range of the z independent variable. ze
must
c           be less than zf.
c
c
c ... tolmax = fparm(7)
c
c           when input positive, tolmax is a maximum
relative error tolerance
c           used to terminate the relaxation iterations.
assume phi1(i,j,k)
c           and phi2(i,j,k) are the last two computed
approximations at all
c           grid points of the finest grid level. if we
define
c
c           phdif = max(abs(phi2(i,j,k)-phi1(i,j,k)))
for all i,j,k
c
c           and
c
c           phmax = max(abs(phi2(i,j,k))) for all
i,j,k
c
c           then "convergence" is considered to have
occurred if and only if
c
c           phdif/phmax < tolmax.
c
c
c           if tolmax=fparm(7)=0.0 is input then there is
no error control
c           and maxcy cycles from the finest grid level
are executed. maxcy
c           is a limit which cannot be exceeded even with
error control.
c           *** calls with tolmax=0.0, when appropriate

```

```

because of known
c           convergence behavior, are more efficient than
calls with tolmax
c           positive (i.e., if possible DO NOT use error
control!) .
c
c
c ... work
c
c           a one dimensional array that must be provided
for work space.
c           see length = iparm(20). the values in work
must be preserved
c           if mud3sp is called again with
intl=iparm(1).ne.0 or if mud34sp
c           is called to improve accuracy.
c
c
c ... bndyc
c
c           a subroutine with arguments
(kbdy,xory,yorz,alfa,gbdy) .
c           which are used to input mixed boundary
conditions to mud3sp.
c           the boundaries are numbered one thru six and
the form of
c           conditions are described below.
c
c
c           (1) the kbdy=1 boundary
c
c           this is the (y,z) plane x=xa where nxa=iparm(2) =
2 flags
c           a mixed boundary condition of the form
c
c           dp/dx + alfxa*p(xa,y,z) = gbdxa(y,z)
c
c           in this case kbdy=1,xory=y,yorz=z will be input to
bndyc and
c           alfa,gbdy corresponding to alfxa,gbdxa(y,z) must
be returned.
c
c
c           (2) the kbdy=2 boundary

```

```

c
c      this is the (y,z) plane x=xb where nxb=iparm(3) =
2 flags
c      a mixed boundary condition of the form
c
c      dp/dx + alfxb*p(xb,y,z) = gbdxb(y,z)
c
c      in this case kbdy=2,xory=y,yorz=z will be input to
bndyc and
c      alfa,gbdy corresponding to alfxb,gbdxb(y,z) must
be returned.
c
c
c      (3) the kbdy=3 boundary
c
c      this is the (x,z) plane y=yc where nyc=iparm(4) =
2 flags
c      a mixed boundary condition of the form
c
c      dp/dy + alfyc*p(x,yc,z) = gbdyc(x,z)
c
c      in this case kbdy=3,xory=x,yorz=z will be input to
bndyc and
c      alfa,gbdy corresponding to alfyc,gbdyc(x,z) must
be returned.
c
c
c      (4) the kbdy=4 boundary
c
c      this is the (x,z) plane y=yd where nyd=iparm(5) =
2 flags
c      a mixed boundary condition of the form
c
c      dp/dy + alfyd*p(x,yd,z) = gbdyd(x,z)
c
c      in this case kbdy=4,xory=x,yorz=z will be input to
bndyc and
c      alfa,gbdy corresponding to alfyd,gbdyd(x,z) must
be returned.
c
c
c      (5) the kbdy=5 boundary
c
c      this is the (x,y) plane z=ze where nze=iparm(6) =

```

```

2 flags
c      a mixed boundary condition of the form
c
c      dp/dz + alfze*p(x,y,ze) = gbdze(x,y)
c
c      in this case kbdy=5,xory=x,yorz=y will be input to
bndyc and
c      alfa,gbdy corresponding to alfze,gbdze(x,y) must
be returned.
c
c
c      (6) the kbdy=6 boundary
c
c      this is the (x,y) plane z=zf where nzf=iparm(7) =
2 flags
c      a mixed boundary condition of the form
c
c      dp/dz + alfzf*p(x,y,zf) = gbdzf(x,y)
c
c      in this case kbdy=6,xory=x,yorz=y will be input to
bndyc and
c      alfa,gbdy corresponding to alfzf,gbdzf(x,y) must
be returned.
c
c
c *** The constants alfxa,alfyc,alfze nonpositive and
alfxb,alfyb,alfze
c      nonnegative will help maintain matrix diagonal
dominance during
c      discretization aiding convergence.
c
c *** bndyc must provide the mixed boundary condition
c      values in correspondence with those flagged in
iparm(2)
c      thru iparm(7). if all boundaries are specified
then
c      mud3sp will never call bndyc. even then it must
be entered
c      as a dummy subroutine. bndyc must be declared
c      external in the routine calling mud3sp. the
actual
c      name chosen may be different.
c
c

```

```
c ... cofx
c
c           a subroutine with arguments (x,cxx,cx,cex)
which provides the
c           known real coefficients of the x derivative
terms for the pde
c           at any grid point x.  the name chosen in the
calling routine
c           may be different where the coefficient routine
must be declared
c           external.

c
c ... cofy
c
c           a subroutine with arguments (y,cyy,cy,cey)
which provides the
c           known real coefficients of the y derivative
terms for the pde
c           at any grid point y.  the name chosen in the
calling routine
c           may be different where the coefficient routine
must be declared
c           external.

c
c ... cofz
c
c           a subroutine with arguments (z,czz,cz,cez)
which provides the
c           known real coefficients of the z derivative
terms for the pde
c           at any grid point z.  the name chosen in the
calling routine
c           may be different where the coefficient routine
must be declared
c           external.

c
c ... rhs
c
c           an array dimensioned nx by ny by nz which
contains
c           the given right hand side values on the
uniform 3-d mesh.
c           rhs(i,j,k) = r(xi,yj,zk) for i=1,...,nx and
j=1,...,ny
```

```
c           and k=1,...,nz.  
c  
c ... phi  
c  
c           an array dimensioned nx by ny by nz .  on  
input phi must  
c           contain specified boundary values and an  
initial guess  
c           to the solution if flagged (see  
iguess=iparm(17)=1).  for  
c           example, if nyd=iparm(5)=1 then phi(i,ny,k)  
must be set  
c           equal to p(xi,yd,zk) for i=1,...,nx and  
k=1,...,nz prior to  
c           calling mud3sp.  the specified values are  
preserved by mud3sp.  
c  
c ***      if no initial guess is given (iguess=0) then  
phi must still  
c           be initialized at non-Dirchlet grid points  
(this is not  
c           checked).  these values are projected down and  
serve as an initial  
c           guess to the pde at the coarsest grid level.  
set phi to 0.0 at  
c           nonDirchlet grid points if nothing better is  
available.  
c  
c  
c ... mgopt  
c  
c           an integer vector of length 4 which allows  
the user to select  
c           among various multigrid options.  if  
mgopt(1)=0 is input then  
c           a default set of multigrid arguments (chosen  
for robustness)  
c           will be internally selected and the  
remaining values in mgopt  
c           will be ignored.  if mgopt(1) is nonzero  
then the arguments  
c           in mgopt are set internally and defined as  
follows: (see the  
c           basic coarse grid correction algorithm
```

```
below)
c
c
c      kcycle = mgopt(1)
c
c          = 0 if default multigrid options are to be
used
c
c          = 1 if v cycling is to be used (the least
expensive per cycle)
c
c          = 2 if w cycling is to be used (the
default)
c
c          > 2 if more general k cycling is to be used
c          *** warning--values larger than 2 increase
c          the execution time per cycle
considerably and
c          result in the nonfatal error ierror =
-5
c          which indicates inefficient multigrid
cycling.
c
c      iprер = mgopt(2)
c
c          the number of "pre-relaxation" sweeps
executed before the
c          residual is restricted and cycling is
invoked at the next
c          coarser grid level (default value is 2
whenever mgopt(1)=0)
c
c      ipost = mgopt(3)
c
c          the number of "post relaxation" sweeps
executed after cycling
c          has been invoked at the next coarser grid
level and the residual
c          correction has been transferred back
(default value is 1
c          whenever mgopt(1)=0).
c
c *** if iprер, ipost, or (especially) kcycle is greater
than 2
```

```
c      than inefficient multigrid cycling has probably
been chosen and
c      the nonfatal error (see below) ierror = -5 will be
set. note
c      this warning may be overridden by any other
nonzero value
c      for ierror.
c
c      interpol = mgopt(4)
c
c          = 1 if multilinear prolongation
c (interpolation) is used to
c          transfer residual corrections and the pde
approximation
c          from coarse to fine grids within full
multigrid cycling.
c
c          = 3 if multicubic prolongation
c (interpolation) is used to
c          transfer residual corrections and the pde
approximation
c          from coarse to fine grids within full
multigrid cycling.
c          (this is the default value whenever
mgopt(1)=0).
c
c *** the default values (2,2,1,3) in the vector mgopt
were chosen for
c      robustness. in some cases v(2,1) cycles with
linear prolongation will
c      give good results with less computation
(especially in two-dimensions).
c      this was the default and only choice in an
earlier version of mudpack
c      (see [1]) and can be set with the integer vector
(1,2,1,1) in mgopt.
c
c *** the schedules for one full multigrid cycle
(iguess=0) using v(2,1)
c      cycles and w(2,1) cycles are depicted for a four
level grid below.
c      the number of relaxation sweeps when each grid is
visited are indicated.
c      the "*" stands for prolongation of the full
```

approximation and the "."

c stands for transfer of residuals and residual corrections within the

c coarse grid correction algorithm (see below).

c

c one fmg with v(2,1) cycles:

c

c

c -----
 
 2-----1-

----- level 4

c \* .

c \*

c -----2-----1-----2-----1-----2-----1-----

----- level 3

c \* .

c \*

c -----2-----1-----2-----1-----2-----1-----

----- level 2

c \* . .

c \* . .

c -----3-----3-----3-----3-----3-----

----- level 1

c

c

c one fmg with w(2,1) cycles:

c

c -----
 
 2-----1-

--1-- level 4

c \* .

c .

c -----2-----1-----2-----3-----

1---- level 3

c \* .

c \*

c -----2-----1-----2-----3-----1-----2-----3-----1-----

----- level 2

c \* . .

c \* . .

c -----6-----6-----6-----6-----6-----6-----6-----

----- level 1

c

c

c the form of the "recursive" coarse grid correction cycling used

c when kcycle.ge.0 is input is described below in pseudo-algorithmic

```

c      language. it is implemented non-recursively in
fortran in mudpack.

c
c      algorithm
cgc(k,l(k),u(k),r(k),kcycle,iprer,ipost,iressw,intpol)
c
c *** approximately solve l(k)*u(k) = r(k) using
multigrid iteration
c *** k is the current grid level
c *** l(k) is the discretized pde operator at level k
c *** u(k) is the initial guess at level k
c *** r(k) is the right hand side at level k
c *** i(k,k-1) is the restriction operator from level k
to level k-1
c *** (the form of i(k,k-1) depends on iressw)
c *** i(k-1,k) is the prolongation operator from level
k-1 to level k
c *** (the form of i(k-1,k) depends on intpol)
c
c      begin algorithm cgc
c
c ***     pre-relax at level k
c
c      . do (i=1,iprer)
c
c      . . relax(l(k),u(k),r(k))
c
c      . end do
c
c      . if (k > 1) then
c
c ***      restrict the residual from level k to level k-
1
c
c      . . r(k-1) = i(k,k-1) (r(k)-l(k)*u(k))
c
c      . . kount = 0
c
c      . . repeat
c
c ***      solve for the residual correction at level k-1
in u(k-1)
c ***      using algorithm cgc "kcycle" times (this is
the recursion)

```

```

C
C      . . . kount = kount+1
C
C      . . . invoke cgc(k-1,l(k-1),u(k-1),r(k-
1),kcycle,iprер,ipost,iresw)
C
C
C      . . until (kount.eq.kcycle)
C
C ***      transfer residual correction in u(k-1) to
level k
C ***      with the prolongation operator and add to u(k)
C
C      . . u(k) = u(k) + i(k-1,k) (u(k-1))
C
C      . end if
C
C ***      post relax at level k
C
C      . do (i=1,ipost)
C
C      . . relax(l(k),u(k),r(k))
C
C      . end do
C
C      . return
C
C      end algorithm cgc
C
C
C ****output
arguments*****
***
```

\*\*\*\*\*

```

C
arguments*****
***
```

\*\*\*\*\*

```

C
C
C ... iparm(21)
C
C          on output iparm(21) contains the actual work
space length

```

```

c           required for the current grid sizes and
method. This value
c           will be computed and returned even if
iparm(20) is less than
c           iparm(21) (see ierror=9).
c
c
c ... iparm(22)
c
c           if error control is selected (tolmax =
fparm(7) .gt. 0.0) then
c           on output iparm(22) contains the actual
number of cycles executed
c           between the coarsest and finest grid levels
in obtaining the
c           approximation in phi. the quantity
(iprter+ipost)*iparm(22) is
c           the number of relaxation sweeps performed at
the finest grid level.
c
c
c ... fparm(8)
c
c           on output fparm(8) contains the final
computed maximum relative
c           difference between the last two iterates at
the finest grid level.
c           fparm(8) is computed only if there is error
control (tolmax.gt.0.)
c           assume phil(i,j,k) and phi2(i,j,k) are the
last two computed
c           values for phi(i,j,k) at all points of the
finest grid level.
c           if we define
c
c           phdif = max (abs(phi2(i,j,k)-phil(i,j,k)))
for all i,j,k
c
c           and
c
c           phmax = max (abs(phi2(i,j,k))) for all
i,j,k
c
c           then

```

```
C
C           fparm(8) = phdif/phmax
C
C           is returned whenever phmax.gt.0.0.  in the
degenerate case
C           phmax = 0.0, fparm(8) = phdif is returned.
C
C
C
C ... work
C
C           on output work contains intermediate values
that must not be
C           destroyed if mud3sp is to be called again
with iparm(1)=1 or
C           if mud34sp is to be called to improve the
estimate to fourth
C           order.
C
C
C ... phi
C
C           on output phi(i,j,k) contains the
approximation to
C           p(xi,yj,zk) for all mesh points i=1,...,nx;
j=1,...,ny;
C           k=1,...,nz.  the last computed iterate in phi
is returned
C           even if convergence is not obtained (ierror=-
1)
C
C ... ierror
C
C           For intl=iparm(1)=0 initialization calls,
ierror is an
C           error flag that indicates invalid input
arguments when
C           returned positive and nonfatal warnings when
returned
C           negative.  Argument checking and
discretization
C           is bypassed for intl=1 calls which can only
return
C           ierror = -1 or 0 or 1.
C
```

```

c
c      non-fatal warnings * * *
c
c
c      ==5 if kcycle=mgopt(1) is greater than 2. values
c      larger than 2 results
c          in an algorithm which probably does far more
c      computation than
c          necessary.  kcycle = 1 (v cycles) or kcycle=2
c      (w cycles) should
c          suffice for most problems.  ierror = -5 is
c      also set if either
c          iprer = mgopt(2) or ipost=mgopt(3) is greater
c      than 2.  the
c          ierror=-5 flag is overridden by any other
c      fatal or non-fatal
c          error.
c
c
c      ==4 if there are dominant nonzero first order
c      terms in the pde which
c          make it "hyperbolic" at the finest grid level.
c      numerically, this
c          happens if:
c
c          abs(cx)*dlx > 2.*abs(cxx)      (dlx = (xb-
c      xa) / (nx-1))
c
c                      (or)
c
c          abs(cy)*dly > 2.*abs(cyy)      (dly = (yd-
c      yc) / (ny-1))
c
c
c          at some fine grid point (xi,yj).  if an
c      adjustment is not made the
c          condition can lead to a matrix coming from the
c      discretization
c          which is not diagonally dominant and
c      divergence is possible. since
c          the condition is "likely" at coarser grid
c      levels for pde's with
c          nonzero first order terms, the adjustments
c      (actually first order
c          approximations to the pde)

```

```

C
C
C           cxx = amax1(cxx,0.5*abs(cx)*dx)
C
C           (and)
C
C           cyy = amax1(cyy,0.5*abs(cy)*dy)
C
C
C           (here dx,dy are the x,y mesh sizes of the
C           subgrid)
C
C           are made to preserve convergence of multigrid
C           iteration. if made
C           at the finest grid level, it can lead to
C           convergence to an
C           erroneous solution (flagged by ierror = -4).
C           a possible remedy
C           is to increase resolution. the ierror = -4
C           flag overrides the
C           nonfatal ierror = -5 flag.
C
C
C           ==3 if the continuous elliptic pde is singular.
C           this means the
C           boundary conditions are periodic or pure
C           derivative at all
C           boundaries and ce(x,y) = 0.0 for all x,y. a
C           solution is still
C           attempted but convergence may not occur due
C           to ill-conditioning
C           of the linear system coming from the
C           discretization. the
C           ierror = -3 flag overrides the ierror=-4 and
C           ierror=-5 nonfatal
C           flags.
C
C
C           ==2 if the pde is not elliptic (i.e.,
C           cxx*cyy.le.0.0 for some (xi,yj))
C           in this case a solution is still attempted
C           although convergence
C           may not occur due to ill-conditioning of the
C           linear system.

```

```

c           the ierror = -2 flag overrides the ierror=-
5,-4,-3 nonfatal
c           flags.
c
c
c     ==1 if convergence to the tolerance specified in
tolmax=fparm(5)>0.
c           is not obtained in maxcy=iparm(13) multigrid
cycles between the
c           coarsest (ixp+1,jyq+1) and finest (nx,ny)
grid levels.
c           in this case the last computed iterate is
still returned.
c           the ierror = -1 flag overrides all other
nonfatal flags
c
c
c     no errors * * *
c
c     = 0
c
c     fatal argument errors * * *
c
c     = 1 if intl=iparm(1) is not 0 on initial call or
not 0 or 1
c           on subsequent calls
c
c     = 2 if any of the boundary condition flags
nxa,nxb,nyc,nyd,nze,nzf
c           in iparm(2) through iparm(7) is not 0,1 or 2 or
if
c           (nxa,nxb) or (nyc,nyd) or (nze,nzf) are not
pairwise zero.
c
c     = 3 if mino(ixp,jyq,kzr) < 2
(ixp=iparm(8),jyq=iparm(9),kzr=iparm(10))
c
c     = 4 if min0(iex,jey,kez) < 1
(iex=iparm(11),jey=iparm(12),kez=iparm(13))
c           or if max0(iex,jey,kez) > 50
c
c     = 5 if nx.ne.ixp*2** (iex-1)+1 or if
ny.ne.jyq*2** (jey-1)+1 or
c           if nz.ne.kzr*2** (kez-1)+1

```

```

(nx=iparm(14),ny=iparm(15),nz=iparm(16))
C
C      = 6 if iguess = iparm(17) is not equal to 0 or 1
C
C      = 7 if maxcy = iparm(18) < 1 (large values for
maxcy should not be used)
C
C      = 8 if method = iparm(19) is not equat to zero
C
C      = 9 if length = iparm(20) is too small (see
iparm(21) on output
C          for minimum required work space length)
C
C      =10 if xa >= xb or yc >= yd or ze >= zf
C
(xa=fparm(1),xb=fparm(2),yc=fparm(3),yd=fparm(4),ze=fpar
m(5),zf=fparm(6))
C
C      =11 if tolmax = fparm(7) < 0.0
C
C      errors in setting multigrid options * * * (see
also ierror=-5)
C
C      =12 if kcycle = mgopt(1) < 0 or
C          if iprer = mgopt(2) < 1 or
C          if ipost = mgopt(3) < 1 or
C          if interpol = mgopt(4) is not 1 or 3
C
C
*****
*
C
*****
*
C
C      end of mud3sp documentation
C
C
*****
**
C
*****
**

```

```
C  
C
```

---

## MUH2

```
C  
C      file muh2.d  
C  
C      * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C      *  
*  
C      *          copyright (c) 2008 by UCAR  
*  
C      *  
*  
C      *          University Corporation for Atmospheric  
Research           *  
C      *  
*  
C      *          all rights reserved  
*  
C      *  
*  
C      *          MUDPACK version 5.0.1  
*  
C      *  
*  
C      *          A Fortran Package of Multigrid  
*  
C      *  
*  
C      *          Subroutines and Example Programs  
*  
C      *  
*  
C      *          for Solving Elliptic Partial Differential  
Equations       *
```

```
C      *
*
C      *                                by
*
C      *
*
C      *                                John Adams
*
C      *
*
C      *                                of
*
C      *
*
C      *                                the National Center for Atmospheric
Research          *
C      *
*
C      *                                Boulder, Colorado (80307)
U.S.A.          *
C      *
*
C      *                                which is sponsored by
*
C      *
*
C      *                                the National Science Foundation
*
C      *
*
C      *      * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C
C ... file muh2.d
C
C contains documentation for:
C subroutine
muh2(iparm,fparm,wk,iwk,coef,bndyc,rhs,phi,mgopt,ierror)
C a sample fortran driver is file "tmuh2.f".
C
C ... required mudpack files
C
C mudcom.f, muhcom.f
C
```

```

c ... purpose
c
c      the "hybrid" multigrid/direct method code muh2
approximates the
c      same 2-d nonseparable elliptic pde as the mudpack
solver mud2.
c      muh2 combines the efficiency of multigrid
iteration with the certainty
c      a direct method. the basic algorithm is modified
by using banded
c      gaussian elimination in place of relaxation
whenever the coarsest
c      subgrid is encountered within multigrid cycling.
this provides
c      additional grid size flexibility by eliminating
the usual multigrid
c      constraint that the coarsest grid consist of "few"
points for effective
c      error reduction with multigrid cycling. In many
cases the hybrid method
c      provides more robust convergence characteristics
than multigrid cycling
c      alone.

c
c      The form of the pde solved is:
c
c
c      cxx(x,y)*pxx + cyy(x,y)*pyy + cx(x,y)*px +
cy(x,y)*py +
c
c      ce(x,y)*p(x,y) = r(x,y).
c
c
c      pxx,pyy,px,py are second and first partial
derivatives of the
c      unknown real solution function p(x,y) with respect
to the
c      independent variables x,y. cxx,cyy,cx,cy,ce are
the known
c      real coefficients of the elliptic pde and r(x,y)
is the known
c      real right hand side of the equation. cxx and cyy
should be
c      positive for all x,y in the solution region.

```

```

nonseparability
c      means some of the coefficients depend on both x
and y. if
c      the pde is separable subroutine mud2sp should be
used instead
c      of mud2 or muh2.
c
c *** muh2 becomes a full direct method if grid size
arguments are chosen
c      so that the coarsest and finest grids coincide.
choosing iex=jey=1
c      and ixp=nx-1, jyq=ny-1
(iex=iparm(6),jey=iparm(7),ixp=iparm(8),
c      jyq=iparm(9),nx=iparm(10),ny=iparm(11)) will set
gaussian elimination
c      on the nx by ny grid. in this case, muh2 produces
a direct solution
c      to the same nonseparable elliptic pde as the
direct solver liptic [5,6].
c      muh2 is more general than liptic since it allows
periodic boundary
c      conditions in the y direction.
c
c
c ... argument differences with mud2.f
c
c      the input and output arguments of muh2 are almost
identical to the
c      arguments of mud2 (see mud2.d) with the following
exceptions:
c
c      (1) the work space vector "wk" requires
c
c                  (ixp+1)*(jyq+1)*(2*ixp+3)
c
c      additional words of storage (ixp = iparm(6),
jyq = iparm(7))
c      if periodic boundary conditions are not
flagged in the y direction
c      (nyc .ne. 0 where nyc = iparm(4)) or
c
c                  (ixp+1)*[2*(ixp+1)*(2*jyq-1)+jyq+1]
c
c      additional words of storage if periodic

```

```
boundary conditions are
c           flagged in the y direction (nyc = 0).  the
extra work space is
c           used for a direct solution with gaussian
elimination whenever the
c           coarsest grid is encountered within multigrid
cycling.

c
c     (2) An integer work space iwk of length at least
(ixp+1)*(jyq+1)
c           must be provided.

c
c     (3) jyq must be greater than 2 if periodic
boundary conditions
c           are flagged in the y direction and ixp must be
greater than
c           2 if periodic boundary conditions are flagged
in the x direction.
c           inputting jyq = 2 when nyc = 0 or inputting
ixp = 2 when nxz = 0
c           will set the fatal error flag ierror=3

c
c *** (4) it is no longer necessary that ixp and jyq be
"small" for
c           effective error reduction with multigrid
iteration.  there
c           is no reduction in convergence rates when
larger values for
c           ixp or jyq are used .  this provides
additional flexibility
c           in choosing grid size.  in many cases muh2
provides more
c           robust convergence than mud2.  it can be used
in place of
c           mud2 for all nonsingular problems (see (5)
below) .

c
c     (5) iguess = iparm(11) = 1 (flagging an initial
guess) or
c           maxcy = iparm(14) > 1 (setting more than one
multigrid
c           cycle) are not allowed if muh2 becomes a full
direct method
c           by choosing iex = jey = 1 (iex = iparm(8),jey
```

```
= iparm(9)).  
c           this conflicting combination of input  
arguments for multigrid  
c           iteration and a full direct method set the  
fatal error flag  
c  
c           ierror = 13  
c  
c           iguess = 0 and maxcy = 1 are required when  
muh2 becomes a  
c           full direct method.  
c  
c           (6) if a "singular" pde is detected (see ierror=-3  
description in mud2.d;  
c           ce(x,y) = 0.0 for all x,y and the boundary  
conditions are a combination  
c           of periodic and/or pure derivatives) then muh2  
sets the fatal error  
c           flag  
c  
c           ierror = 14  
c  
c           The direct method utilized by muh2 would  
likely cause a division  
c           by zero in the singular case. mud2 can be  
tried for singular problems  
c  
c  
c ... grid size considerations  
c  
c           (1) flexibility  
c  
c           muh2 should be used in place of mud2 whenever  
grid size  
c           requirements do not allow choosing ixp and jyq  
to be "small"  
c           positive integers (typically less than 4).  
c  
c           example:  
c  
c           suppose we wish to solve an elliptic pde on a  
one degree grid on  
c           the full surface of a sphere. choosing ixp =  
jyq = 45 and iex = 4
```

```

c           and jyq = 3 fits the required 361 by 181 grid
exactly. multigrid
c           cycling will be used on the sequence of
subgrid sizes:
c
c           46 x 46 < 91 x 46 < 181 x 91 < 361 x
181
c
c           the 46 x 46 coarsest subgrid has too much
resolution for effective
c           error reduction with relaxation only. muh2
circumvents this
c           difficulty by generating an exact direct
solution (modulo roundoff
c           error) whenever the coarsest grid is
encountered.
c
c           (2) additional work space (see (1) under
"arguments differences") is
c           required by muh2 to implement gaussian
elimination at the coarsest
c           grid level. this may limit the size of ixp
and jyq.
c
c           (3) operation counts
c
c           for simplicity, assume p = ixp = jyq and n =
nx = ny. banded
c           gaussian elimination requires  $\mathcal{O}(p^{**4})$ 
operations for solution
c           on the coarsest subgrid while multigrid
iteration is a  $\mathcal{O}(n^{**2})$ 
c           algorithm. these are approximately balanced
when
c
c            $p^{**4} =: (n / (2^{**k}))^{**4} =: n^{**2}$ 
c
c           or
c
c            $k =: \log_2(n) / 2$ 
c
c           grid levels are chosen with the hybrid method.
so if
c           p is approximately equal to

```

```

c
c      n/ (2** (log2 (n) /2))
c
c      then the direct method and multigrid parts of
c      the hybrid algorithm
c      require roughly the same amount of computer
c      time. larger values
c      for p mean the direct method will dominate the
c      computation. smaller
c      values mean the hybrid method will cost only
c      marginally more than
c      multigrid iteration with coarse grid
c      relaxation.
c
c
c *** the remaining documentation is almost identical to
mud2.d
c      except for the modifications already indicated.
c
c ... mesh description . . .
c
c      the approximation is generated on a uniform nx by
ny grid. the grid
c      is superimposed on the rectangular solution region
c
c      [xa,xb] x [yc,yd].
c
c      let
c
c      dlx = (xb-xa) / (nx-1), dly = (yd-yc) / (ny-1)
c
c      be the uniform grid increments in the x,y
c      directions. then
c
c      xi=xa+(i-1)*dlx,   yj=yc+(j-1)*dly
c
c      for i=1,...,nx and j=1,...,ny denote the x,y
c      uniform mesh points
c
c
c ... language
c
c      fortran90/fortran77
c

```

```
c
c ... portability
c
c      mudpack5.0.1 software has been compiled and tested
with fortran77
c      and fortran90 on a variety of platforms.
c
c ... methods
c
c      details of the methods employed by the solvers in
mudpack are given
c      in [1,9]. [1,2,9] contain performance
measurements on a variety of
c      elliptic pdes (see "references" in the file
"readme"). in summary:
c
c *** discretization and solution (second-order solvers)
(see [1])
c
c      the pde and boundary conditions are automatically
discretized at all
c      grid levels using second-order finite difference
formula. diagonal
c      dominance at coarser grid levels is maintained in
the presence of
c      nonzero first-order terms by adjusting the second-
order coefficient
c      when necessary. the resulting block tri-diagonal
linear system is
c      approximated using multigrid iteration
[10,11,13,15,16,18]. version
c      5.0.1 of mudpack uses only fully weighted residual
restriction. defaults
c      include cubic prolongation and w(2,1) cycles.
these can be overridden
c      with selected multigrid options (see "mgopt").
error control based on
c      maximum relative differences is available. full
multigrid cycling (fmg)
c      or cycling beginning or restarting at the finest
grid level can be
c      selected. a menu of relaxation methods including
gauss-seidel point,
c      line relaxation(s) (in any combination of
```

```
directions) and planar
c      relaxation (for three-dimensional anisotropic
problems) are provided.
c      all methods use ordering based on alternating
points (red/black),
c      lines, or planes for cray vectorization and
improved convergence
c      rates [14].
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
estimate (i.e.,
c      discretization level error is reached) then this
can be improved to a
c      fourth-order estimate using the technique of
"deferred corrections."
c      the values in the solution array are used to
generate a fourth-order
c      approximation to the truncation error. second-
order finite difference
c      formula are used to approximate third and fourth
partial derivatives
c      of the solution function [3]. the truncation
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side. the default
multigrid options
c      are used to compute the fourth-order correction
term which is added
c      to the original solution array.
c
c
c ... references (partial)
c
c
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
c      Solution of Linear Elliptic Partial Differential
Equations,"
c      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
```

```
C
C      [2] J. Adams, "FMG Results with the Multigrid
Software Package MUDPACK,"
C      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
C      1989, pp.1-12.
C      .
C      .
C      .
C      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
Haidvogel, and V. Pizzo,
C      "Applications of Multigrid Software in the
Atmospheric Sciences,"
C      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-
1458.
C      .
C      .
C      .
C      [9] J. Adams, "Recent Enhancements in MUDPACK, a
Multigrid Software
C      package for Elliptic Partial Differential
Equations," Applied Math.
C      and Comp., 1991, vol. 43, May 1991, pp. 79-94.
C
C      [10]J. Adams, "MUDPACK-2: Multigrid Software for
Approximating
C      Elliptic Partial Differential Equations on Uniform
Grids with
C      any Resolution," Applied Math. and Comp., 1993,
vol. 53, February
C      1993, pp. 235-249
C      .
C      .
C      .
C      ...
C      argument description
C
C
C
C*****
*****  

C *** input arguments
*****  

C
```

```
*****
*****
C
C
C ... iparm
C
C           an integer vector of length 17 used to pass
integer
C           arguments. iparm is set internally and
defined as
C           follows:
C
C
C ... intl=iparm(1)
C
C           an initialization argument. intl=0 must be
input
C           on an initial call. in this case input
arguments will
C           be checked for errors and the elliptic
partial differential
C           equation and boundary conditions will be
discretized using
C           second order finite difference formula.
C
C ***      an approximation is not generated after an
intl=0 call!
C           muh2 should be called with intl=1 to
approximate the elliptic
C           pde discretized by the intl=0 call. intl=1
should also
C           be input if muh2 has been called earlier and
only the
C           values in in rhs (see below) or gbdy (see
bndyc below)
C           or phi (see below) have changed. this will
bypass
C           redundant pde discretization and argument
checking
C           and save computational time. some examples
of when
C           intl=1 calls should be used are:
C
C           (0) after a intl=0 argument checking and
```

```
discretization call
c
c           (1) muh2 is being recalled for additional
accuracy.  in
c           this case iguess=iparm(12)=1 should also
be used.
c
c           (2) muh2 is being called every time step in a
time dependent
c           problem (see discussion below) where the
elliptic operator
c           does not depend on time.
c
c           (3) muh2 is being used to solve the same
elliptic equation
c           for several different right hand sides
(igueess=0 should
c           probably be used for each new righthand
side) .
c
c           intl = 0 must be input before calling with
intl = 1 when any
c           of the following conditions hold:
c
c           (a) the initial call to muh2
c
c           (b) any of the integer arguments other than
igueess=iparm(12)
c           or maxcy=iparm(13) or mgopt have changed
since the previous
c           call.
c
c           (c) any of the floating point arguments other
than tolmax=
c           fparm(5) have changed since the previous
call
c
c           (d) any of the coefficients input by coef
(see below) have
c           changed since the previous call
c
c           (e) any of the "alfa" coefficients input by
bndyc (see below)
c           have changed since the previous call.
```

```
c
c           if any of (a) through (e) are true then the
elliptic pde
c           must be discretized or rediscretized.  if
none of (a)
c           through (e) holds, calls can be made with
intl=1.
c           incorrect calls with intl=1 will produce
erroneous results.
c ***      the values set in the saved work space "work"
(see below) with
c           an intl=0 call must be preserved with
subsequent intl=1 calls.
c
c           MUDPACK software performance should be
monitored for intl=1
c           calls.  The intl=0 discretization call
performance depends
c           primarily on the efficiency or lack of
efficiency of the
c           user provided subroutines for pde
coefficients and
c           boundary conditions.

c
c
c ... nxa=iparm(2)
c
c           flags boundary conditions on the edge x=xa
c
c           = 0 if p(x,y) is periodic in x on [xa,xb]
c           (i.e., p(x+xb-xa,y) = p(x,y) for all x,y
c           (if nxa=0 then nxb=0 is required, see
ierror = 2)
c
c           = 1 if p(xa,y) is specified (this must be input
thru phi(1,j))
c
c           = 2 if there are mixed derivative boundary
conditions at x=xa
c           (see bndyc)
c
c
c ... nxb=iparm(3)
```

```

c           flags boundary conditions on the edge x=xb
c
c           = 0 if p(x,y) is periodic in x on [xa,xb]
c                   (i.e., p(x+xb-xa,y) = p(x,y) for all x,y)
c                   (if nxb=0 then nxa=0 is required, see
ierror = 2)
c
c           = 1 if p(xb,y) is specified (this must be input
thru phi(nx,j))
c
c           = 2 if there are mixed derivative boundary
conditions at x=xb
c                   (see bndyc)
c
c
c ... nyc=iparm(4)
c
c           flags boundary conditions on the edge y=yc
c
c           = 0 if p(x,y) is periodic in y on [yc,yd]
c                   (i.e., p(x,y+yd-yc) = p(x,y) for all x,y
c                   (if nyc=0 then nyd=0 is required, see
ierror = 2)
c
c           = 1 if p(x,yc) is specified (this must be input
thru phi(i,1))
c
c           = 2 if there are mixed derivative boundary
conditions at y=yc
c                   (see bndyc)
c
c
c ... nyd=iparm(5)
c
c           flags boundary conditions on the edge y=yd
c
c           = 0 if p(x,y) is periodic in y on [yc,yd]
c                   (i.e., p(x,y+yd-yc) = p(x,y) for all x,y
c                   (if nyd=0 then nyc=0 is required, see
ierror = 2)
c
c           = 1 if p(x,yd) is specified (this must be input
thru phi(i,ny))
c

```

```

c      = 2 if there are mixed derivative boundary
conditions at y=yd
c          (see bndyc)
c
c
c *** grid size arguments
c
c
c ... ixp = iparm(6)
c
c      an integer greater than one which is used in
defining the number
c          of grid points in the x direction (see nx =
iparm(10)). "ixp+1"
c          is the number of points on the coarsest x
grid visited during
c          multigrid cycling. ixp should be chosen as
small as possible.
c          recommended values are the small primes 2 or
3.
c          larger values can reduce multigrid
convergence rates considerably,
c          especially if line relaxation in the x
direction is not used.
c          if ixp > 2 then it should be 2 or a small odd
value since a power
c          of 2 factor of ixp can be removed by
increasing iex = iparm(8)
c          without changing nx = iparm(10).
c
c
c ... jyq = iparm(7)
c
c      an integer greater than one which is used in
defining the number
c          of grid points in the y direction (see ny =
iparm(11)). "jyq+1"
c          is the number of points on the coarsest y
grid visited during
c          multigrid cycling. jyq should be chosen as
small as possible.
c          recommended values are the small primes 2 or
3.
c          larger values can reduce multigrid

```

```
convergence rates considerably,  
c especially if line relaxation in the y  
direction is not used.  
c if jyq > 2 then it should be 2 or a small odd  
value since a power  
c of 2 factor of jyq can be removed by  
increasing jey = iparm(9)  
c without changing ny = iparm(11).  
c  
c  
c ... iex = iparm(8)  
c  
c a positive integer exponent of 2 used in  
defining the number  
c of grid points in the x direction (see nx =  
iparm(10)).  
c iex .le. 50 is required. for efficient  
multigrid cycling,  
c iex should be chosen as large as possible and  
ixp=iparm(8)  
c as small as possible within grid size  
constraints when  
c defining nx.  
c  
c  
c ... jey = iparm(9)  
c  
c a positive integer exponent of 2 used in  
defining the number  
c of grid points in the y direction (see ny =  
iparm(11)).  
c jey .le. 50 is required. for efficient  
multigrid cycling,  
c jey should be chosen as large as possible and  
jyq=iparm(7)  
c as small as possible within grid size  
constraints when  
c defining ny.  
c  
c  
c  
c ... nx = iparm(10)  
c  
c the number of equally spaced grid points in
```

```

the interval [xa,xb]
c           (including the boundaries).  nx must have the
form
c
c           nx = ixp*(2** (iex-1)) + 1
c
c           where ixp = iparm(6), iex = iparm(8).
c
c
c ... ny = iparm(11)
c
c           the number of equally spaced grid points in
the interval [yc,yd]
c           (including the boundaries).  ny must have the
form:
c
c           ny = jyq*(2** (jey-1)) + 1
c
c           where jyq = iparm(7), jey = iparm(9).
c
c
c *** example
c
c           suppose a solution is wanted on a 33 by 97
grid.  then
c           ixp=2, jyq=6 and iex=jey=5 could be used.  a
better
c           choice would be ixp=2, jyq=3, and iex=5,
jey=6.
c
c *** grid size flexibility considerations:
c
c           the hybrid multigrid/direct method code muh2
provides more grid size
c           flexibility than mud2 by removing the constraint
that ixp and jyq are
c           2 or 3.  this is accomplished by using a direct
method whenever the
c           coarsest (ixp+1) x (jyq+1) grid is encountered in
multigrid cycling.
c           if nx = ixp+1 and ny = jyq+1 then muh2 becomes a
full direct method.
c           muh2 is roughly equivalent to mud2 in efficiency
as long as ixp and

```

```

c      jyq remain "small".  if the problem to be
approximated requires
c      a grid neither mud2 por muh2 can exactly fit then
another option
c      is to generate an approximation on a "close grid"
using mud2 or muh2.
c      then transfer the result to the required grid
using cubic interpolation
c      via the package "regridpack"(contact john adams
about this software)
c
c *** note
c
c      let G be the nx by ny fine grid on which the
approximation is
c      generated and let n = max0(iex,jey).  in mudpack,
multigrid
c      cycling is implemented on the ascending chain of
grids
c
c          G(1) < ... < G(k) < ... < G(n) = g.
c
c      each G(k) (k=1,...,n) has mx(k) by my(k) grid
points
c      given by:
c
c          mx(k) = ixp*[2**max0(iex+k-n,1)-1] + 1
c
c          my(k) = jyq*[2**max0(jey+k-n,1)-1] + 1
c
c      If iex = jey = 1 then G(1) = G(n) and muh2 solves
the problem
c      directly with block banded Gaussian elimination.
Otherwise
c      muh2 replaces relaxation with a direct method on
G(1).
c
c ... iguess=iparm(12)
c
c          = 0 if no initial guess to the pde is
provided
c
c          = 1 if an initial guess to the pde is at the
finest grid

```

```
c           level is provided in phi (see below)
c
c   comments on iguess = 0 or 1 . . .
c
c   even if iguess = 0, phi must be initialized at all
grid points (this
c   is not checked). phi can be set to 0.0 at non-
dirchlet grid points
c   if nothing better is available. the values set in
phi when iguess = 0
c   are passed down and serve as an initial guess to
the pde at the coarsest
c   grid level where cycling commences. in this
sense, values input in
c   phi always serve as an initial guess. setting
iguess = 0 forces full
c   multigrid cycling beginning at the coarsest and
finishing at the finest
c   grid level.
c
c   if iguess = 1 then the values input in phi are an
initial guess to the
c   pde at the finest grid level where cycling begins.
this option should
c   be used only if a "very good" initial guess is
available (as, for
c   example, when restarting from a previous iguess=0
call).
c
c   time dependent problems . . .
c
c *** assume we are solving an elliptic pde every time
step in a
c   marching problem of the form:
c
c   l(p(t)) = r(t)
c
c   where the differential operator "l" has no time
dependence,
c   "p(t)" is the solution and "r(t)" is the right
hand side at
c   current time "t". let "dt" be the increment
between time steps.
c   then p(t) can be used as an initial guess to
```

```

p(t+dt) with
c      intl = 1 when solving
c
c      l(p(t+dt)) = r(t+dt).
c
c      after the first two time steps, rather than
continue, it would
c      be better to define the "correction" term:
c
c      e(t,dt) = p(t+dt) - p(t)
c
c      this clearly satisfies the equation
c
c      l(e(t,dt)) = r(t+dt) - r(t).
c
c      this should be solved with iguess = 0 and intl =
1. boundary
c      conditions for e(t,dt) are obtained from the
boundary conditions
c      for p(t) by subtracting given values at t from
given values at
c      t+dt. for example if
c
c      d(p(t))/dx = f(t), d(p(t+dt))/dx = f(t+dt)
c
c      at some x boundary then e(t,dt) satisfies the
derivative
c      boundary condition
c
c      d(e(t,dt))/dx = f(t+dt) - f(t).
c
c      e(t,dt) can be preset to 0.0 (at nondirichlet
points) or (if p(t-dt)
c      is saved) to p(t)-p(t-dt). with iguess = 0, these
values will serve
c      as an initial guess to e(t,dt) at the coarsest
grid level. this
c      approach has the advantage that a full sequence of
multigrid cycles,
c      beginning at the coarsest grid level, is invoked
every time step in
c      solving for e(t,dt). a few digits of accuracy in
e(t,dt), which is
c      ordinarily much smaller than p(t), will yield

```

```

several more digits of
c      accuracy in the final approximation:
c
c      p(t+dt) = p(t) + e(t,dt).
c
c      using this approach to integrate in time will give
more accuracy
c      then using p(t) as an initial guess to p(t+dt) for
all time steps.
c      it does require additional storage.
c
c      if the differential operator "l" has time
dependence (either thru
c      the coefficients in the pde or the coefficients in
the derivative
c      boundary conditions) then use p(t) as an initial
guess to p(t+dt)
c      when solving
c
c      l(t+dt) (p(t+dt)) = r(t+dt)
c
c      with intl = 0 for all time steps (the
discretization must be repeated
c      for each new "t"). either iguess = 0 (p(t) will
then be an initial
c      guess at the coarsest grid level where cycles will
commence) or
c      iguess = 1 (p(t) will then be an initial guess at
the finest grid
c      level where cycles will remain fixed) can be
tried.
c
c
c ... maxcy = iparm(13)
c
c      the exact number of cycles executed between
the finest (nx by
c      ny) and the coarsest ((ixp+1) by (jyq+1))
grid levels when
c      tolmax=fparm(5)=0.0 (no error control). when
tolmax > 0.0
c      is input (error control) then maxcy is a
limit on the number
c      of cycles between the finest and coarsest

```

```
grid levels. in
c           any case, at most maxcy*(iprert+ipost)
relaxation sweeps are
c           are performed at the finest grid level (see
iprert=mgopt(2),
c           ipost=mgopt(3) below). when multigrid
iteration is working
c           "correctly" only a few are required for
convergence. large
c           values for maxcy should not be necessary.
c
c
c ... method = iparm(14) determines the method of
relaxation
c           (gauss-seidel based on alternating points
or lines)
c
c           = 0 for point relaxation
c
c           = 1 for line relaxation in the x direction
c
c           = 2 for line relaxation in the y direction
c
c           = 3 for line relaxation in both the x and y
direction
c
c
c *** choice of method. . .
c
c     let fx represent the quantity cxx(x,y)/dlx**2 over
the solution region.
c
c     let fy represent the quantity cyy(x,y)/dly**2 over
the solution region
c
c     if fx,fy are roughly the same size and do not vary
too much over
c     the solution region choose method = 0. if this
fails try method=3.
c
c     if fx is much greater than fy choose method = 1.
c
c     if fy is much greater than fx choose method = 2
c
```

```

c      if neither fx or fy dominates over the solution
region and they
c      both vary considerably choose method = 3.
c
c
c ... length = iparm(15)
c
c      the length of the work space provided in
vector work (see below).
c      let isx = 0 if method = 0 or method = 2
c      let isx = 3 if method = 1 or method = 3 and
nxa.ne.0
c      let isx = 5 if method = 1 or method = 3 and
nxa.eq.0
c      let jsy = 0 if method = 0 or method = 1
c      let jsy = 3 if method = 2 or method = 3 and
nyc.ne.0
c      let jsy = 5 if method = 2 or method = 3 and
nyc.eq.0
c
c      let ldir = (ixp+1)*(jyq+1)*(2*ixp+3) if
nyc.ne.0 or
c          let ldir = (ixp+1)*[2*(ixp+1)*(2*jyq-
1)+jyq+1] if nyc=0
c
c      then . . .
c
c      length =
4*[nx*ny*(10+isx+jsy)+8*(nx+ny+2)]/3 + ldir
c
c      will suffice in most cases. the exact
minimal work space
c      length required for the current nx,ny and
method is output
c      in iparm(16) (even if iparm(15) is too
small). this will be
c      less then the value given by the simplified
formula above
c      in most cases.
c
c
c ... fparm
c
c      a floating point vector of length 6 used to

```

```
efficiently
c           pass floating point arguments.  fparm is set
internally
c           in muh2 and defined as follows . . .
c
c
c ... xa=fparm(1), xb=fparm(2)
c
c           the range of the x independent variable. xa
must
c           be less than xb
c
c
c ... yc=fparm(3), yd=fparm(4)
c
c           the range of the y independent variable. yc
must
c           be less than yd.
c
c
c ... tolmax = fparm(5)
c
c           when input positive, tolmax is a maximum
relative error tolerance
c           used to terminate the relaxation iterations.
assume phil(i,j)
c           and phi2(i,j) are the last two computed
approximations at all
c           grid points of the finest grid level. if we
define
c
c           phdif = max(abs(phi2(i,j)-phil(i,j))) for
all i,j
c
c           and
c
c           phmax = max(abs(phi2(i,j))) for all i,j
c
c           then "convergence" is considered to have
occurred if and only if
c
c           phdif/phmax < tolmax.
c
c
```

```
c           if tolmax=fparm(5)=0.0 is input then there is
no error control
c           and maxcy cycles from the finest grid level
are executed. maxcy
c           is a limit which cannot be exceeded even with
error control.
c   *** calls with tolmax=0.0, when appropriate
because of known
c           convergence behavior, are more efficient than
calls with tolmax
c           positive (i.e., if possible DO NOT error
control!).
c
c ... wk
c
c           a one dimensional real saved work space (see
iparm(15) for
c           length) which must be preserved from the
previous call when
c           calling with intl=iparm(1)=1.
c
c ... iwk
c
c           an integer vector dimensioned of length at
least (ixp+1)*(jyq+1)
c           (ixp = iparm(6),jyq=iparm(7)) in the routine
calling muh2.
c           The length of iwk is not checked! If iwk has
length less than
c           (ixp+1)*(jyq+1) then undetectable errors will
result.
c
c ... bndyc
c
c           a subroutine with arguments
(kbdy,xory,alfa,gbdy) which
c           are used to input mixed boundary conditions
to muh2. bndyc
c           must be declared "external" in the program
calling muh2.
c           the boundaries are numbered one thru four and
the mixed
c           derivative boundary conditions are described
below (see the
```

```

C           sample driver code "tmuh2.f" for an example
of how bndyc is
C           can beset up) .

C
C           * * * * * * * * * * * * * * *   y=yd
C           *           kbdy=4           *
C           *
C           *
C           *
C           *
C           * kbdy=1           kbdy=2   *
C           *           *
C           *
C           *
C           *
C           *           kbdy=3           *
C           * * * * * * * * * * * * *   y=yc
C
C           x=xa           x=xb
C
C
C           (1) the kbdy=1 boundary
C
C           this is the edge x=xa where nxa=iparm(2)=2
flags
C           a mixed boundary condition of the form
C
C           dp/dx + alfxa(y) *p(xa,y) = gbdxa(y)
C
C           in this case kbdy=1,xory=y will be input to
bndyc and
C           alfa,gbdy corresponding to alfxa(y),gbdxa(y)
must be returned.
C
C
C           (2) the kbdy=2 boundary
C
C           this is the edge x=xb where nxb=iparm(3)=2
flags
C           a mixed boundary condition of the form
C
C           dp/dx + alfxb(y) *p(xb,y) = gbdxb(y)
C
C           in this case kbdy=2,xory=y, will be input to
bndyc and
C           alfa,gbdy corresponding to alfxb(y),gbdxb(y)

```

must be returned.

C

C

C                   (3) the kbdy=3 boundary

C

C                   this is the edge y=yc where nyc=iparm(4)=2

flags

C                   a mixed boundary condition of the form

C

C                    $\frac{dp}{dy} + \text{alfyc}(x) * p(x, yc) = \text{gbdyc}(x)$

C

C                   in this case kbdy=3, xory=x will be input to

bndyc and

C                   alfa, gbdy corresponding to alfy(x), gbdyc(x)

must be returned.

C

C

C                   (4) the kbdy=4 boundary

C

C                   this is the edge y=yd where nyd=iparm(5)=2

flags

C                   a mixed boundary condition of the form

C

C                    $\frac{dp}{dy} + \text{alfyd}(x) * p(x, yd) = \text{gbdyd}(x)$

C

C                   in this case kbdy=4, xory=x will be input to

bndyc and

C                   alfa, gbdy corresponding to alfyd(x), gbdyd(x)

must be returned.

C

C

C \*\*\*           bndyc must provide the mixed boundary

condition values

C                   in correspondence with those flagged in

iparm(2) thru

C                   iparm(5). if all boundaries are specified or

periodic

C                   muh2 will never call bndyc. even then it

must be entered

C                   as a dummy subroutine. bndyc must be declared

"external"

C                   in the routine calling muh2. the actual name

chosen may

C                   be different.

```

c
c
c ... coef
c
c           a subroutine with arguments
(x,y,cxx,cyy,cx,cy,ce) which
c           provides the known real coefficients for the
elliptic pde at
c           any grid point (x,y). the name chosen in the
calling routine
c           may be different where the coefficient routine
must be declared
c           "external."
c
c ... rhs
c
c           an array dimensioned nx by ny which contains
the given
c           right hand side values on the uniform 2-d
mesh.
c
c           rhs(i,j) = r(xi,yj) for i=1,...,nx and
j=1,...,ny
c
c ... phi
c
c           an array dimensioned nx by ny. on input phi
must contain
c           specified boundary values. for example, if
nyd=iparm(5)=1
c           then phi(i,ny) must be set equal to p(xi,yd)
for i=1,...nx
c           prior to calling muh2. these values are
preserved by muh2.
c           if an initial guess is provided
(iguess=iparm(11)=1) it must
c           be input thru phi.
c
c
c ***      if no initial guess is given (iguess=0) then
phi must still
c           be initialized at all grid points (this is
not checked). these
c           values will serve as an initial guess to the

```

```

pde at the coarsest
c           grid level after a transfer from the fine
solution grid.  set phi
c           equal to to 0.0 at all internal and non-
specified boundaries
c           grid points if nothing better is available.
c
c
c ... mgopt
c
c           an integer vector of length 4 which allows
the user to select
c           among various multigrid options.  if
mgopt(1)=0 is input then
c           a default set of multigrid parameters
(chosen for robustness)
c           will be internally selected and the
remaining values in mgopt
c           will be ignored.  if mgopt(1) is nonzero
then the parameters
c           in mgopt are set internally and defined as
follows: (see the
c           basic coarse grid correction algorithm
below)
c
c
c     kcycle = mgopt(1)
c
c           = 0 if default multigrid options are to be
used
c
c           = 1 if v cycling is to be used (the least
expensive per cycle)
c
c           = 2 if w cycling is to be used (the
default)
c
c           > 2 if more general k cycling is to be used
c           *** warning--values larger than 2 increase
c           the execution time per cycle
considerably and
c           result in the nonfatal error ierror =
-5
c           which indicates inefficient multigrid

```

```
cycling.  
c  
c      iprер = mgopt(2)  
c  
c          the number of "pre-relaxation" sweeps  
executed before the  
c          residual is restricted and cycling is  
invoked at the next  
c          coarser grid level (default value is 2  
whenever mgopt(1)=0)  
c  
c      ipost = mgopt(3)  
c  
c          the number of "post relaxation" sweeps  
executed after cycling  
c          has been invoked at the next coarser grid  
level and the residual  
c          correction has been transferred back  
(default value is 1  
c          whenever mgopt(1)=0).  
c  
c *** if iprер, ipost, or (especially) kcycle is greater  
than 2  
c      than inefficient multigrid cycling has probably  
been chosen and  
c      the nonfatal error (see below) ierror = -5 will be  
set. note  
c      this warning may be overridden by any other  
nonzero value  
c      for ierror.  
c  
c      intpol = mgopt(4)  
c  
c          = 1 if multilinear prolongation  
(interpolation) is used to  
c          transfer residual corrections and the pde  
approximation  
c          from coarse to fine grids within full  
multigrid cycling.  
c  
c          = 3 if multicubic prolongation  
(interpolation) is used to  
c          transfer residual corrections and the pde  
approximation
```

```

c           from coarse to fine grids within full
multigrid cycling.
c           (this is the default value whenever
mgopt(1)=0).
c
c *** the default values (2,2,1,3) in the vector mgopt
were chosen for
c       robustness. in some cases v(2,1) cycles with
linear prolongation will
c       give good results with less computation
(especially in two-dimensions).
c       this was the default and only choice in an
earlier version of mudpack
c       (see [1]) and can be set with the integer vector
(1,2,1,1) in mgopt.
c
c *** the schedules for one full multigrid cycle
(iguess=0) using v(2,1)
c       cycles and w(2,1) cycles are depicted for a four
level grid below.
c       the number of relaxation sweeps when each grid is
visited are indicated.
c       the "*" stands for prolongation of the full
approximation and the "."
c       stands for transfer of residuals and residual
corrections within the
c       coarse grid correction algorithm (see below). all
version 5.0.1
c       mudpack solvers use only fully weighted residual
restriction. The
c       "D" at grid level 1 indicates a direct method is
used.
c
c       one fmg with v(2,1) cycles:
c
c
c       -----
c       -----2-----1-
c       level 4
c               *
c               .
c               .
c       -----
c       -----2-----1-----2-----1-----
c       level 3
c               *
c               .
c               .
c               .
c               .

```

```

C      -----2-----1-----2-----1-----2-----1-----
-----      level 2
C      * . .
C      * . .
C      ---D---D-----D-----D-----
-----      level 1
C
C
C
C      one fmg with w(2,1) cycles:
C
C      -----
C      -----2-----
--1--      level 4
C                  *
.
C      -----2-----1-----2-----3-----
1----      level 3
C                  *
C      -----2---1---2---3---1-----2---3---1---2---3---1-
-----      level 2
C      * . . . . . . . . . . . . .
C      --D---D-----D---D-----D---D-----D---D-----
-----      level 1
C
C
C      the form of the "recursive" coarse grid correction
cycling used
c      when kcycle.ge.0 is input is described below in
pseudo-algorithmic
c      language. it is implemented non-recursively in
fortran in mudpack.
c *** this algorithm is modified with the hybrid
solvers which use
c      a direct method whenever grid level 1 is
encountered.
C
C      algorithm
cgc(k,l(k),u(k),r(k),kcycle,iprer,ipost,iresw,intpol)
C
C      *** approximately solve l(k)*u(k) = r(k) using
multigrid iteration
C      *** k is the current grid level
C      *** l(k) is the discretized pde operator at level k
C      *** u(k) is the initial guess at level k

```

```

c *** r(k) is the right hand side at level k
c *** i(k,k-1) is the restriction operator from level k
to level k-1
c *** (the form of i(k,k-1) depends on iresw)
c *** i(k-1,k) is the prolongation operator from level
k-1 to level k
c *** (the form of i(k-1,k) depends on interpol)
c
c      begin algorithm cgc
c
c ***    pre-relax at level k
c
c      . do (i=1,iprer)
c
c      . . relax(l(k),u(k),r(k))
c
c      . end do
c
c      . if (k > 1) then
c
c ***      restrict the residual from level k to level k-
1
c
c      . . r(k-1) = i(k,k-1) (r(k)-l(k)*u(k))
c
c      . . kount = 0
c
c      . . repeat
c
c ***      solve for the residual correction at level k-1
in u(k-1)
c ***      using algorithm cgc "kcycle" times (this is
the recursion)
c
c      . . . kount = kount+1
c
c      . . . invoke cgc(k-1,l(k-1),u(k-1),r(k-
1),kcycle,iprer,ipost,iresw)
c
c
c      . . until (kount.eq.kcycle)
c
c ***      transfer residual correction in u(k-1) to
level k

```

```

C ***      with the prolongation operator and add to u(k)
C
C      . . u(k) = u(k) + i(k-1,k) (u(k-1))
C
C      . end if
C
C *** post relax at level k
C
C      . do (i=1,ipost)
C
C      . . relax(l(k),u(k),r(k))
C
C      . end do
C
C      . return
C
C      end algorithm cgc
C
C
C
C ****
C ****
C *** output arguments
C ****
C
C
C ... iparm(16) *** set for intl=0 calls only
C
C          on output iparm(16) contains the actual work
C space length
C          required. this will usually be less than
C that given by the
C          simplified formula for length=iparm(15) (see
C as input argument)
C
C
C ... iparm(17) *** set for intl=1 calls only
C
C          on output iparm(17) contains the actual
C number of multigrid cycles
C          between the finest and coarsest grid levels

```

```

used to obtain the
c           approximation when error control (tolmax >
0.0) is set.
c
c
c ... fparm(6) *** set for intl=1 calls with fparm(5)
> 0. only
c
c           on output fparm(6) contains the final
computed maximum relative
c           difference between the last two iterates at
the finest grid level.
c           fparm(6) is computed only if there is error
control (tolmax > 0.0)
c           assume phi1(i,j,k) and phi2(i,j,k) are the
last two computed
c           values for phi(i,j,k) at all points of the
finest grid level.
c           if we define
c
c           phdif = max(abs(phi2(i,j)-phi1(i,j)))
over all i,j
c
c           and
c
c           phmax = max(abs(phi2(i,j))) over all i,j
c
c           then
c
c           fparm(6) = phdif/phmax
c
c           is returned whenever phmax > 0.0. in the
degenerate case
c           phmax = 0.0, fparm(6) = phdif is returned.
c
c
c ... work
c
c           on output work contains intermediate values
that must not
c           be destroyed if muh2 is to be called again
with intl=1
c
c

```

```
c ... phi *** for intl=1 calls only
c
c           on output phi(i,j) contains the approximation
to p(xi,yj)
c           for all mesh points i = 1,...,nx and
j=1,...,ny. the last
c           computed iterate in phi is returned even if
convergence is
c           not obtained
c
c
c ... ierror
c
c           for intl=iparm(1)=0 initialization calls,
ierror is an
c           error flag that indicates invalid input
arguments when
c           returned positive and nonfatal warnings when
returned
c           negative. discretization is bypassed for
intl=1 calls
c           which can only return ierror = -1 or 0 or 1.
c
c
c     non-fatal warnings * * *
c
c
c     ==5 if kcycle=mgopt(1) is greater than 2. values
larger than 2 results
c           in an algorithm which probably does far more
computation than
c           necessary. kcycle = 1 (v cycles) or kcycle=2
(w cycles) should
c           suffice for most problems. ierror = -5 is
also set if either
c           iprer = mgopt(2) or ipost=mgopt(3) is greater
than 2. the
c           ierror=-5 flag is overridden by any other
fatal or non-fatal
c           error.
c
c     ==4 if there are dominant nonzero first order
terms in the pde which
c           make it "hyperbolic" at the finest grid level.
```



```

c
c
c      ==2 if the pde is not elliptic (i.e.,
c      cxx*cyy.le.0.0 for some (xi,yj))
c          in this case a solution is still attempted
c          although convergence
c          may not occur due to ill-conditioning of the
c          linear system.
c          the ierror = -2 flag overrides the ierror=-
c, -4 nonfatal
c          flags.
c
c
c      ==1 if convergence to the tolerance specified in
c      tolmax=iparm(5)>0.
c          is not obtained in maxcy=iparm(13) multigrid
c      cycles between the
c          coarsest (ixp+1,jyq+1) and finest (nx,ny)
c      grid levels.
c          in this case the last computed iterate is
c      still returned.
c          the ierror = -1 flag overrides all other
c      nonfatal flags
c
c
c      no errors * * *
c
c      = 0
c
c      fatal argument errors * * *
c
c      = 1 if intl=iparm(1) is not 0 on initial call or
c      not 0 or 1
c          on subsequent calls
c
c      = 2 if any of the boundary condition flags
nxa,nxb,nyc,nyd
c          in iparm(2),iparm(3),iparm(4),iparm(5) are not
0,1 or 2
c          or if nxa,nxb or nyc,nyd are not pairwise
zero.
c
c      = 3 if mino(ixp,jyq) < 2 (ixp = iparm(6), jyq =
iparm(7))

```

```

c      of if ixp < 3 when nxa=0 or if jyq < 3 when
nyc=0.
c
c      = 4 if min0(iex,jey) < 1 (iex = iparm(8), jey =
iparm(9)) or
c          if max0(iex,jey) > 50
c
c      = 5 if nx.ne.ipx*2** (iex-1)+1 or
ny.ne.jyq*2** (jey-1)+1
c          (nx = iparm(10), ny = iparm(11))
c
c      = 6 if iguess = iparm(12) is not equal to 0 or 1
c
c      = 7 if maxcy = iparm(13) < 1
c
c      = 8 if method = iparm(14) is not 0,1,2, or 3
c
c      = 9 if length = iparm(15) is too small (see
iparm(16) on output
c          for minimum required work space length)
c
c      =10 if xa >= xb or yc >= yd
c
c (xa=fparm(1),xb=fparm(2),yc=fparm(3),yd=fparm(4))
c
c      =11 if tolmax = fparm(5) < 0.0
c
c      errors in setting multigrid options * * * (see
also ierror=-5)
c
c      =12 if kcycle = mgopt(1) < 0 or
c          if iprer = mgopt(1) < 1 or
c          if ipost = mgopt(3) < 1 or
c          if interpol = mgopt(4) is not 1 or 3
c
c      =13 if iex=jey=1 (full direct method) and iguess=1
or maxcy > 1
c
c      =14 if the elliptic pde is singular (see ierror=-3
in mud2.d)
c
c
*****
*
```

```
C
*****
*
C
C      end of muh2 documentation
C
C
*****
**
C
*****
**
C
C
```

---

## MUH24

```
C
C      file muh24.d
C
C      * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK version 5.0.1
*
```



```
C
C ... file muh24.d
C
C     contains documentation for:
C     subroutine muh24(wk,iwk,phi,ierror)
C     A sample fortran driver is file "tmuh24.f".
C
C ... required MUDPACK files
C
C     muh2.f, mudcom.f
C
C ... purpose
C
C     muh24 attempts to improve the estimate in phi,
C     obtained by calling
C         muh2, from second to fourth order accuracy. see
C     the file "muh2.d"
C         for a detailed discussion of the elliptic pde
C     approximated and
C         arguments "wk,iwk,phi" which are also part of the
C     argument list for
C         muh2.
C
C ... assumptions
C
C     * phi contains a second-order approximation from
C     an earlier muh2 call
C
C     * arguments "wk,iwk,phi" are the same used in
C     calling muh2
C
C     * "wk,iwk,phi" have not changed since the last
C     call to muh2
C
C     * the finest grid level contains at least 6
C     points in each direction
C
C
C *** warning
C
C     if the first assumption is not true then a fourth
C     order approximation
C     cannot be produced in phi. the last assumption
C     (adequate grid size)
```

```
c      is the only one checked. failure in any of the
others can result in
c      in an undetectable error.
c
c ... language
c
c      fortran90/fortran77
c
c ... portability
c
c      mudpack5.0.1 software has been compiled and tested
with fortran77
c      and fortran90 on a variety of platforms.
c
c ... methods
c
c      details of the methods employed by the solvers in
mudpack are given
c      in [1,9]. [1,2,9] contain performance
measurements on a variety of
c      elliptic pdes (see "references" in the file
"readme"). in summary:
c
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
estimate (i.e.,
c      discretization level error is reached) then this
can be improved to a
c      fourth-order estimate using the technique of
"deferred corrections"
c      the values in the solution array are used to
generate a fourth-order
c      approximation to the truncation error. second-
order finite difference
c      formula are used to approximate third and fourth
partial derivatives
c      of the solution function [3]. the truncation
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side. the default
```

```
multigrid options
c      are used to compute the fourth-order correction
term which is added
c      to the original solution array.
c
c
c ... references (partial)
c
c
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
c      Solution of Linear Elliptic Partial Differential
Equations,"
c      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
c
c      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
c      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
c      1989, pp.1-12.
c      .
c      .
c      .
c      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
Haidvogel, and V. Pizzo,
c      "Applications of Multigrid Software in the
Atmospheric Sciences,"
c      Mon. Wea. Rev.,vol. 120 # 7, July 1992, pp. 1447-
1458.
c      .
c      .
c      .
c      [9] J. Adams, "Recent Enhancements in MUDPACK, a
Multigrid Software
c      package for Elliptic Partial Differential
Equations," Applied Math.
c      and Comp., 1991, vol. 43, May 1991, pp. 79-94.
c
c      [10]J. Adams, "MUDPACK-2: Multigrid Software for
Approximating
c      Elliptic Partial Differential Equations on Uniform
Grids with
c      any Resolution," Applied Math. and Comp., 1993,
```

vol. 53, February  
c 1993, pp. 235-249  
c .  
c .  
c .  
c  
c ... error parameter  
c  
c = 0 if no error is detected  
c  
c = 30 if min0(nx,ny) < 6 where nx,ny are the fine  
grid sizes  
c in the x,y directions.  
c  
c  
c  
\*\*\*\*\*  
\*\*\*\*\*  
c  
\*\*\*\*\*  
\*\*\*\*\*  
\*\*\*\*\*  
c  
c end of muh24 documentation  
c  
c  
\*\*\*\*\*  
\*\*\*\*\*  
c  
\*\*\*\*\*  
\*\*\*\*\*  
c  
\*\*\*\*\*  
\*\*\*\*\*

MUH24CR

```
* * * * * *
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK version 5.0.1
*
C      *
*
C      *               A Fortran Package of Multigrid
*
C      *
*
C      *               Subroutines and Example Programs
*
C      *
*
C      *               for Solving Elliptic Partial Differential
Equations        *
C      *
*
C      *               by
*
C      *
*
C      *               John Adams
*
C      *
*
C      *               of
*
C      *
*
C      *               the National Center for Atmospheric
```

```
Research          *
C      *
*
C      *                      Boulder, Colorado (80307)
U.S.A.          *
C      *
*
C      *                      which is sponsored by
*
C      *
*
C      *                      the National Science Foundation
*
C      *
*
C      *      *      *      *      *      *      *      *      *      *      *
*      *      *      *      *      *      *      *      *      *      *      *
C
C ... file muh24cr.d
C
C     contains documentation for:
C     subroutine muh24cr(wk,iwk,coef,bndyc,phi,ierror)
C     A sample fortran driver is file "tmuh24cr.f".
C
C ... required MUDPACK files
C
C     muh2cr.f, mudcom.f
C
C ... purpose
C
C     muh24cr attempts to improve the estimate in phi,
C     obtained by calling
C     muh2cr, from second to fourth order accuracy.
see the file "muh2cr.d"
C     for a detailed discussion of the elliptic pde
approximated and
C     arguments "wk,iwk,coef,bndyc,phi" which are also
part of the argument
C     list for muh2cr.
C
C ... assumptions
C
C     * phi contains a second-order approximation from
an earlier muh2cr call
```

```
c
c      * arguments "wk,iwk,coef,bndyc,phi" are the same
used in calling muh2cr
c
c      * "wk,iwk,coef,bndyc,phi" have not changed since
the last call to muh2cr
c
c      * the finest grid level contains at least 6
points in each direction
c
c
c *** warning
c
c      if the first assumption is not true then a fourth
order approximation
c      cannot be produced in phi.  the last assumption
(adequate grid size)
c      is the only one checked. failure in any of the
others can result in
c      in an undetectable error.
c
c ... language
c
c      fortran90/fortran77
c
c ... portability
c
c      mudpack5.0.1 software has been compiled and tested
with fortran77
c      and fortran90 on a variety of platforms.
c
c ... methods
c
c      details of the methods employed by the solvers in
mudpack are given
c      in [1,9].  [1,2,9] contain performance
measurements on a variety of
c      elliptic pdes (see "references" in the file
"readme") .  in summary:
c
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
```

```
c      if the multigrid cycling results in a second-order
c      estimate (i.e.,
c      discretization level error is reached) then this
c      can be improved to a
c      fourth-order estimate using the technique of
c      "deferred corrections"
c      the values in the solution array are used to
c      generate a fourth-order
c      approximation to the truncation error. second-
c      order finite difference
c      formula are used to approximate third and fourth
c      partial derivatives
c      of the solution function [3]. the truncation
c      error estimate is
c      transferred down to all grid levels using weighted
c      averaging where
c      it serves as a new right hand side. the default
c      multigrid options
c      are used to compute the fourth-order correction
c      term which is added
c      to the original solution array.

c
c
c ... references (partial)

c
c
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
c          for the Efficient
c          Solution of Linear Elliptic Partial Differential
c          Equations,"
c          Applied Math. and Comput. vol.34, Nov 1989,
c          pp.113-146.

c
c      [2] J. Adams,"FMG Results with the Multigrid
c          Software Package MUDPACK,"
c          proceedings of the fourth Copper Mountain
c          Conference on Multigrid, SIAM,
c          1989, pp.1-12.

c      .
c      .
c      .
c      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
c          Haidvogel, and V. Pizzo,
c          "Applications of Multigrid Software in the
```

```
Atmospheric Sciences,"  
c      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-  
1458.  
c      .  
c      .  
c      .  
c      [9] J. Adams, "Recent Enhancements in MUDPACK, a  
Multigrid Software  
c      package for Elliptic Partial Differential  
Equations," Applied Math.  
c      and Comp., 1991, vol. 43, May 1991, pp. 79-94.  
c  
c      [10]J. Adams, "MUDPACK-2: Multigrid Software for  
Approximating  
c      Elliptic Partial Differential Equations on Uniform  
Grids with  
c      any Resolution," Applied Math. and Comp., 1993,  
vol. 53, February  
c      1993, pp. 235-249  
c      .  
c      .  
c      .  
c ... error parameter  
c  
c      = 0 if no error is detected  
c  
c      = 30 if min0(nx,ny) < 6 where nx,ny are the fine  
grid sizes  
c          in the x,y directions.  
c  
c  
c  
*****  
*****  
c  
*****  
*****  
*****  
c  
c      end of muh24cr documentation  
c  
c  
*****  
*****  
c
```

```
*****
*****
C
```

---

## MUH2CR

```
C
C      file muh2cr.d
C
C      * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *                               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *                               all rights reserved
*
C      *
*
C      *           MUDPACK   version 5.0.1
*
C      *
*
C      *           A Fortran Package of Multigrid
*
C      *
*
C      *           Subroutines and Example Programs
*
C      *
*
C      *           for Solving Elliptic Partial Differential
```

Equations \*  
C \*  
\*  
C \* by  
\*  
C \*  
\* John Adams  
\*  
C \*  
\*  
C \* of  
\*  
C \*  
\*  
C \* the National Center for Atmospheric  
Research \*  
C \*  
\*  
C \* Boulder, Colorado (80307)  
U.S.A. \*  
C \*  
\*  
C \* which is sponsored by  
\*  
C \*  
\*  
C \* the National Science Foundation  
\*  
C \*  
\*  
C \*  
\* \* \* \* \* \* \*  
C  
C ... file muh2cr.d  
C  
C contains documentation for:  
C subroutine  
muh2cr(iparm,fparm,work,coef,bndyc,rhs,phi,mgopt,ierror)  
C a sample fortran driver is file "tmuh2cr.f".  
C  
C ... required mudpack files  
C  
C mudcom.f

```
c
c ... purpose
c
c      the "hybrid" multigrid/direct method code muh2cr
c approximates the
c      same 2-d nonseparable elliptic pde as the mudpack
c solver mud2cr.
c      muh2cr combines the efficiency of multigrid
c iteration with the certainty
c      a direct method. the basic algorithm is modified
c by using banded
c      gaussian elimination in place of relaxation
c whenever the coarsest
c      subgrid is encountered within multigrid cycling.
c this provides
c      additional grid size flexibility by eliminating
c the usual multigrid
c      constraint that the coarsest grid consist of "few"
c points for effective
c      error reduction with multigrid cycling. in many
c cases the hybrid method
c      provides more robust convergence characteristics
c than multigrid cycling
c      alone.
c
c      the form of the pde solved is:
c
c      cxx(x,y)*pxx + cxy(x,y)*pxy + cyy(x,y)*pyy +
c      cx(x,y)*px +
c
c      cy(x,y)*py + ce(x,y)*p(x,y) = r(x,y) .
c
c
c      pxx,pxy,pyy,px,py are second and first partial
c derivatives of the
c      unknown real solution function p(x,y) with respect
c to the
c      independent variables x,y. cxx,cxy,cyy,cx,cy,ce
c are the known
c      real coefficients of the elliptic pde and r(x,y)
c is the known
c      real right hand side of the equation. cxx and cyy
c should be
c      positive for all x,y in the solution region and
```

```

c
c      4*cxx(x,y)*cyy(x,y) .le. cxy(x,y)**2
c
c      for ellipticity (see ierror=-2).  nonseparability
means some
c      of the coefficients depend on both x and y and
cxy.ne.0.  if
c      the pde is separable and cxy = 0 then subroutine
muh2sp should
c      be used.
c
c *** muh2cr becomes a full direct method if grid size
arguments are chosen
c      so that the coarsest and finest grids coincide.
choosing iex=jey=1
c      and ixp=nx-1, jyq=ny-1
(iex=iparm(6),jey=iparm(7),ixp=iparm(8),
c      jyq=iparm(9),nx=iparm(10),ny=iparm(11)) will set
gaussian elimination
c      on the nx by ny grid.  in this case, muh2cr
produces a direct solution
c      to the same nonseparable elliptic pde as the
direct solver crosel [5,6].
c      muh2cr is more general than crosel since it allows
periodic boundary
c      conditions in the y direction.
c
c
c ... argument differences with mud2cr.f
c
c      the input and output arguments of muh2cr are
almost identical to the
c      arguments of mud2cr (see mud2cr.d) with the
following exceptions:
c
c      (1) the work space vector "wk" requires
c
c            (ixp+1)*(jyq+1)*(2*ixp+3)
c
c      additional words of storage (ixp = iparm(6),
jyq = iparm(7))
c      if periodic boundary conditions are not
flagged in the y direction
c      (nyc .ne. 0 where nyc = iparm(4)) or

```

```

c
c          (ixp+1)*[2*(ixp+1)*(2*jyq-1)+jyq+1]
c
c          additional words of storage if periodic
boundary conditions are
c          flagged in the y direction (nyc = 0). the
extra work space is
c          used for a direct solution with gaussian
elimination whenever the
c          coarsest grid is encountered within multigrid
cycling.
c
c          (2) an integer work space iwk of length at least
(ixp+1)*(jyq+1)
c          must be provided.
c
c          (3) jyq must be greater than 2 if periodic
boundary conditions
c          are flagged in the y direction and ixp must be
greater than
c          2 if periodic boundary conditions are flagged
in the x direction.
c          inputting jyq = 2 when nyc = 0 or inputting
ixp = 2 when nxz = 0
c          will set the fatal error flag ierror=3
c
c *** (4) it is no longer necessary that ixp and jyq be
"small" for
c          effective error reduction with multigrid
iteration. there
c          is no reduction in convergence rates when
larger values for
c          ixp or jyq are used . this provides
additional flexibility
c          in choosing grid size. in many cases muh2cr
provides more
c          robust convergence than mud2cr. it can be
used in place of
c          mud2cr for all nonsingular problems (see (6)
below) .
c
c          (5) iguess = iparm(11) = 1 (flagging an initial
guess) or
c          maxcy = iparm(14) > 1 (setting more than one

```

```
multigrid
c           cycle) are not allowed if muh2cr becomes a
full direct method
c           by choosing iex = jey = 1 (iex = iparm(8),jey
= iparm(9)).
c           this conflicting combination of input
arguments for multigrid
c           iteration and a full direct method set the
fatal error flag
c
c           ierror = 13
c
c           iguess = 0 and maxcy = 1 are required when
muh2 becomes a
c           full direct method.
c
c           (6) if a "singular" pde is detected (see ierror=-3
description in mud2cr.d;
c           ce(x,y) = 0.0 for all x,y and the boundary
conditions are a combination
c           of periodic and/or pure derivatives) then
muh2cr sets the fatal error
c           flag
c
c           ierror = 14
c
c           the direct method utilized by muh2cr would
likely cause a division
c           by zero in the singular case. mud2cr can be
tried for singular problems
c
c
c ... grid size considerations
c
c           (1) flexibility
c
c           muh2cr should be used in place of mud2cr
whenever grid size
c           requirements do not allow choosing ixp and jyq
to be "small"
c           positive integers (typically less than 4).
c
c           example:
```

```

c      suppose we wish to solve an elliptic pde on a
one degree grid on
c      the full surface of a sphere. choosing ixp =
jyq = 45 and iex = 4
c      and jyq = 3 fits the required 361 by 181 grid
exactly. multigrid
c      cycling will be used on the sequence of
subgrid sizes:
c
c      46 x 46 < 91 x 46 < 181 x 91 < 361 x
181
c
c      the 46 x 46 coarsest subgrid has too much
resolution for effective
c      error reduction with relaxation only. muh2cr
circumvents this
c      difficulty by generating an exact direct
solution (modulo roundoff
c      error) whenever the coarsest grid is
encountered.
c
c      (2) additional work space (see (1) under
"arguments differences") is
c      required by muh2 to implement gaussian
elimination at the coarsest
c      grid level. this may limit the size of ixp
and jyq.
c
c      (3) operation counts
c
c      for simplicity, assume p = ixp = jyq and n =
nx = ny. banded
c      gaussian elimination requires  $\mathcal{O}(p^{**4})$ 
operations for solution
c      on the coarsest subgrid while multigrid
iteration is a  $\mathcal{O}(n^{**2})$ 
c      algorithm. these are approximately balanced
when
c
c       $p^{**4} =: (n / (2^{**k}))^{**4} =: n^{**2}$ 
c
c      or
c
c       $k =: \log_2(n) / 2$ 

```

```
c
c      grid levels are chosen with the hybrid method.
so if
c      p is approximately equal to
c
c      n/ (2** (log2 (n) /2) )
c
c      then the direct method and multigrid parts of
the hybrid algorithm
c      require roughly the same amount of computer
time. larger values
c      for p mean the direct method will dominate the
computation. smaller
c      values mean the hybrid method will cost only
marginally more than
c      multigrid iteration with coarse grid
relaxation.
c
c
c *** the remaining documentation is almost identical to
mud2cr.d
c      except for the modifications already indicated.
c
c
c ... mesh description . . .
c
c      the approximation is generated on a uniform nx by
ny grid. the grid
c      is superimposed on the rectangular solution region
c
c      [xa,xb] x [yc,yd].
c
c      let
c
c      dlx = (xb-xa) / (nx-1), dly = (yd-yc) / (ny-1)
c
c      be the uniform grid increments in the x,y
directions. then
c
c      xi=xa+(i-1)*dlx,   yj=yc+(j-1)*dly
c
c      for i=1,...,nx and j=1,...,ny denote the x,y
uniform mesh points
c
```

```
c
c ... language
c
c      fortran90/fortran77
c
c
c ... portability
c
c      mudpack5.0.1 software has been compiled and tested
with fortran77
c      and fortran90 on a variety of platforms.
c
c ... methods
c
c      details of the methods employed by the solvers in
mudpack are given
c      in [1,9]. [1,2,9] contain performance
measurements on a variety of
c      elliptic pdes (see "references" in the file
"readme"). in summary:
c
c *** discretization and solution (second-order solvers)
(see [1])
c
c      the pde and boundary conditions are automatically
discretized at all
c      grid levels using second-order finite difference
formula. diagonal
c      dominance at coarser grid levels is maintained in
the presence of
c      nonzero first-order terms by adjusting the second-
order coefficient
c      when necessary. the resulting block tri-diagonal
linear system is
c      approximated using multigrid iteration
[10,11,13,15,16,18]. version
c      5.0.1 of mudpack uses only fully weighted residual
restriction. defaults
c      include cubic prolongation and w(2,1) cycles.
these can be overridden
c      with selected multigrid options (see "mgopt").
error control based on
c      maximum relative differences is available. full
multigrid cycling (fmg)
```

```
c      or cycling beginning or restarting at the finest
grid level can be
c      selected. a menu of relaxation methods including
gauss-seidel point,
c      line relaxation(s) (in any combination of
directions) and planar
c      relaxation (for three-dimensional anisotropic
problems) are provided.
c      all methods use ordering based on alternating
points (red/black),
c      lines, or planes for cray vectorization and
improved convergence
c      rates [14].
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
estimate (i.e.,
c      discretization level error is reached) then this
can be improved to a
c      fourth-order estimate using the technique of
"deferred corrections."
c      the values in the solution array are used to
generate a fourth-order
c      approximation to the truncation error. second-
order finite difference
c      formula are used to approximate third and fourth
partial derivatives
c      of the solution function [3]. the truncation
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side. the default
multigrid options
c      are used to compute the fourth-order correction
term which is added
c      to the original solution array.
c
c
c ... references (partial)
c
c
c [1] J. Adams, "MUDPACK: Multigrid Fortran Software
```

```
for the Efficient
c      Solution of Linear Elliptic Partial Differential
Equations,"
c      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
c
c      [2] J. Adams,"FMG Results with the Multigrid
Software Package MUDPACK,"
c      proceedings of the fourth Copper Mountain
Conference on Multigrid, SIAM,
c      1989, pp.1-12.
c      .
c      .
c      .
c      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.
Haidvogel, and V. Pizzo,
c      "Applications of Multigrid Software in the
Atmospheric Sciences,"
c      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-
1458.
c      .
c      .
c      .
c      [9] J. Adams, "Recent Enhancements in MUDPACK, a
Multigrid Software
c      package for Elliptic Partial Differential
Equations," Applied Math.
c      and Comp., 1991, vol. 43, May 1991, pp. 79-94.
c
c      [10]J. Adams, "MUDPACK-2: Multigrid Software for
Approximating
c      Elliptic Partial Differential Equations on Uniform
Grids with
c      any Resolution," Applied Math. and Comp., 1993,
vol. 53, February
c      1993, pp. 235-249
c      .
c      .
c      .
c
c ... argument description
c
c
```

```
*****
*****  
C *** input arguments  
*****  
C  
*****  
C  
C ... iparm  
C  
C      an integer vector of length 17 used to pass  
integer  
C      arguments. iparm is set internally and  
defined as  
C      follows:  
C  
C  
C ... intl=iparm(1)  
C  
C      an initialization argument. intl=0 must be  
input  
C      on an initial call. in this case input  
arguments will  
C      be checked for errors and the elliptic  
partial differential  
C      equation and boundary conditions will be  
discretized using  
C      second order finite difference formula.  
C  
C ***      an approximation is not generated after an  
intl=0 call!  
C      muh2cr should be called with intl=1 to  
approximate the elliptic  
C      pde discretized by the intl=0 call. intl=1  
should also  
C      be input if muh2cr has been called earlier  
and only the  
C      values in in rhs (see below) or gbdy (see  
bndyc below)  
C      or phi (see below) have changed. this will  
bypass  
C      redundant pde discretization and argument  
checking
```

```
c           and save computational time. some examples
of when
c           intl=1 calls should be used are:
c
c           (0) after a intl=0 argument checking and
discretization call
c
c           (1) muh2cr is being recalled for additional
accuracy. in
c           this case iguess=iparm(12)=1 should also
be used.
c
c           (2) muh2cr is being called every time step in
a time dependent
c           problem (see discussion below) where the
elliptic operator
c           does not depend on time.
c
c           (3) muh2cr is being used to solve the same
elliptic equation
c           for several different right hand sides
(igueess=0 should
c           probably be used for each new righthand
side).
c
c           intl = 0 must be input before calling with
intl = 1 when any
c           of the following conditions hold:
c
c           (a) the initial call to muh2cr
c
c           (b) any of the integer arguments other than
igueess=iparm(12)
c           or maxcy=iparm(13) or mgopt have changed
since the previous
c           call.
c
c           (c) any of the floating point arguments other
than tolmax=
c           fparm(5) have changed since the previous
call
c
c           (d) any of the coefficients input by coef
(see below) have
```

```
c           changed since the previous call
c
c           (e) any of the "alfa" coefficients input by
bndyc (see below)
c           have changed since the previous call.
c
c           if any of (a) through (e) are true then the
elliptic pde
c           must be discretized or rediscretized.  if
none of (a)
c           through (e) holds, calls can be made with
intl=1.
c           incorrect calls with intl=1 will produce
erroneous results.
c ***      the values set in the saved work space "work"
(see below) with
c           an intl=0 call must be preserved with
subsequent intl=1 calls.
c
c           MUDPACK software performance should be
monitored for intl=1
c           calls.  The intl=0 discretization call
performance depends
c           primarily on the efficiency or lack of
efficiency of the
c           user provided subroutines for pde
coefficients and
c           boundary conditions.

c
c
c ... nxa=iparm(2)
c
c           flags boundary conditions on the edge x=xa
c
c           = 0 if p(x,y) is periodic in x on [xa,xb]
c           (i.e., p(x+xb-xa,y) = p(x,y) for all x,y
c           (if nxa=0 then nxb=0 is required, see
ierror = 2)
c
c           = 1 if p(xa,y) is specified (this must be input
thru phi(1,j))
c
c           = 2 if there are mixed derivative boundary
conditions at x=xa
```

```

C           (see bndyc)
C
C
C ... nxb=iparm(3)
C
C           flags boundary conditions on the edge x=xb
C
C           = 0 if p(x,y) is periodic in x on [xa,xb]
C           (i.e., p(x+xb-xa,y) = p(x,y) for all x,y)
C           (if nxb=0 then nxa=0 is required, see
ierror = 2)
C
C           = 1 if p(xb,y) is specified (this must be input
thru phi(nx,j))
C
C           = 2 if there are mixed derivative boundary
conditions at x=xb
C           (see bndyc)
C
C
C ... nyc=iparm(4)
C
C           flags boundary conditions on the edge y=yc
C
C           = 0 if p(x,y) is periodic in y on [yc,yd]
C           (i.e., p(x,y+yd-yc) = p(x,y) for all x,y)
C           (if nyc=0 then nyd=0 is required, see
ierror = 2)
C
C           = 1 if p(x,yc) is specified (this must be input
thru phi(i,1))
C
C           = 2 if there are mixed derivative boundary
conditions at y=yc
C           (see bndyc)
C
C
C ... nyd=iparm(5)
C
C           flags boundary conditions on the edge y=yd
C
C           = 0 if p(x,y) is periodic in y on [yc,yd]
C           (i.e., p(x,y+yd-yc) = p(x,y) for all x,y)
C           (if nyd=0 then nyc=0 is required, see

```

```
ierror = 2)
c
c      = 1 if p(x,yd) is specified (this must be input
thru phi(i,ny))
c
c      = 2 if there are mixed derivative boundary
conditions at y=yd
c          (see bndyc)
c
c
c *** grid size arguments
c
c
c ... ixp = iparm(6)
c
c      an integer greater than one which is used in
defining the number
c          of grid points in the x direction (see nx =
iparm(10)). "ixp+1"
c          is the number of points on the coarsest x
grid visited during
c          multigrid cycling. ixp should be chosen as
small as possible.
c          recommended values are the small primes 2 or
3.
c          larger values can reduce multigrid
convergence rates considerably,
c          especially if line relaxation in the x
direction is not used.
c          if ixp > 2 then it should be 2 or a small odd
value since a power
c          of 2 factor of ixp can be removed by
increasing iex = iparm(8)
c          without changing nx = iparm(10).
c
c
c ... jyq = iparm(7)
c
c      an integer greater than one which is used in
defining the number
c          of grid points in the y direction (see ny =
iparm(11)). "jyq+1"
c          is the number of points on the coarsest y
grid visited during
```

```

c      multigrid cycling.  jyq should be chosen as
small as possible.
c      recommended values are the small primes 2 or
3.
c      larger values can reduce multigrid
convergence rates considerably,
c      especially if line relaxation in the y
direction is not used.
c      if jyq > 2 then it should be 2 or a small odd
value since a power
c      of 2 factor of jyq can be removed by
increasing jey = iparm(9)
c      without changing ny = iparm(11).
c
c
c ... iex = iparm(8)
c
c      a positive integer exponent of 2 used in
defining the number
c      of grid points in the x direction (see nx =
iparm(10)).
c      iex .le. 50 is required.  for efficient
multigrid cycling,
c      iex should be chosen as large as possible and
ixp=iparm(8)
c      as small as possible within grid size
constraints when
c      defining nx.
c
c
c ... jey = iparm(9)
c
c      a positive integer exponent of 2 used in
defining the number
c      of grid points in the y direction (see ny =
iparm(11)).
c      jey .le. 50 is required.  for efficient
multigrid cycling,
c      jey should be chosen as large as possible and
jyq=iparm(7)
c      as small as possible within grid size
constraints when
c      defining ny.
c

```

```
c
c
c ... nx = iparm(10)
c
c           the number of equally spaced grid points in
the interval [xa,xb]
c           (including the boundaries).  nx must have the
form
c
c           nx = ixp*(2** (iex-1)) + 1
c
c           where ixp = iparm(6), iex = iparm(8).
c
c
c ... ny = iparm(11)
c
c           the number of equally spaced grid points in
the interval [yc,yd]
c           (including the boundaries).  ny must have the
form:
c
c           ny = jyq*(2** (jey-1)) + 1
c
c           where jyq = iparm(7), jey = iparm(9).
c
c
c *** example
c
c           suppose a solution is wanted on a 33 by 97
grid.  then
c           ixp=2, jyq=6 and iex=jey=5 could be used.  a
better
c           choice would be ixp=2, jyq=3, and iex=5,
jey=6.
c
c *** grid size flexibility considerations:
c
c           the hybrid multigrid/direct method code muh2cr
provides more grid size
c           flexibility than muh2cr by removing the constraint
that ixp and jyq are
c           2 or 3.  this is accomplished by using a direct
method whenever the
c           coarsest (ixp+1) x (jyq+1) grid is encountered in
```

```

multigrid cycling.

c      if nx = ixp+1 and ny = jyq+1 then muh2cr becomes a
full direct method.

c      muh2cr is roughly equivalent to muh2cr in
efficiency as long as ixp and
c      jyq remain "small" (see muh2cr.d).  if the problem
to be approximated
c      requires a grid neither muh2cr or muh2cr can
exactly fit then another option
c      is to generate an approximation on a "close grid"
using mud2cr or muh2cr.

c      then transfer the result to the required grid
using cubic interpolation
c      via the package "regridpack" (contact john adams
about this software)

c
c *** note
c
c      let G be the nx by ny fine grid on which the
approximation is
c      generated and let n = max0(iex,jey).  in mudpack,
multigrid
c      cycling is implemented on the ascending chain of
grids
c
c          G(1) < ... < G(k) < ... < G(n) = G.
c
c      each G(k) (k=1,...,n) has mx(k) by my(k) grid
points
c      given by:
c
c          mx(k) = ixp*[2** (max0(iex+k-n,1)-1)] + 1
c
c          my(k) = jyq*[2** (max0(jey+k-n,1)-1)] + 1
c
c
c
c ... iguess=iparm(12)
c
c          = 0 if no initial guess to the pde is
provided
c
c          = 1 if an initial guess to the pde is at the
finest grid

```

```
c           level is provided in phi (see below)
c
c   comments on iguess = 0 or 1 . . .
c
c   even if iguess = 0, phi must be initialized at all
grid points (this
c   is not checked). phi can be set to 0.0 at non-
dirchlet grid points
c   if nothing better is available. the values set in
phi when iguess = 0
c   are passed down and serve as an initial guess to
the pde at the coarsest
c   grid level where cycling commences. in this
sense, values input in
c   phi always serve as an initial guess. setting
iguess = 0 forces full
c   multigrid cycling beginning at the coarsest and
finishing at the finest
c   grid level.
c
c   if iguess = 1 then the values input in phi are an
initial guess to the
c   pde at the finest grid level where cycling begins.
this option should
c   be used only if a "very good" initial guess is
available (as, for
c   example, when restarting from a previous iguess=0
call).
c
c   time dependent problems . . .
c
c *** assume we are solving an elliptic pde every time
step in a
c   marching problem of the form:
c
c   l(p(t)) = r(t)
c
c   where the differential operator "l" has no time
dependence,
c   "p(t)" is the solution and "r(t)" is the right
hand side at
c   current time "t". let "dt" be the increment
between time steps.
c   then p(t) can be used as an initial guess to
```

```

p(t+dt) with
c      intl = 1 when solving
c
c      l(p(t+dt)) = r(t+dt).
c
c      after the first two time steps, rather than
continue, it would
c      be better to define the "correction" term:
c
c      e(t,dt) = p(t+dt) - p(t)
c
c      this clearly satisfies the equation
c
c      l(e(t,dt)) = r(t+dt) - r(t).
c
c      this should be solved with iguess = 0 and intl =
1. boundary
c      conditions for e(t,dt) are obtained from the
boundary conditions
c      for p(t) by subtracting given values at t from
given values at
c      t+dt. for example if
c
c      d(p(t))/dx = f(t), d(p(t+dt))/dx = f(t+dt)
c
c      at some x boundary then e(t,dt) satisfies the
derivative
c      boundary condition
c
c      d(e(t,dt))/dx = f(t+dt) - f(t).
c
c      e(t,dt) can be preset to 0.0 (at nondirichlet
points) or (if p(t-dt)
c      is saved) to p(t)-p(t-dt). with iguess = 0, these
values will serve
c      as an initial guess to e(t,dt) at the coarsest
grid level. this
c      approach has the advantage that a full sequence of
multigrid cycles,
c      beginning at the coarsest grid level, is invoked
every time step in
c      solving for e(t,dt). a few digits of accuracy in
e(t,dt), which is
c      ordinarily much smaller than p(t), will yield

```

```

several more digits of
c      accuracy in the final approximation:
c
c      p(t+dt) = p(t) + e(t,dt).
c
c      using this approach to integrate in time will give
more accuracy
c      then using p(t) as an initial guess to p(t+dt) for
all time steps.
c      it does require additional storage.
c
c      if the differential operator "l" has time
dependence (either thru
c      the coefficients in the pde or the coefficients in
the derivative
c      boundary conditions) then use p(t) as an initial
guess to p(t+dt)
c      when solving
c
c      l(t+dt) (p(t+dt)) = r(t+dt)
c
c      with intl = 0 for all time steps (the
discretization must be repeated
c      for each new "t"). either iguess = 0 (p(t) will
then be an initial
c      guess at the coarsest grid level where cycles will
commence) or
c      iguess = 1 (p(t) will then be an initial guess at
the finest grid
c      level where cycles will remain fixed) can be
tried.
c
c
c ... maxcy = iparm(13)
c
c      the exact number of cycles executed between
the finest (nx by
c      ny) and the coarsest ((ixp+1) by (jyq+1))
grid levels when
c      tolmax=fparm(5)=0.0 (no error control). when
tolmax > 0.0
c      is input (error control) then maxcy is a
limit on the number
c      of cycles between the finest and coarsest

```

```
grid levels. in
c           any case, at most maxcy*(iprert+ipost)
relaxation sweeps are
c           are performed at the finest grid level (see
iprert=mgopt(2),
c           ipost=mgopt(3) below). when multigrid
iteration is working
c           "correctly" only a few are required for
convergence. large
c           values for maxcy should not be necessary.
c
c
c ... method = iparm(14) determines the method of
relaxation
c           (gauss-seidel based on alternating points
or lines)
c
c           = 0 for point relaxation
c
c           = 1 for line relaxation in the x direction
c
c           = 2 for line relaxation in the y direction
c
c           = 3 for line relaxation in both the x and y
direction
c
c
c *** choice of method. . .
c
c     let fx represent the quantity cxx(x,y)/dlx**2 over
the solution region.
c
c     let fy represent the quantity cyy(x,y)/dly**2 over
the solution region
c
c     if fx,fy are roughly the same size and do not vary
too much over
c     the solution region choose method = 0. if this
fails try method=3.
c
c     if fx is much greater than fy choose method = 1.
c
c     if fy is much greater than fx choose method = 2
c
```

```

c      if neither fx or fy dominates over the solution
region and they
c      both vary considerably choose method = 3.
c
c
c ... length = iparm(15)
c
c      the length of the work space provided in
vector work (see below).
c      let isx = 0 if method = 0 or method = 2
c      let isx = 3 if method = 1 or method = 3 and
nxa.ne.0
c      let isx = 5 if method = 1 or method = 3 and
nxa.eq.0
c      let jsy = 0 if method = 0 or method = 1
c      let jsy = 3 if method = 2 or method = 3 and
nyc.ne.0
c      let jsy = 5 if method = 2 or method = 3 and
nyc.eq.0
c      let
c
c      len =
4* [ (nx+2) * (ny+2)+(10+isx+jsy) *nx*ny]/3+(nx+2) * (ny+2)
c
c      let (space for coarse grid direct method)
c
c      ldir = (ixp+1)*(jyq+1)*(2*ixp+3) (if
nyc.ne.0) or
c
c      ldir = (ixp+1)*[2*(ixp+1)*(2*jyq-1)+jyq+1]
(if nyc=0)
c
c      then
c
c      length = len + ldir
c
c      will suffice in most cases. the exact
minimal work space
c      length required for the current nx,ny and
method is output
c      in iparm(16) (even if iparm(15) is too
small). this will be
c      less then the value given by the simplified
formula above

```

```
c           in most cases.  
c  
c  
c ... fparm  
c  
c           a floating point vector of length 6 used to  
efficiently  
c           pass floating point arguments.  fparm is set  
internally  
c           in muh2cr and defined as follows . . .  
c  
c  
c ... xa=fparm(1), xb=fparm(2)  
c  
c           the range of the x independent variable. xa  
must  
c           be less than xb  
c  
c  
c ... yc=fparm(3), yd=fparm(4)  
c  
c           the range of the y independent variable. yc  
must  
c           be less than yd.  
c  
c  
c ... tolmax = fparm(5)  
c  
c           when input positive, tolmax is a maximum  
relative error tolerance  
c           used to terminate the relaxation iterations.  
assume phil(i,j)  
c           and phi2(i,j) are the last two computed  
approximations at all  
c           grid points of the finest grid level. if we  
define  
c  
c           phdif = max(abs(phi2(i,j)-phil(i,j))) for  
all i,j  
c  
c           and  
c  
c           phmax = max(abs(phi2(i,j))) for all i,j  
c
```

```
c           then "convergence" is considered to have
occurred if and only if
c
c           phdif/phmax < tolmax.
c
c
c           if tolmax=fparm(5)=0.0 is input then there is
no error control
c           and maxcy cycles from the finest grid level
are executed. maxcy
c           is a limit which cannot be exceeded even with
error control.
c           *** calls with tolmax=0.0, when appropriate
because of known
c           convergence behavior, are more efficient than
calls with tolmax
c           positive (i.e., if possible do not use error
control!).
c
c ... wk
c
c           a one dimensional real saved work space (see
iparm(15) for
c           length) which must be preserved from the
previous call when
c           calling with intl=iparm(1)=1.
c
c ... iwk
c
c           an integer vector dimensioned of length at
least (ixp+1)*(jyq+1)
c           (ixp = iparm(6),jyq=iparm(7)) in the routine
calling muh2cr.
c           the length of iwk is not checked! if iwk has
length less than
c           (ixp+1)*(jyq+1) then undetectable errors will
result.
c
c ... bndyc
c
c           a subroutine with arguments
(kbdy,xory,alfa,beta,gama,gbdy) which
c           are used to input mixed boundary conditions
to muh2cr. bndyc
```

```

c           must be declared "external" in the program
calling muh2cr.
c           the boundaries are numbered one thru four and
the mixed
c           derivative boundary conditions are described
below (see the
c           sample driver code "tmuh2cr.f" for an example
of how bndyc is
c           can beset up) .
c
c           * * * * * * * * * * * * * * * * y=yd
c           *           kbdy=4           *
c           *
c           *
c           *
c           *
c           *   kbdy=1           kbdy=2   *
c           *
c           *
c           *
c           *
c           *           kbdy=3           *
c           * * * * * * * * * * * * * * * * y=yc
c
c           x=x'a           x=x'b
c
c
c
c           (1)   the kbdy=1 boundary
c
c           this is the edge x=x'a where nx'a=iparm(2) = 2
flags
c           a mixed boundary condition of the form
c
c           alfxa(y)*px + betxa(y)*py +
gamxa(y)*p(xa,y) = gbdxa(y)
c
c           in this case kbdy=1,xory=y will be input to
bndyc and
c           alfa,beta,gama,gbdy corresponding to
alfxa(y),betxa(y),gamxa(y),
c           gbdxa(y) must be returned.  alfxa(y) = 0. is
not allowed for any y.
c           (see ierror = 13)
c
c           (2)   the kbdy=2 boundary

```

```

c
c      this is the edge x=xb where nxb=iparm(3) = 2
flags
c      a mixed boundary condition of the form
c
c          alfxb(y)*px + betxb(y)*py +
gamxb(y)*p(xb,y) = gbdxb(y)
c
c      in this case kbdy=2,xory=y will be input to
bndyc and
c          alfa,beta,gama,gbdy corresponding to
alfxb(y),betxb(y),gamxb(y),
c          gbdxb(y) must be returned. alfxb(y) = 0.0 is
not allowed for any y.
c          (see ierror = 13)
c
c      (3)    the kbdy=3 boundary
c
c      this is the edge y=yc where nyc=iparm(4) = 2
flags
c      a mixed boundary condition of the form
c
c          alfyc(x)*px + betyc(x)*py +
gamyc(x)*p(x,yc) = gbdyc(x)
c
c      in this case kbdy=3,xory=x will be input to
bndyc and
c          alfa,beta,gama,gbdy corresponding to
alfyc(x),betyc(x),gamyc(x),
c          gbdyc(x) must be returned. betyc(x) = 0.0 is
not allowed for any x.
c          (see ierror = 13)
c
c      (4)    the kbdy=4 boundary
c
c      this is the edge y=yd where nyd=iparm(5) = 2
flags
c      a mixed boundary condition of the form
c
c          alfyd(x)*px + betyd(x)*py +
gamyd(x)*p(x,yd) = gbdyd(x)
c
c      in this case kbdy=4,xory=x will be input to
bndyc and

```

```
c           alfa,beta,gama,gbdy corresponding to
alfa,yd(x),beta,yd(x),gamma,yd(x),
c           gbdy,yd(x) must be returned. bety,yd(x) = 0.0 is
not allowed for any x.
c           (see ierror = 13)
c
c
c ***      bndyc must provide the mixed boundary
condition values
c           in correspondence with those flagged in
iparm(2) thru
c           iparm(5). if all boundaries are specified or
periodic
c           mud2cr will never call bndyc. even then it
must be entered
c           as a dummy subroutine. bndyc must be declared
"external"
c           in the routine calling mud2cr. the actual
name chosen may
c           be different.
c
c
c ... coef
c
c           a subroutine with arguments
(x,y,cxx,cxy,cyy,cx,cy,ce) which
c           provides the known real coefficients for the
elliptic pde at
c           any grid point (x,y). the name chosen in the
calling routine
c           may be different where the coefficient routine
must be declared
c           "external."
c
c ... rhs
c
c           an array dimensioned nx by ny which contains
the given
c           right hand side values on the uniform 2-d
mesh.
c
c           rhs(i,j) = r(xi,yj) for i=1,...,nx and
j=1,...,ny
c
```

```
c ... phi
c
c           an array dimensioned nx by ny.  on input phi
must contain
c           specified boundary values.  for example, if
nyd=iparm(5)=1
c           then phi(i,ny) must be set equal to p(xi,yd)
for i=1,...nx
c           prior to calling muh2cr.  these values are
preserved by muh2cr.
c           if an initial guess is provided
(iguess=iparm(11)=1) it must
c           be input thru phi.
c
c
c ***      if no initial guess is given (iguess=0) then
phi must still
c           be initialized at all grid points (this is
not checked).  these
c           values will serve as an initial guess to the
pde at the coarsest
c           grid level after a transfer from the fine
solution grid.  set phi
c           equal to to 0.0 at all internal and non-
specified boundaries
c           grid points if nothing better is available.
c
c
c ... mgopt
c
c           an integer vector of length 4 which allows
the user to select
c           among various multigrid options.  if
mgopt(1)=0 is input then
c           a default set of multigrid arguments (chosen
for robustness)
c           will be internally selected and the
remaining values in mgopt
c           will be ignored.  if mgopt(1) is nonzero
then the arguments
c           in mgopt are set internally and defined as
follows: (see the
c           basic coarse grid correction algorithm
below)
```

```
C
C
C      kcycle = mgopt(1)
C
C          = 0 if default multigrid options are to be
used
C
C          = 1 if v cycling is to be used (the least
expensive per cycle)
C
C          = 2 if w cycling is to be used (the
default)
C
C          > 2 if more general k cycling is to be used
C          *** warning--values larger than 2 increase
C          the execution time per cycle
considerably and
C          result in the nonfatal error ierror =
-5
C          which indicates inefficient multigrid
cycling.
C
C      iprер = mgopt(2)
C
C          the number of "pre-relaxation" sweeps
executed before the
C          residual is restricted and cycling is
invoked at the next
C          coarser grid level (default value is 2
whenever mgopt(1)=0)
C
C      ipost = mgopt(3)
C
C          the number of "post relaxation" sweeps
executed after cycling
C          has been invoked at the next coarser grid
level and the residual
C          correction has been transferred back
(default value is 1
C          whenever mgopt(1)=0).
C
C *** if iprер, ipost, or (especially) kcycle is greater
than 2
C      than inefficient multigrid cycling has probably
```

```
been chosen and
c      the nonfatal error (see below) ierror = -5 will be
set. note
c      this warning may be overridden by any other
nonzero value
c      for ierror.
c
c      intpol = mgopt(4)
c
c          = 1 if multilinear prolongation
c (interpolation) is used to
c          transfer residual corrections and the pde
approximation
c          from coarse to fine grids within full
multigrid cycling.
c
c          = 3 if multicubic prolongation
c (interpolation) is used to
c          transfer residual corrections and the pde
approximation
c          from coarse to fine grids within full
multigrid cycling.
c          (this is the default value whenever
mgopt(1)=0).
c
c *** the default values (2,2,1,3) in the vector mgopt
were chosen for
c      robustness. in some cases v(2,1) cycles with
linear prolongation will
c      give good results with less computation
(especially in two-dimensions).
c      this was the default and only choice in an
earlier version of mudpack
c      (see [1]) and can be set with the integer vector
(1,2,1,1) in mgopt.
c
c *** the schedules for one full multigrid cycle
(iguess=0) using v(2,1)
c      cycles and w(2,1) cycles are depicted for a four
level grid below.
c      the number of relaxation sweeps when each grid is
visited are indicated.
c      the "*" stands for prolongation of the full
approximation and the ".*"
```

```

c      stands for transfer of residuals and residual
corrections within the
c      coarse grid correction algorithm (see below). all
version 5.0.1
c      mudpack solvers use only fully weighted residual
restriction. the
c      "d" at grid level 1 indicates a direct method
(banded gaussian
c      elimination replaces relaxation).

c
c      one fmg with v(2,1) cycles:
c
c
c      -----
c      -----2-----1-----1-
c      ----- level 4
c
c      *
c      .
c
c      -----
c      -----2-----1-----2-----1-----1-
c      ----- level 3
c
c      *
c      .
c
c      -----
c      -----2-----1-----2-----1-----2-----1-----
c      ----- level 2
c
c      *
c      .
c
c      -----
c      ---d---d-----d-----d-----
c      ----- level 1
c
c
c      one fmg with w(2,1) cycles:
c
c
c      -----
c      -----2-----
c      --1-- level 4
c
c      *
c      .
c
c      -----
c      -----2-----1-----2-----3-----
c      1---- level 3
c
c      *
c      .
c
c      -----
c      -----2-----1-----2-----3-----1-----2-----3-----1-
c      ----- level 2
c
c      *
c      .
c
c      -----
c      ---d---d-----d-----d-----d-----d-----d-----d-----
c      ----- level 1
c

```

```

c
c      the form of the "recursive" coarse grid correction
cycling used
c      when kcycle.ge.0 is input is described below in
pseudo-algorithmic
c      language. it is implemented non-recursively in
fortran in mudpack.
c *** this algorithm is modified with the hybrid
solvers which use
c      a direct method whenever grid level 1 is
encountered.

c
c      algorithm
cgc(k,l(k),u(k),r(k),kcycle,iprer,ipost,iressw,intpol)
c
c *** approximately solve l(k)*u(k) = r(k) using
multigrid iteration
c *** k is the current grid level
c *** l(k) is the discretized pde operator at level k
c *** u(k) is the initial guess at level k
c *** r(k) is the right hand side at level k
c *** i(k,k-1) is the restriction operator from level k
to level k-1
c *** (the form of i(k,k-1) depends on iressw)
c *** i(k-1,k) is the prolongation operator from level
k-1 to level k
c *** (the form of i(k-1,k) depends on intpol)
c
c      begin algorithm cgc
c
c ***     pre-relax at level k
c
c      . do (i=1,iprer)
c
c      . . relax(l(k),u(k),r(k))
c
c      . end do
c
c      . if (k > 1) then
c
c ***      restrict the residual from level k to level k-
1
c
c      . . r(k-1) = i(k,k-1) (r(k)-l(k)*u(k))

```

```

C
C      . . kount = 0
C
C      . . repeat
C
C ***      solve for the residual correction at level k-1
in u(k-1)
C ***      using algorithm cgc "kcycle" times (this is
the recursion)
C
C      . . . kount = kount+1
C
C      . . . invoke cgc(k-1,l(k-1),u(k-1),r(k-
1),kcycle,iprер,ipost,iresw)
C
C
C      . . until (kount.eq.kcycle)
C
C ***      transfer residual correction in u(k-1) to
level k
C ***      with the prolongation operator and add to u(k)
C
C      . . u(k) = u(k) + i(k-1,k) (u(k-1))
C
C      . end if
C
C ***      post relax at level k
C
C      . do (i=1,ipost)
C
C      . . relax(l(k),u(k),r(k))
C
C      . end do
C
C      . return
C
C      end algorithm cgc
C
C
C
*****
*****  

C *** output arguments
*****

```

```

C
*****
*****  

C
C
C ... iparm(16) *** set for intl=0 calls only
C
C           on output iparm(16) contains the actual work
space length
C           required. this will usually be less than
that given by the
C           simplified formula for length=iparm(15) (see
as input argument)
C
C
C ... iparm(17) *** set for intl=1 calls only
C
C           on output iparm(17) contains the actual
number of multigrid cycles
C           between the finest and coarsest grid levels
used to obtain the
C           approximation when error control (tolmax >
0.0) is set.
C
C
C ... fparm(6) *** set for intl=1 calls with fparm(5)
> 0. only
C
C           on output fparm(6) contains the final
computed maximum relative
C           difference between the last two iterates at
the finest grid level.
C           fparm(6) is computed only if there is error
control (tolmax > 0.0)
C           assume phi1(i,j,k) and phi2(i,j,k) are the
last two computed
C           values for phi(i,j,k) at all points of the
finest grid level.
C           if we define
C
C           phdif = max (abs(phi2(i,j)-phi1(i,j)))
over all i,j
C
C           and

```

```

C
C           phmax = max (abs(phi2(i,j)) over all i,j
C
C           then
C
C           fparm(6) = phdif/phmax
C
C           is returned whenever phmax > 0.0. in the
degenerate case
C           phmax = 0.0, fparm(6) = phdif is returned.
C
C
C ... work
C
C           on output work contains intermediate values
that must not
C           be destroyed if muh2cr is to be called again
with intl=1
C
C
C ... phi   *** for intl=1 calls only
C
C           on output phi(i,j) contains the approximation
to p(xi,yj)
C           for all mesh points i = 1,...,nx and
j=1,...,ny. the last
C           computed iterate in phi is returned even if
convergence is
C           not obtained
C
C
C ... ierror
C
C           for intl=iparm(1)=0 initialization calls,
ierror is an
C           error flag that indicates invalid input
arguments when
C           returned positive and nonfatal warnings when
returned
C           negative. argument checking and
discretization
C           is bypassed for intl=1 calls which can only
return
C           ierror = -1 or 0 or 1.

```

```

c
c
c      non-fatal warnings * * *
c
c
c      ==5 if kcycle=mgopt(1) is greater than 2. values
lager than 2 results
c          in an algorithm which probably does far more
computation than
c          necessary.  kcycle = 1 (v cycles) or kcycle=2
(w cycles) should
c          suffice for most problems.  ierror = -5 is
also set if either
c          iprер = mgopt(2) or ipost=mgopt(3) is greater
than 2.  the
c          ierror=-5 flag is overridden by any other
fatal or non-fatal
c          error.
c
c      ==4 if there are dominant nonzero first order
terms in the pde which
c          make it "hyperbolic" at the finest grid level.
numerically, this
c          happens if:
c
c          abs(cx)*dlx > 2.*abs(cxx)    (dlx = (xb-
xa) / (nx-1))
c
c                      (or)
c
c          abs(cy)*dly > 2.*abs(cyy)    (dly = (yd-
yc) / (ny-1))
c
c
c          at some fine grid point (xi,yj).  if an
adjustment is not made the
c          condition can lead to a matrix coming from the
discretization
c          which is not diagonally dominant and
divergence is possible. since
c          the condition is "likely" at coarser grid
levels for pde's with
c          nonzero first order terms, the adjustments
(actually first order

```

```

c      approximations to the pde)
c
c
c      cxx = amax1(cxx,0.5*abs(cx)*dx)
c
c              (and)
c
c      cyy = amax1(cyy,0.5*abs(cy)*dy)
c
c
c      (here dx,dy are the x,y mesh sizes of the
c      subgrid)
c
c      are made to preserve convergence of multigrid
c      iteration. if made
c          at the finest grid level, it can lead to
c      convergence to an
c          erroneous solution (flagged by ierror = -4).
c      a possible remedy
c          is to increase resolution. the ierror = -4
c      flag overrides the
c          nonfatal ierror = -5 flag.
c
c
c
c      ==3 if the continuous elliptic pde is singular.
c      this means the
c          boundary conditions are periodic or pure
c      derivative at all
c          boundaries and ce(x,y) = 0.0 for all x,y. a
c      solution is still
c          attempted but convergence may not occur due
c      to ill-conditioning
c          of the linear system coming from the
c      discretization. the
c          ierror = -3 flag overrides the ierror=-4 and
c      ierror=-5 nonfatal
c          flags.
c
c
c
c      ==2 if the pde is not elliptic (i.e.,
c      cxx*cyy.le.0.0 for some (xi,yj))
c          in this case a solution is still attempted
c      although convergence
c          may not occur due to ill-conditioning of the

```

```

linear system.
c           the ierror = -2 flag overrides the ierror=-
5,-4,-3 nonfatal
c           flags.
c
c
c     ==1 if convergence to the tolerance specified in
tolmax=iparm(5)>0.
c           is not obtained in maxcy=iparm(13) multigrid
cycles between the
c           coarsest (ixp+1,jyq+1) and finest (nx,ny)
grid levels.
c           in this case the last computed iterate is
still returned.
c           the ierror = -1 flag overrides all other
nonfatal flags
c
c
c     no errors * * *
c
c     = 0
c
c     fatal argument errors * * *
c
c     = 1 if intl=iparm(1) is not 0 on initial call or
not 0 or 1
c           on subsequent calls
c
c     = 2 if any of the boundary condition flags
nxa,nxb,nyc,nyd
c           in iparm(2),iparm(3),iparm(4),iparm(5) are not
0,1 or 2
c           or if nxa,nxb or nyc,nyd are not pairwise
zero.
c
c     = 3 if min0(ixp,jyq) < 2 (ixp = iparm(6), jyq =
iparm(7))
c
c     = 4 if min0(iex,jey) < 1 (iex = iparm(8), jey =
iparm(9)) or
c           if max0(iex,jey) > 50
c
c     = 5 if nx.ne.ixp*2** (iex-1)+1 or
ny.ne.jyq*2** (jey-1)+1

```

```

c      (nx = iparm(10), ny = iparm(11))
c
c      = 6 if iguess = iparm(12) is not equal to 0 or 1
c
c      = 7 if maxcy = iparm(13) < 1
c
c      = 8 if method = iparm(14) is not 0,1,2, or 3
c
c      = 9 if length = iparm(15) is too small (see
iparm(16) on output
c          for minimum required work space length)
c
c      =10 if xa >=xb or yc >=yd
c
c      (xa=fparm(1),xb=fparm(2),yc=fparm(3),yd=fparm(4))
c
c      =11 if tolmax = fparm(5) < 0.0
c
c      errors in setting multigrid options * * * (see
also ierror=-5)
c
c      =12 if kcycle = mgopt(1) < 0 or
c          if iprer = mgopt(1) < 1 or
c          if ipost = mgopt(3) < 1 or
c          if interpol = mgopt(4) is not 1 or 3
c
c      =13 if there is a pure tangential derivative along
a mixed derivative
c          boundary (e.g., nyd = 2 and betyd(x) = 0.0 for
some
c          grid point x along y = yd)
c
c      =14 if there is the "singular" condition described
below at a
c          corner which is the intersection of two
derivative boundaries.
c
c          (1) the corner (xa,yc) if nxa=nyc=2 and
c              alfxa(yc)*betyc(xa)-alfyc(xa)*betxa(yc) =
0.0.
c
c          (2) the corner (xa,yd) if nxa=nyd=2 and
c              alfxa(yd)*betyd(xa)-alfyd(xa)*betxa(yd) =
0.0.

```

MUH3

```
* * * * * *
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK version 5.0.1
*
C      *
*
C      *               A Fortran Package of Multigrid
*
C      *
*
C      *               Subroutines and Example Programs
*
C      *
*
C      *               for Solving Elliptic Partial Differential
Equations        *
C      *
*
C      *               by
*
C      *
*
C      *               John Adams
*
C      *
*
C      *               of
*
C      *
*
C      *               the National Center for Atmospheric
```

```
Research          *
C      *
*
C      *                      Boulder, Colorado (80307)
U.S.A.          *
C      *
*
C      *                      which is sponsored by
*
C      *
*
C      *                      the National Science Foundation
*
C      *
*
C      *      *      *      *      *      *      *      *      *      *      *
*      *      *      *      *      *      *      *      *      *      *      *
C
C ... file muh3.d
C
C      contains documentation for:
C      subroutine
muh3(iparm,fparm,wk,iw,coef,bndyc,rhs,phi,mgopt,ierror)
C      a sample fortran driver is file "tmuh3.f".
C
C ... required mudpack files
C
C      mudcom.f, mud3ln.f, mud3pn.f
C
C ... purpose
C
C      the "hybrid" multigrid/direct method code muh3
approximates the
C      same 3-d nonseparable elliptic pde as the mudpack
solver mud3.
C      the basic algorithm is modified by using block
banded gaussian
C      elimination in place of relaxation whenever the
coarsest subgrid is
C      encountered within multigrid cycling.  use of the
direct method at
C      the coarsest grid level gives muh3 at least two
advantages over mud3:
C
```

```
c      (1) improved convergence rates
c
c      the use of a direct method at the coarsest
grid level can
c      improve convergence rates at a small
additional computational
c      cost if the coarse grid parameters
ixp=iparm(8),jyq=iparm(9),
c      kzs=iparm(10) are small relative to the fine
grid parameters
c      nx=iparm(14),ny=iparm(15),nz=iparm(16). this
is especially true
c      in the presence of certain boundary
conditions. for example,
c      if all boundary conditions are neuman (pure
derivative) and/or
c      periodic then mud3 may fail to converge. muh3
should handle
c      these boundary conditions with the expected
multigrid efficiency
c      (see tmuh3.f). in all cases, muh3 should give
convergence
c      rates which equal or exceed mud3.
c
c
c      (2) more resolution choices
c
c      muh3 allows more grid size flexibility by
"relaxing" the
c      constraint on the coarse grid parameters that
ixp,jyq,kzs
c      be "very" small (2 or 3 for mud3) for
effective error
c      error reduction within multigrid cycling.
convergence
c      rates will not deteriorate with larger values
for ixp,
c      jyq,kzs.
c
c *** caution
c
c      because of the very large computational and
storage
c      requirements, the three-dimensional dimensional
```

```

direct
c      method costs can overwhelm the multigrid cycling
costs
c      if the coarsest grid is not small relative to the
finest
c      solution grid.  this is a user decision set by the
choice
c      of coarse and fine grid parameters (see iparm(8)
through
c      iparm(16) and iparm(21) descriptions)
c
c      subroutine muh3 automatically discretizes and
attempts to compute
c      the second order finite difference approximation
to a three-
c      dimensional linear nonseparable elliptic partial
differential
c      equation on a box.  the approximation is generated
on a uniform
c      grid covering the box (see mesh description
below).  boundary
c      conditions may be any combination of mixed,
specified (Dirchlet)
c      or periodic.  the form of the pde solved is . . .
c
c      cxx(x,y,z)*pxx + cyy(x,y,z)*pyy +
czz(z,y,z)*pzz +
c
c      cx(x,y,z)*px + cy(x,y,z)*py + cz(x,y,z)*pz +
c
c      ce(x,y,z)*p(x,y,z) = r(x,y,z)
c
c      here cxx,cyy,czz,cx,cy,cz,ce are the known real
coefficients
c      of the pde; pxx,pyy,pzz,px,py,pz are the second
and first
c      partial derivatives of the unknown solution
function p(x,y,z)
c      with respect to the independent variables x,y,z;
r(x,y,z) is
c      is the known real right hand side of the elliptic
pde.  cxx,cyy
c      and czz should be positive for all (x,y,z) in the
solution region.

```

```

c
c
c ... argument differences with mud3.f
c
c      the input and output arguments of muh3 are almost
c      identical to the
c      arguments of mud3 (see mud3.d) with the following
c      exceptions:
c
c      (1) let mx=ixp+1, my=jyq+1, mz=kzr+1 (the coarsest
c      grid
c          resolutions, ixp=iparm(8), jyq=iparm(9),
c          kzr=iparm(10))
c          then the work space vector "wk" requires
c
c          mx*my*mz* (2*mx*my+1))
c (nze.ne.0)
c
c          additional words of storage if periodic
c boundary conditions
c          are not flagged in the z direction or
c
c          mx*my* (mz* (4*mx*my+1))           (nze=0)
c
c          additional words of storage if periodic
c boundary conditions are
c          flagged in the z direction (nze = 0). the
c extra work space is
c          used for a direct solution with gaussian
c elimination whenever the
c          coarsest grid is encountered within multigrid
c cycling.
c
c      (2) an integer work space iwk of length at least
c      mx*my*mz words
c          must be provided. the length of iwk is not
c checked!
c
c      (3) kzr > 2 if nze=0, jyq > 2 if nyc=0, ixp > 2 if
c      nxe = 0 are
c          required (i.e., the coarsest grid must contain
c at least four
c          points in any direction with periodic boundary
c conditions,

```

```
c           see the expanded meaning of ierror=3).
c
c *** (4) it is no longer necessary that ixp,jyq,kzr be
2 or 3 for
c           effective error reduction with multigrid
iteration. there
c           is no reduction in convergence rates when
larger values for
c           ixp,jyq,kzr are used . this provides
additional flexibility
c           in choosing grid size. in many cases muh3
provides more
c           robust convergence than mud3. it can be used
in place of
c           mud3 for all nonsingular problems (see (5)
below).
c
c           (5) iguess = iparm(17) = 1 (flagging an initial
guess) or
c           maxcy = iparm(18) > 1 (setting more than one
multigrid
c           cycle) are not allowed if muh3 becomes a full
direct method
c           by choosing iex=jey=kez=1. this conflicting
combination
c           of input arguments for multigrid iteration and
a full
c           direct method set the fatal error flag
c
c           ierror = 13
c
c           iguess = 0 and maxcy = 1 are required when
muh3 becomes a
c           full direct method. ordinarily (see ***
caution above) this
c           should not happen except when testing with
very coarse resolution.
c
c
c           (6) if a "singular" pde is detected (see ierror=-3
description in
c           mud3.d, ce(x,y) = 0.0 for all x,y and the
boundary conditions
c           are a combination of periodic and/or pure
```

```

derivatives) then muh3
c           sets the fatal error flag
c
c           ierror = 14
c
c           the direct method utilized by muh3 would
likely cause a near
c           division by zero in the singular case.  mud3
can be tried for
c           singular problems.
c
c ... grid size considerations
c
c     (1) flexibility
c
c           muh3 should be used in place of mud3 whenever
grid size
c           requirements do not allow choosing ixp,jyq,kzr
to be 2 or 3.
c
c     (2) additional work space (see (1) under
"arguments differences") is
c           required by muh3 to implement gaussian
elimination at the coarsest
c           grid level.  this may limit the size of
ixp,jyq,kzr.
c
c     (3) operation counts
c
k
c           for simplicity, assume p=ixp=jyq=kzr and
n=nx=ny=nz=2 *p.
c           gaussian elimination requires  $\mathcal{O}(p^{**7})$ 
operations for solution
c           on the coarsest subgrid while multigrid
iteration is a  $\mathcal{O}(n^{**3})$ 
c           algorithm.  consequently the storage and
computational
c           requirements for the 3-d direct method will
dominate the
c           calculation if p is "large."  note that
 $\mathcal{O}(p^{**7}) =: \mathcal{O}(n^{**3})$ 
c           whenever k =: (4/3)*log2(p) grid levels are
used in cycling.

```

```
c           larger values mean the direct method will
dominate the
c           calculation. smaller values for k mean the
direct method
c           will only marginally add to the cost of
multigrid iteration
c           alone.
c
c *** the remaining documentation is almost identical to
mud3.d
c       except for the modifications already indicated.
c
c
c ... mesh description . . .
c
c       the approximation is generated on a uniform nx by
ny by nz grid.
c       the grid is superimposed on the rectangular
solution region
c
c       [xa,xb] x [yc,yd] x [ze,zf].
c
c       let
c
c       dlx = (xb-xa) / (nx-1), dly = (yd-yc) / (ny-1),
dlz = (zf-ze) / (nz-1)
c
c       be the uniform grid increments in the x,y,z
directions. then
c
c       xi=xa+(i-1)*dlx,   yj=yc+(j-1)*dly,  zk =
ze+(k-1)*dlz
c
c       for i=1,...,nx; j=1,...,ny; k=1,...,nz  denote the
x,y,z uniform
c       mesh points.
c
c
c ... language
c
c       fortran90/fortran77
c
c
c ... portability
```

```
c
c      mudpack5.0.1 software has been compiled and tested
with fortran77
c      and fortran90 on a variety of platforms.
c
c ... methods
c
c      details of the methods employed by the solvers in
mudpack are given
c      in [1,9]. [1,2,9] contain performance
measurements on a variety of
c      elliptic pdes (see "references" in the file
"readme"). in summary:
c
c *** discretization and solution (second-order solvers)
(see [1])
c
c      the pde and boundary conditions are automatically
discretized at all
c      grid levels using second-order finite difference
formula. diagonal
c      dominance at coarser grid levels is maintained in
the presence of
c      nonzero first-order terms by adjusting the second-
order coefficient
c      when necessary. the resulting block tri-diagonal
linear system is
c      approximated using multigrid iteration
[10,11,13,15,16,18]. version
c      5.0.1 of mudpack uses only fully weighted residual
restriction. defaults
c      include cubic prolongation and w(2,1) cycles.
these can be overridden
c      with selected multigrid options (see "mgopt").
error control based on
c      maximum relative differences is available. full
multigrid cycling (fmg)
c      or cycling beginning or restarting at the finest
grid level can be
c      selected. a menu of relaxation methods including
gauss-seidel point,
c      line relaxation(s) (in any combination of
directions) and planar
c      relaxation (for three-dimensional anisotropic
```

```
problems) are provided.  
c      all methods use ordering based on alternating  
points (red/black),  
c      lines, or planes for cray vectorization and  
improved convergence  
c      rates [14].  
c  
c *** higher order solution (fourth-order solvers) (see  
[9,19,21])  
c  
c      if the multigrid cycling results in a second-order  
estimate (i.e.,  
c      discretization level error is reached) then this  
can be improved to a  
c      fourth-order estimate using the technique of  
"deferred corrections."  
c      the values in the solution array are used to  
generate a fourth-order  
c      approximation to the truncation error. second-  
order finite difference  
c      formula are used to approximate third and fourth  
partial derivatives  
c      of the solution function [3]. the truncation  
error estimate is  
c      transferred down to all grid levels using weighted  
averaging where  
c      it serves as a new right hand side. the default  
multigrid options  
c      are used to compute the fourth-order correction  
term which is added  
c      to the original solution array.  
c  
c  
c ... references (partial)  
c  
c  
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software  
for the Efficient  
c      Solution of Linear Elliptic Partial Differential  
Equations,"  
c      Applied Math. and Comput. vol.34, Nov 1989,  
pp.113-146.  
c  
c      [2] J. Adams, "FMG Results with the Multigrid
```

```
Software Package MUDPACK,"  
c      proceedings of the fourth Copper Mountain  
Conference on Multigrid, SIAM,  
c      1989, pp.1-12.  
c      .  
c      .  
c      .  
c      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.  
Haidvogel, and V. Pizzo,  
c      "Applications of Multigrid Software in the  
Atmospheric Sciences,"  
c      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-  
1458.  
c      .  
c      .  
c      .  
c      [9] J. Adams, "Recent Enhancements in MUDPACK, a  
Multigrid Software  
c      package for Elliptic Partial Differential  
Equations," Applied Math.  
c      and Comp., 1991, vol. 43, May 1991, pp. 79-94.  
c  
c      [10]J. Adams, "MUDPACK-2: Multigrid Software for  
Approximating  
c      Elliptic Partial Differential Equations on Uniform  
Grids with  
c      any Resolution," Applied Math. and Comp., 1993,  
vol. 53, February  
c      1993, pp. 235-249  
c      .  
c      .  
c      .  
c  
c ... argument description  
c  
c  
c  
*****  
*****  
c *** input arguments  
*****  
c  
*****  
*****
```

```
c
c
c ... iparm
c
c           an integer vector of length 23 used to
efficiently pass
c           integer arguments. iparm is set internally
in muh3
c           and defined as follows . . .
c
c
c ... intl=iparm(1)
c
c           an initialization argument. intl=0 must be
input
c           on an initial call. in this case input
arguments will
c           be checked for errors and the elliptic
partial differential
c           equation and boundary conditions will be
discretized using
c           second order finite difference formula.
c
c ***      an approximation is not generated after an
intl=0 call!
c           muh3 should be called with intl=1 to
approximate the elliptic
c           pde discretized by the intl=0 call. intl=1
should also
c           be input if muh3 has been called earlier and
only the
c           values in in rhs (see below) or gbdy (see
bndyc below)
c           or phi (see below) have changed. this will
bypass
c           redundant pde discretization and argument
checking
c           and save computational time. some examples
of when
c           intl=1 calls should be used are:
c
c           (0) after a intl=0 argument checking and
discretization call
c
```

```
c           (1) muh3 is being recalled for additional
accuracy.  in
c           this case iguess=iparm(12)=1 should also
be used.
c
c           (2) muh3 is being called every time step in a
time dependent
c           problem (see discussion below) where the
elliptic operator
c           does not depend on time.
c
c           (3) muh3 is being used to solve the same
elliptic equation
c           for several different right hand sides
(igueess=0 should
c           probably be used for each new righthand
side).
c
c           intl = 0 must be input before calling with
intl = 1 when any
c           of the following conditions hold:
c
c           (a) the initial call to muh3
c
c           (b) any of the integer arguments other than
igueess=iparm(12)
c           or maxcy=iparm(13) or mgopt have changed
since the previous
c           call.
c
c           (c) any of the floating point arguments other
than tolmax=
c           fparm(5) have changed since the previous
call
c
c           (d) any of the coefficients input by coef
(see below) have
c           changed since the previous call
c
c           (e) any of the "alfa" coefficients input by
bndyc (see below)
c           have changed since the previous call.
c
c           if any of (a) through (e) are true then the
```

```
elliptic pde
c           must be discretized or rediscretized.  if
none of (a)
c           through (e) holds, calls can be made with
intl=1.
c           incorrect calls with intl=1 will produce
erroneous results.
c ***      the values set in the saved work space "wk"
(see below) with
c           an intl=0 call must be preserved with
subsequent intl=1 calls.
c
c           MUDPACK software performance should be
monitored for intl=1
c           calls.  The intl=0 discretization call
performance depends
c           primarily on the efficiency or lack of
efficiency of the
c           user provided subroutines for pde
coefficients and
c           boundary conditions.

c
c
c ... nxa=iparm(2)
c
c           flags boundary conditions on the (y,z) plane
x=xa
c
c           = 0 if p(x,y,z) is periodic in x on [xa,xb]
c           (i.e., p(x+xb-xa,y,z) = p(x,y,z) for all
x,y,z)
c
c           = 1 if p(xa,y,z) is specified (this must be
input thru phi(1,j,k))
c
c           = 2 if there are mixed derivative boundary
conditions at x=xa
c           (see "bndyc" description below where kbdy =
1)
c
c
c ... nxb=iparm(3)
c
c           flags boundary conditions on the (y,z) plane
```

```

x=xb
c
c      = 0 if p(x,y,z) is periodic in x on [xa,xb]
c          (i.e., p(x+xb-xa,y,z) = p(x,y,z) for all
x,y,z)
c
c      = 1 if p(xb,y,z) is specified (this must be
input thru phi(nx,j,k))
c
c      = 2 if there are mixed derivative boundary
conditions at x=xb
c          (see "bndyc" description below where kbdy =
2)
c
c
c ... nyc=iparm(4)
c
c      flags boundary conditions on the (x,z) plane
y=yc
c
c      = 0 if p(x,y,z) is periodic in y on [yc,yd]
c          (i.e., p(x,y+yd-yc,z) = p(x,y,z) for all
x,y,z)

c      = 1 if p(x,yc,z) is specified (this must be
input thru phi(i,1,k))

c      = 2 if there are mixed derivative boundary
conditions at y=yc
c          (see "bndyc" description below where kbdy =
3)
c
c
c ... nyd=iparm(5)
c
c      flags boundary conditions on the (x,z) plane
y=yd
c
c      = 0 if p(x,y,z) is periodic in y on [yc,yd]
c          (i.e., p(x,y+yd-yc,z) = p(x,y,z) for all
x,y,z)

c      = 1 if p(x,yd,z) is specified (this must be
input thru phi(i,ny,k))

```

```

c      = 2 if there are mixed derivative boundary
conditions at y=yd
c      (see "bndyc" description below where kbdy =
4)
c
c
c ... nze=iparm(6)
c
c      flags boundary conditions on the (x,y) plane
z=ze
c
c      = 0 if p(x,y,z) is periodic in z on [ze,zf]
c      (i.e., p(x,y,z+zf-ze) = p(x,y,z) for all
x,y,z

c      = 1 if p(x,y,ze) is specified (this must be
input thru phi(i,j,1))

c      = 2 if there are mixed derivative boundary
conditions at z=ze
c      (see "bndyc" description below where kbdy =
5)
c
c
c ... nzf=iparm(7)
c
c      flags boundary conditions on the (x,y) plane
z=zf
c
c      = 0 if p(x,y,z) is periodic in z on [ze,zf]
c      (i.e., p(x,y,z+zf-ze) = p(x,y,z) for all
x,y,z

c      = 1 if p(x,y,zf) is specified (this must be
input thru phi(i,j,nz))

c      = 2 if there are mixed derivative boundary
conditions at z=zf
c      (see "bndyc" description below where kbdy =
6)
c
c
c *** grid size arguments

```

```
c
c
c ... ixp = iparm(8)
c
c           an integer greater than one which is used in
defining the number
c           of grid points in the x direction (see nx =
iparm(14)). "ixp+1"
c           is the number of points on the coarsest x
grid visited during
c           multigrid cycling. ixp should be chosen as
small as possible
c           within grid size requirements.

c
c
c ... jyq = iparm(9)
c
c           an integer greater than one which is used in
defining the number
c           of grid points in the y direction (see ny =
iparm(15)). "jyq+1"
c           is the number of points on the coarsest y
grid visited during
c           multigrid cycling. jyq should be chosen as
small as possible
c           within grid size requirements.

c
c
c ... kzs = iparm(10)
c
c           an integer greater than one which is used in
defining the number
c           of grid points in the z direction (see nz =
iparm(16)). "kzs+1"
c           is the number of points on the coarsest z
grid visited during
c           multigrid cycling. kzs should be chosen as
small as possible
c           within grid size requirements.

c
c
c ... iex = iparm(11)
c
c           a positive integer exponent of 2 used in
```

```
defining the number
c          of grid points in the x direction (see nx =
iparm(14)).
c          iex .le. 50 is required. for efficient
multigrid cycling,
c          iex should be chosen as large as possible and
ixp=iparm(8)
c          as small as possible within grid size
constraints when
c          defining nx = iparm(14).

c
c
c ... jey = iparm(12)
c
c          a positive integer exponent of 2 used in
defining the number
c          of grid points in the y direction (see ny =
iparm(15)).
c          jey .le. 50 is required. for efficient
multigrid cycling,
c          jey should be chosen as large as possible and
jyq=iparm(9)
c          as small as possible within grid size
constraints when
c          defining ny = iparm(15).

c
c
c ... kez = iparm(13)
c
c          a positive integer exponent of 2 used in
defining the number
c          of grid points in the z direction (see nz =
iparm(16)).
c          kez .le. 50 is required. for efficient
multigrid cycling,
c          kez should be chosen as large as possible and
kzr=iparm(10)
c          as small as possible within grid size
constraints when
c          defining nz = iparm(16).

c
c
c ... nx = iparm(14)
c
```

```

c           the number of equally spaced grid points in
the interval [xa,xb]
c           (including the boundaries).  nx must have the
form
c
c           nx = ixp*(2**(iex-1)) + 1
c
c           where ixp = iparm(8), iex = iparm(11).
c
c
c ... ny = iparm(15)
c
c           the number of equally spaced grid points in
the interval [yc,yd]
c           (including the boundaries).  ny must have the
form:
c
c           ny = jyq*(2**(jey-1)) + 1
c
c           where jyq = iparm(9), jey = iparm(12).
c
c
c ... nz = iparm(16)
c
c           the number of equally spaced grid points in
the interval [ze,zf]
c           (including the boundaries).  nz must have the
form
c
c           nz = kzs*(2**(kez-1)) + 1
c
c           where kzs = iparm(10), kez = iparm(13)
c
c
c *** note
c
c       let g be the nx by ny by nz fine grid on which the
approximation is
c       generated and let n = max0(iex,jey,kez).  in
mudpack, multigrid
c       cycling is implemented on the ascending chain of
grids
c
c       g(1) < ... < g(k) < ... < g(n) = g.

```

```

C
C      each g(k) (k=1,...,n) has mx(k) by my(k) by mz(k)
grid points
C      given by:
C
C      mx(k) = ixp*[2** (max0(iex+k-n,1)-1)] + 1
C
C      my(k) = jyq*[2** (max0(jey+k-n,1)-1)] + 1
C
C      mz(k) = kxr*[2** (max0(kez+k-n,1)-1)] + 1
C
C      additionally muh3 implements a direct method
whenever g(1) is
C      encountered.
C
C ... iguess=iparm(17)
C
C      = 0 if no initial guess to the pde is
provided
C              and/or full multigrid cycling beginning
at the
C              coarsest grid level is desired.
C
C      = 1 if an initial guess to the pde at the
finest grid
C              level is provided in phi (see below). in
this case
C              cycling beginning or restarting at the
finest grid
C              is initiated.
C
C *** comments on iguess = 0 or 1 . . .
C
C
C      setting iguess=0 forces full multigrid or "fmg"
cycling. phi
C      must be initialized at all grid points. it can be
set to zero at
C      non-Dirchlet grid points if nothing better is
available.
C
C      if iguess = 1 then the values input in phi are an
initial guess to the
C      pde at the finest grid level where cycling begins.

```

```

this option should
c      be used only if a "very good" initial guess is
available (as, for
c      example, when restarting from a previous iguess=0
call).
c
c *** time dependent problems . . .
c
c      assume we are solving an elliptic pde every time
step in a
c      marching problem of the form:
c
c          l(p(t)) = r(t)
c
c      where the differential operator "l" has no time
dependence,
c      "p(t)" is the solution and "r(t)" is the right
hand side at
c      current time "t". let "dt" be the increment
between time steps.
c      then p(t) can be used as an initial guess to
p(t+dt) with
c      intl = 1 when solving
c
c          l(p(t+dt)) = r(t+dt).
c
c      after the first two time steps, rather than
continue, it would
c      be better to define the "correction" term:
c
c          e(t,dt) = p(t+dt) - p(t)
c
c      this clearly satisfies the equation
c
c          l(e(t,dt)) = r(t+dt) - r(t).
c
c      this should be solved with iguess = 0 and intl =
1. boundary
c      conditions for e(t,dt) are obtained from the
boundary conditions
c      for p(t) by subtracting given values at t from
given values at
c      t+dt. for example if
c

```

```

c      d(p(t))/dx = f(t), d(p(t+dt))/dx = f(t+dt)
c
c      at some x boundary then e(t,dt) satisfies the
derivative
c      boundary condition
c
c      d(e(t,dt))/dx = f(t+dt) - f(t).
c
c      e(t,dt) can be preset to 0.0 (at non-Dirchlet
points) or (if p(t-dt)
c      is saved) to p(t)-p(t-dt). with iguess = 0, these
values will serve
c      as an initial guess to e(t,dt) at the coarsest
grid level. this
c      approach has the advantage that a full sequence of
multigrid cycles,
c      beginning at the coarsest grid level, is invoked
every time step in
c      solving for e(t,dt). a few digits of accuracy in
e(t,dt), which is
c      ordinarily much smaller than p(t), will yield
several more digits of
c      accuracy in the final approximation:
c
c      p(t+dt) = p(t) + e(t,dt).
c
c      using this approach to integrate in time will give
more accuracy
c      then using p(t) as an initial guess to p(t+dt) for
all time steps.
c      it does require additional storage.
c
c      if the differential operator "l" has time
dependence (either thru
c      the coefficients in the pde or the coefficients in
the derivative
c      boundary conditions) then use p(t) as an initial
guess to p(t+dt)
c      when solving
c
c      l(t+dt)(p(t+dt)) = r(t+dt)
c
c      with intl = 0 for all time steps (the
discretization must be repeated

```

```
c      for each new "t"). either iguess = 0 (p(t) will
then be an initial
c      guess at the coarsest grid level where cycles will
commence) or
c      iguess = 1 (p(t) will then be an initial guess at
the finest grid
c      level where cycles will remain fixed) can be
tried.
c
c
c ... maxcy = iparm(18)
c
c           the exact number of cycles executed between
the finest
c           (nx by ny by nz) and the coarsest ((ixp+1) by
(jyq+1) by
c           (kzr+1)) grid levels when tolmax=fparm(7)=0.0
(no error
c           control). when tolmax=fparm(7).gt.0.0 is
input (error control)
c           then maxcy is a limit on the number of cycles
between the
c           finest and coarsest grid levels. in any
case, at most
c           maxcy*(iprert+ipost) relaxation sweeps are
performed at the
c           finest grid level (see
iprer=mgopt(2),ipost=mgopt(3) below)
c           when multigrid iteration is working
"correctly" only a few
c           cycles are required for convergence. large
values for maxcy
c           should not be required.
c
c
c ... method = iparm(19)
c
c           this sets the method of relaxation (all
relaxation
c           schemes in mudpack use red/black type
ordering)
c
c           = 0 for gauss-seidel pointwise relaxation
c
```

```

C      = 1 for line relaxation in the x direction
C
C      = 2 for line relaxation in the y direction
C
C      = 3 for line relaxation in the z direction
C
C      = 4 for line relaxation in the x and y
direction
C
C      = 5 for line relaxation in the x and z
direction
C
C      = 6 for line relaxation in the y and z
direction
C
C      = 7 for line relaxation in the x,y and z
direction
C
C      = 8 for x,y planar relaxation
C
C      = 9 for x,z planar relaxation
C
C      =10 for y,z planar relaxation
C
C *** choice of method
C
C      this is very important for efficient convergence.
in some cases
C      experimentation may be required.
C
C      let fx represent the quantity cxx(x,y,z)/dlx**2
over the solution box
C
C      let fy represent the quantity cyy(x,y,z)/dly**2
over the solution box
C
C      let fz represent the quantity czz(x,y,z)/dlz**2
over the solution box
C
C      (0) if fx,fy,fz are roughly the same size and do
not vary too
C            much choose method = 0.  if this fails try
method = 7.
C

```

```
c      (1) if fx is much greater than fy,fz and fy,fz  
are roughly the same  
c          size choose method = 1  
c  
c      (2) if fy is much greater than fx,fz and fx,fz  
are roughly the same  
c          size choose method = 2  
c  
c      (3) if fz is much greater than fx,fy and fx,fy  
are roughly the same  
c          size choose method = 3  
c  
c      (4) if fx,fy are roughly the same and both are  
much greater than fz  
c          try method = 4.  if this fails try method = 8  
c  
c      (5) if fx,fz are roughly the same and both are  
much greater than fy  
c          try method = 5.  if this fails try method = 9  
c  
c      (6) if fy,fz are roughly the same and both are  
much greater than fx  
c          try method = 6.  if this fails try method =  
10  
c  
c      (7) if fx,fy,fz vary considerably with none  
dominating try method = 7  
c  
c      (8) if fx and fy are considerably greater than fz  
but not necessarily  
c          the same size and method=4 fails try method =  
8  
c  
c      (9) if fx and fz are considerably greater than fy  
but not necessarily  
c          the same size and method=5 fails try method =  
9  
c  
c      (10)if fy and fz are considerably greater than fx  
but not necessarily  
c          the same size and method=6 fails try method =  
10  
c  
c
```

```

c ... meth2 = iparm(20) determines the method of
relaxation used in the planes
c           when method = 8 or 9 or 10.
c
c
c           as above, let fx,fy,fz represent the
quantities cxx/dlx**2,
c           cyy/dly**2,czz/dlz**2 over the box.
c
c           (if method = 8)
c
c           = 0 for gauss-seidel pointwise relaxation
c               in the x,y plane for each fixed z
c           = 1 for line relaxation in the x direction
c               in the x,y plane for each fixed z
c           = 2 for line relaxation in the y direction
c               in the x,y plane for each fixed z
c           = 3 for line relaxation in the x and y
direction
c               in the x,y plane for each fixed z
c
c           (1) if fx,fy are roughly the same and vary
little choose meth2 = 0
c           (2) if fx is much greater then fy choose
meth2 = 1
c           (3) if fy is much greater then fx choose
meth2 = 2
c           (4) if none of the above or meth2 = 0 fails
choose meth2 = 3
c
c           (if method = 9)
c
c           = 0 for gauss-seidel pointwise relaxation
with red/black ordering
c               in the x,z plane for each fixed y
c           = 1 for simultaneous line relaxation in the x
direction
c               of the x,z plane for each fixed y
c           = 2 for simultaneous line relaxation in the z
direction
c               of the x,z plane for each fixed y
c           = 3 for simultaneous line relaxation in the x
and z direction
c               of the x,z plane for each fixed y

```

```

c
c           (1) if fx,fz are roughly the same and vary
little choose meth2 = 0
c           (2) if fx is much greater then fz choose
meth2 = 1
c           (3) if fz is much greater then fx choose
meth2 = 2
c           (4) if none of the above or meth2 = 0 fails
choose meth2 = 3
c
c           (if method = 10)
c
c           = 0 for gauss-seidel pointwise relaxation
with red/black ordering
c           in the y,z plane for each fixed x
c           = 1 for simultaneous line relaxation in the y
direction
c           of the y,z plane for each fixed x
c           = 2 for simultaneous line relaxation in the z
direction
c           of the y,z plane for each fixed x
c           = 3 for simultaneous line relaxation in the y
and z direction
c           of the y,z plane for each fixed x
c
c           (1) if fy,fz are roughly the same and vary
little choose meth2 = 0
c           (2) if fy is much greater then fz choose
meth2 = 1
c           (3) if fz is much greater then fy choose
meth2 = 2
c           (4) if none of the above or meth2 = 0 fails
choose meth2 = 3
c
c
c *** note that planar relaxation implements full two-
dimensional multigrid
c     cycling for each plane visited during three
dimensional multigrid
c     cycling. Consequently it can be computationally
expensive.
c
c ... length = iparm(21)
c

```

```

c           the length of the work space provided in
vector wk.

c
c           let isx = 3 if method = 1,4,5 or 7 and
nxa.ne.0
c           let isx = 5 if method = 1,4,5 or 7 and
nxa.eq.0
c           let isx = 0 if method has any other value
c
c           let jsy = 3 if method = 2,4,6 or 7 and
nyc.ne.0
c           let jsy = 5 if method = 2,4,6 or 7 and
nyc.eq.0
c           let jsy = 0 if method has any other value
c
c           let ksz = 3 if method = 3,5,6 or 7 and
nze.ne.0
c           let ksz = 5 if method = 3,5,6 or 7 and
nze.eq.0
c           let ksz = 0 if method has any other value
c
c
c           let ls =
(nx+2)*(ny+2)*(nz+2)*(10+isx+jsy+ksz)
c
c           let mx = ixp+1; my = jyq+1; mz = kzs+1.  the
block gaussian
c           elimination at the coarsest mx by my by mz
grid level requires
c
c           ld = mx*my*mz*(2*mx*my+1)
(nze.ne.0)
c
c           words of storage if z boundary conditions are
not periodic or
c
c           ld = mx*my*(mz*(4*mx*my+1))
(nze=0)
c
c           words of storage if z boundary conditions are
periodic.
c           if ixp,jyq,kzs are not the same, this
quantity is

```

```
c           minimized if they are chosen so that kzs >
max0(ixp,jyq) .
c
c           finally
c
c           length = ls + ld

c           will usually but not always suffice.  the
exact minimal length depends,
c           in a complex way, on the grid size arguments
and method chosen.
c ***      it can be predetermined for the current input
arguments by calling
c           muh3 with length=iparm(21)=0 and printing
iparm(22) or (in f90)
c           dynamically allocating the work space using
the value in iparm(22)
c           in a subsequent muh3 call.
c
c ... fparm
c
c           a floating point vector of length 8 used to
efficiently
c           pass floating point arguments.  fparm is set
internally
c           in muh3 and defined as follows . . .
c
c
c ... xa=fparm(1), xb=fparm(2)
c
c           the range of the x independent variable.  xa
must
c           be less than xb
c
c
c ... yc=fparm(3), yd=fparm(4)
c
c           the range of the y independent variable.  yc
must
c           be less than yd.
c
c
c ... ze=fparm(5), zf=fparm(6)
c
```

```
c           the range of the z independent variable. ze
must
c           be less than zf.
c
c
c ... tolmax = fparm(7)
c
c           when input positive, tolmax is a maximum
relative error tolerance
c           used to terminate the relaxation iterations.
assume phil(i,j,k)
c           and phi2(i,j,k) are the last two computed
approximations at all
c           grid points of the finest grid level. if we
define
c
c           phdif = max(abs(phi2(i,j,k)-phil(i,j,k)))
for all i,j,k
c
c           and
c
c           phmax = max(abs(phi2(i,j,k))) for all
i,j,k
c
c           then "convergence" is considered to have
occurred if and only if
c
c           phdif/phmax < tolmax.
c
c
c           if tolmax=fparm(7)=0.0 is input then there is
no error control
c           and maxcy cycles from the finest grid level
are executed. maxcy
c           is a limit which cannot be exceeded even with
error control.
c           *** calls with tolmax=0.0, when appropriate
because of known
c           convergence behavior, are more efficient than
calls with tolmax
c           positive (i.e., if possible DO NOT use error
control!).
c
c
```

```

c ... wk
c
c           a one dimensional array that must be provided
for work space.
c           see length = iparm(21). the values in wk must
be preserved
c           if muh3 is called again with
intl=iparm(1).ne.0 or if muh34
c           is called to improve accuracy.
c
c
c ... iwk
c
c           an integer vector dimensioned of length at
least
c
c           (ixp+1)*(jyq+1)*(kzr+1)
c
c           in the routine calling muh3. the length of
iwk is not
c           checked! if iwk has length too small then
undetectable
c           undetectable errors will result.
c
c
c ... bndyc
c
c           a subroutine with arguments
(kbdy,xory,yorz,alfa,gbdy) .
c           which are used to input mixed boundary
conditions to muh3.
c           the boundaries are numbered one thru six and
the form of
c           conditions are described below.
c
c
c           (1) the kbdy=1 boundary
c
c           this is the (y,z) plane x=xa where nxa=iparm(2) =
2 flags
c           a mixed boundary condition of the form
c
c           dp/dx + alfxa(y,z)*p(xa,y,z) = gbdxa(y,z)
c

```

```

c      in this case kbdy=1,xory=y,yorz=z will be input to
bndyc and
c      alfa,gbdy corresponding to alfxa(y,z),gbdxa(y,z)
must be returned.
c
c
c      (2) the kbdy=2 boundary
c
c      this is the (y,z) plane x=xb where nxb=iparm(3) =
2 flags
c      a mixed boundary condition of the form
c
c          dp/dx + alfxb(y,z)*p(xb,y,z) = gbdxb(y,z)
c
c      in this case kbdy=2,xory=y,yorz=z will be input to
bndyc and
c      alfa,gbdy corresponding to alfxb(y,z),gbdxb(y,z)
must be returned.
c
c
c      (3) the kbdy=3 boundary
c
c      this is the (x,z) plane y=yc where nyc=iparm(4) =
2 flags
c      a mixed boundary condition of the form
c
c          dp/dy + alfyc(x,z)*p(x,yc,z) = gbdyc(x,z)
c
c      in this case kbdy=3,xory=x,yorz=z will be input to
bndyc and
c      alfa,gbdy corresponding to alfyc(x,z),gbdyc(x,z)
must be returned.
c
c
c      (4) the kbdy=4 boundary
c
c      this is the (x,z) plane y=yd where nyd=iparm(5) =
2 flags
c      a mixed boundary condition of the form
c
c          dp/dy + alfyd(x,z)*p(x,yd,z) = gbdyd(x,z)
c
c      in this case kbdy=4,xory=x,yorz=z will be input to
bndyc and

```

```

c      alfa,gbdy corresponding to alfyd(x,z),gbdyd(x,z)
must be returned.

c
c
c      (5) the kbdy=5 boundary
c
c      this is the (x,y) plane z=ze where nze=iparm(6) =
2 flags
c      a mixed boundary condition of the form
c
c      dp/dz + alfze(x,y)*p(x,y,ze) = gbdze(x,y)
c
c      in this case kbdf=5,xory=x,yorz=y will be input to
bndyc and
c      alfa,gbdy corresponding to alfze(x,y),gbdze(x,y)
must be returned.

c
c
c      (6) the kbdf=6 boundary
c
c      this is the (x,y) plane z=zf where nzf=iparm(7) =
2 flags
c      a mixed boundary condition of the form
c
c      dp/dz + alfzf(x,y)*p(x,y,zf) = gbdzf(x,y)
c
c      in this case kbdf=6,xory=x,yorz=y will be input to
bndyc and
c      alfa,gbdy corresponding to alfzf(x,y),gbdzf(x,y)
must be returned.

c
c
c *** alfxa,alfyc,alfze nonpositive and
alfxb,alfyd,alfze nonnegative
c      will help maintain matrix diagonal dominance
during discretization
c      aiding convergence.

c
c *** bndyc must provide the mixed boundary condition
c      values in correspondence with those flagged in
iparm(2)
c      thru iparm(7). if all boundaries are specified
then
c      muh3 will never call bndyc. even then it must be

```

```

entered
c      as a dummy subroutine. bndyc must be declared
c      external in the routine calling muh3. the actual
c      name chosen may be different.
c
c
c ... coef
c
c      a subroutine with arguments
(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
c      which provides the known real coefficients for
the elliptic pde
c      at any grid point (x,y,z). the name chosen in
the calling routine
c      may be different where the coefficient routine
must be declared
c      external.
c
c ... rhs
c
c      an array dimensioned nx by ny by nz which
contains
c      the given right hand side values on the
uniform 3-d mesh.
c      rhs(i,j,k) = r(xi,yj,zk) for i=1,...,nx and
j=1,...,ny
c      and k=1,...,nz.
c
c ... phi
c
c      an array dimensioned nx by ny by nz . on
input phi must
c      contain specified boundary values and an
initial guess
c      to the solution if flagged (see
iguess=iparm(17)=1). for
c      example, if nyd=iparm(5)=1 then phi(i,ny,k)
must be set
c      equal to p(xi,yd,zk) for i=1,...,nx and
k=1,...,nz prior to
c      calling muh3. the specified values are
preserved by muh3.
c
c ***      if no initial guess is given (iguess=0) then

```

```

phi must still
c           be initialized at non-Dirchlet grid points
(this is not
c           checked). these values are projected down and
serve as an initial
c           guess to the pde at the coarsest grid level.
set phi to 0.0 at
c           nonDirchlet grid points if nothing better is
available.

c
c
c ... mgopt
c
c           an integer vector of length 4 which allows
the user to select
c           among various multigrid options. if
mgopt(1)=0 is input then
c           a default set of multigrid arguments (chosen
for robustness)
c           will be internally selected and the
remaining values in mgopt
c           will be ignored. if mgopt(1) is nonzero
then the arguments
c           in mgopt are set internally and defined as
follows: (see the
c           basic coarse grid correction algorithm
below)

c
c
c     kcycle = mgopt(1)
c
c           = 0 if default multigrid options are to be
used
c
c           = 1 if v cycling is to be used (the least
expensive per cycle)
c
c           = 2 if w cycling is to be used (the
default)
c
c           > 2 if more general k cycling is to be used
c           *** warning--values larger than 2 increase
c           the execution time per cycle
considerably and

```

```
c           result in the nonfatal error ierror =
-5
c           which indicates inefficient multigrid
c cycling.
c
c     iprер = mgopt(2)
c
c           the number of "pre-relaxation" sweeps
c executed before the
c           residual is restricted and cycling is
c invoked at the next
c           coarser grid level (default value is 2
c whenever mgopt(1)=0)
c
c     ipost = mgopt(3)
c
c           the number of "post relaxation" sweeps
c executed after cycling
c           has been invoked at the next coarser grid
c level and the residual
c           correction has been transferred back
c (default value is 1
c           whenever mgopt(1)=0).
c
c *** if iprер, ipost, or (especially) kcycle is greater
c than 2
c           than inefficient multigrid cycling has probably
c been chosen and
c           the nonfatal error (see below) ierror = -5 will be
c set. note
c           this warning may be overridden by any other
c nonzero value
c           for ierror.
c
c     interpol = mgopt(4)
c
c           = 1 if multilinear prolongation
c (interpolation) is used to
c           transfer residual corrections and the pde
c approximation
c           from coarse to fine grids within full
c multigrid cycling.
c
c           = 3 if multicubic prolongation
```

```

(interpolation) is used to
c           transfer residual corrections and the pde
approximation
c           from coarse to fine grids within full
multigrid cycling.
c           (this is the default value whenever
mgopt(1)=0).
c
c *** the default values (2,2,1,3) in the vector mgopt
were chosen for
c   robustness. in some cases v(2,1) cycles with
linear prolongation will
c   give good results with less computation
(especially in two-dimensions).
c   this was the default and only choice in an
earlier version of mudpack
c   (see [1]) and can be set with the integer vector
(1,2,1,1) in mgopt.
c
c *** the schedules for one full multigrid cycle
(iguess=0) using v(2,1)
c   cycles and w(2,1) cycles are depicted for a four
level grid below.
c   the number of relaxation sweeps when each grid is
visited are indicated.
c   the "*" stands for prolongation of the full
approximation and the "."
c   stands for transfer of residuals and residual
corrections within the
c   coarse grid correction algorithm (see below). the
"d" at level 1
c   indicates a direct method is used
c
c   one fmg with v(2,1) cycles:
c
c
c   -----
-----2-----1-
-----      level 4
c           *
c           .
c           *
c           .
c   -----
-----2-----1-----2-----1-----
-----      level 3
c           *
c           .
c           *
c           .

```

```

c      -----2-----1-----2-----1-----2-----1-----
-----      level 2
c      * . .
c      * . .
c      ---d---d-----d-----d-----
-----      level 1
c
c
c      one fmg with w(2,1) cycles:
c
c      -----
--1--      level 4
c                  *
.
c      -----2-----1-----2-----3-----
1----      level 3
c                  *
c      -----2-----1-----2-----3-----1-----
c      -----2-----1-----2-----3-----1-----
-----      level 2
c      * . . . . . . . . . . . .
c      --d---d-----d---d-----d---d-----d---d-----
-----      level 1
c
c
c      the form of the "recursive" coarse grid correction
cycling used
c      when kcyle.ge.0 is input is described below in
pseudo-algorithmic
c      language. it is implemented non-recursively in
fortran in mudpack.
c *** this algorithm is modified with the hybrid
solvers which use
c      a direct method whenever grid level 1 is
encountered.
c
c      algorithm
cgc(k,l(k),u(k),r(k),kcycle,iprer,ipost,iresw,intpol)
c
c *** approximately solve l(k)*u(k) = r(k) using
multigrid iteration
c *** k is the current grid level
c *** l(k) is the discretized pde operator at level k
c *** u(k) is the initial guess at level k
c *** r(k) is the right hand side at level k

```

```

c *** i(k,k-1) is the restriction operator from level k
to level k-1
c *** (the form of i(k,k-1) depends on iresw)
c *** i(k-1,k) is the prolongation operator from level
k-1 to level k
c *** (the form of i(k-1,k) depends on interpol)
c
c      begin algorithm cgc
c
c ***     pre-relax at level k
c
c      . do (i=1,iprer)
c
c      . . relax(l(k),u(k),r(k))
c
c      . end do
c
c      . if (k > 1) then
c
c ***      restrict the residual from level k to level k-
c
c      . . r(k-1) = i(k,k-1) (r(k)-l(k)*u(k))
c
c      . . kount = 0
c
c      . . repeat
c
c ***      solve for the residual correction at level k-1
in u(k-1)
c ***      using algorithm cgc "kcycle" times (this is
the recursion)
c
c      . . . kount = kount+1
c
c      . . . invoke cgc(k-1,l(k-1),u(k-1),r(k-
c      1),kcycle,iprer,ipost,iresw)
c
c
c      . . until (kount.eq.kcycle)
c
c ***      transfer residual correction in u(k-1) to
level k
c ***      with the prolongation operator and add to u(k)

```

```

C
C      . . u(k) = u(k) + i(k-1,k) (u(k-1))
C
C      . end if
C
C *** post relax at level k
C
C      . do (i=1,ipost)
C
C      . . relax(l(k),u(k),r(k))
C
C      . end do
C
C      . return
C
C      end algorithm cgc
C
C
C ****output
arguments*****
***

C
*****
C ... iparm(22)
C
C      on output iparm(22) contains the actual work
space length
C      required for the current grid sizes and
method. this value
C      will be computed and returned even if
iparm(21) is less then
C      iparm(22) (see ierror=9).
C
C
C ... iparm(23)
C
C      if error control is selected (tolmax =
fparm(7) .gt. 0.0) then
C      on output iparm(23) contains the actual

```

```
number of cycles executed
c           between the coarsest and finest grid levels
in obtaining the
c           approximation in phi.  the quantity
(iprter+ipost)*iparm(23) is
c           the number of relaxation sweeps performed at
the finest grid level.

c
c
c ... fparm(8)
c
c           on output fparm(8) contains the final
computed maximum relative
c           difference between the last two iterates at
the finest grid level.
c           fparm(8) is computed only if there is error
control (tolmax.gt.0.)
c           assume phi1(i,j,k) and phi2(i,j,k) are the
last two computed
c           values for phi(i,j,k) at all points of the
finest grid level.
c           if we define
c
c           phdif = max(abs(phi2(i,j,k)-phi1(i,j,k)))
for all i,j,k
c
c           and
c
c           phmax = max(abs(phi2(i,j,k))) for all
i,j,k
c
c           then
c
c           fparm(8) = phdif/phmax
c
c           is returned whenever phmax.gt.0.0.  in the
degenerate case
c           phmax = 0.0, fparm(8) = phdif is returned.
c
c
c ... wk
c
c           on output wk contains intermediate values
```

```

that must not be
c           destroyed if muh3 is to be called again with
iparm(1)=1 or
c           if muh34 is to be called to improve the
estimate to fourth
c           order.
c
c ... phi
c
c           on output phi(i,j,k) contains the
approximation to
c           p(xi,yj,zk) for all mesh points i=1,...,nx;
j=1,...,ny;
c           k=1,...,nz.  the last computed iterate in phi
is returned
c           even if convergence is not obtained (ierror=-1)
c
c ... ierror
c
c           for intl=iparm(1)=0 initialization calls,
ierror is an
c           error flag that indicates invalid input
arguments when
c           returned positive and nonfatal warnings when
returned
c           negative.  argument checking and
discretization
c           is bypassed for intl=1 calls which can only
return
c           ierror = -1 or 0 or 1.
c
c
c     non-fatal warnings * * *
c
c
c     ==5 if kcyle=mgopt(1) is greater than 2. values
larger than 2 results
c           in an algorithm which probably does far more
computation than
c           necessary.  kcyle = 1 (v cycles)  or kcyle=2
(w cycles) should
c           suffice for most problems.  ierror = -5 is
also set if either

```

```

c           iprер = mgopt(2) or ipost=mgopt(3) is greater
than 2. the
c           ierror=-5 flag is overridden by any other
fatal or non-fatal
c           error.
c
c           ==4 if there are dominant nonzero first order
terms in the pde which
c           make it "hyperbolic" at the finest grid level.
numerically, this
c           happens if:
c
c           abs(cx)*dlx > 2.*abs(cxx)      (dlx = (xb-
xa) / (nx-1))
c
c                           (or)
c
c           abs(cy)*dly > 2.*abs(cyy)      (dly = (yd-
yc) / (ny-1))
c
c
c           at some fine grid point (xi,yj). if an
adjustment is not made the
c           condition can lead to a matrix coming from the
discretization
c           which is not diagonally dominant and
divergence is possible. since
c           the condition is "likely" at coarser grid
levels for pde's with
c           nonzero first order terms, the adjustments
(actually first order
c           approximations to the pde)
c
c
c           cxx = amax1(cxx,0.5*abs(cx)*dx)
c
c                           (and)
c
c           cyy = amax1(cyy,0.5*abs(cy)*dy)
c
c
c           (here dx,dy are the x,y mesh sizes of the
subgrid)
c

```

```

c      are made to preserve convergence of multigrid
iteration. if made
c      at the finest grid level, it can lead to
convergence to an
c      erroneous solution (flagged by ierror = -4).
a possible remedy
c      is to increase resolution. the ierror = -4
flag overrides the
c      nonfatal ierror = -5 flag.

c
c
c      ==3 if the continuous elliptic pde is singular.
this means the
c      boundary conditions are periodic or pure
derivative at all
c      boundaries and ce(x,y) = 0.0 for all x,y. a
solution is still
c      attempted but convergence may not occur due
to ill-conditioning
c      of the linear system coming from the
discretization. the
c      ierror = -3 flag overrides the ierror=-4 and
ierror=-5 nonfatal
c      flags.

c
c
c      ==2 if the pde is not elliptic (i.e.,
cxx*cyy.le.0.0 for some (xi,yj))
c      in this case a solution is still attempted
although convergence
c      may not occur due to ill-conditioning of the
linear system.
c      the ierror = -2 flag overrides the ierror=-
5,-4,-3 nonfatal
c      flags.

c
c
c      ==1 if convergence to the tolerance specified in
tolmax=fparm(5)>0.
c      is not obtained in maxcy=iparm(13) multigrid
cycles between the
c      coarsest (ixp+1,jyq+1) and finest (nx,ny)
grid levels.
c      in this case the last computed iterate is

```

```
still returned.  
C           the ierror = -1 flag overrides all other  
nonfatal flags  
C  
C  
C       no errors * * *  
C  
C       = 0  
C  
C       fatal argument errors * * *  
C  
C       = 1 if intl=iparm(1) is not 0 on initial call or  
not 0 or 1  
C           on subsequent calls  
C  
C       = 2 if any of the boundary condition flags  
nxa,nxb,nyc,nyd,nze,nzf  
C           in iparm(2) through iparm(7) is not 0,1 or 2 or  
if  
C           (nxa,nxb) or (nyc,nyd) or (nze,nzf) are not  
pairwise zero.  
C  
C       = 3 if mino(ixp,jyq,kzr) < 2  
(ixp=iparm(8),jyq=iparm(9),kzr=iparm(10))  
C           or if ixp<3 when nxa=0 or jyq<3 when nyc=0 or  
kzr<3 when nze=0.  
C  
C       = 4 if min0(iex,jey,kez) < 1  
(iex=iparm(11),jey=iparm(12),kez=iparm(13))  
C           or if max0(iex,jey,kez) > 50  
C  
C       = 5 if nx.ne.ixp*2** (iex-1)+1 or if  
ny.ne.jyq*2** (jey-1)+1 or  
C           if nz.ne.kzr*2** (kez-1)+1  
(nx=iparm(14),ny=iparm(15),nz=iparm(16))  
C  
C       = 6 if iguess = iparm(17) is not equal to 0 or 1  
C  
C       = 7 if maxcy = iparm(18) < 1 (large values for  
maxcy should not be used)  
C  
C       = 8 if method = iparm(19) is less than 0 or  
greater than 10  
C
```

```

c      = 9 if length = iparm(20) is too small (see
iparm(21) on output
c          for minimum required work space length)
c
c      =10 if xa > xb or yc > yd or ze > zf
c
(xa=fparm(1),xb=fparm(2),yc=fparm(3),yd=fparm(4),ze=fpar
m(5),zf=fparm(6))
c
c      =11 if tolmax = fparm(7) < 0.0
c
c      errors in setting multigrid options * * * (see
also ierror=-5)
c
c      =12 if kcycle = mgopt(1) < 0 or
c          if iprer = mgopt(2) < 1 or
c          if ipost = mgopt(3) < 1 or
c          if interpol = mgopt(4) is not 1 or 3
c
c      =13 if iex=jey=kez=1 (full direct method) and
iguess=1 or maxcy > 1
c
c      =14 if the elliptic pde is singular (see ierror=-3
in mud3.d)
c
c
c
*****
*
c
*****
*
c
*****
end of muh3 documentation
c
c
*****
**
c
*****
**
c
c

```

---

## MUH34

```
C
C      file muh34.d
C
C      * * * * * * * * * * * * * * * *
* * * * * * *
C      *
*
C      *           copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *           all rights reserved
*
C      *
*
C      *           MUDPACK version 5.0.1
*
C      *
*
C      *           A Fortran Package of Multigrid
*
C      *
*
C      *           Subroutines and Example Programs
*
C      *
*
C      *           for Solving Elliptic Partial Differential
Equations          *
C      *
*
C      *           by
*
C      *
```

```
*  
C      *                               John Adams  
*  
C      *  
*  
C      *                               of  
*  
C      *  
*  
C      *           the National Center for Atmospheric  
Research          *  
C      *  
*  
C      *           Boulder, Colorado (80307)  
U.S.A.          *  
C      *  
*  
C      *           which is sponsored by  
*  
C      *  
*  
C      *           the National Science Foundation  
*  
C      *  
*  
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C  
C ... file muh34.d  
C  
C     contains documentation for:  
C     subroutine muh34(wk,iwk,phi,ierror)  
C     A sample fortran driver is file "tmuh34.f".  
C  
C ... required MUDPACK files  
C  
C     muh3.f, mudcom.f,mud3ln.f, mud3pn.f  
C  
C ... purpose  
C  
C     muh34 attempts to improve the estimate in phi,  
obtained by calling  
C     muh3, from second to fourth order accuracy. see  
the file "muh3.d"
```

```
c      for a detailed discussion of the elliptic pde
approximated and
c      arguments "wk,iwk,phi" which are also part of the
argument list for
c      muh3.
c
c ... assumptions
c
c      * phi contains a second-order approximation from
an earlier muh3 call
c
c      * arguments "wk,iwk,phi" are the same used in
calling muh3
c
c      * "wk,iwk,phi" have not changed since the last
call to muh3
c
c      * the finest grid level contains at least 6
points in each direction
c
c
c *** warning
c
c      if the first assumption is not true then a fourth
order approximation
c      cannot be produced in phi. the last assumption
(adequate grid size)
c      is the only one checked. failure in any of the
others can result in
c      in an undetectable error.
c
c ... language
c
c      fortran90/fortran77
c
c ... portability
c
c      mudpack5.0.1 software has been compiled and tested
with fortran77
c      and fortran90 on a variety of platforms.
c
c ... methods
c
c      details of the methods employed by the solvers in
```

```
mudpack are given
c      in [1,9]. [1,2,9] contain performance
measurements on a variety of
c      elliptic pdes (see "references" in the file
"readme"). in summary:
c
c
c *** higher order solution (fourth-order solvers) (see
[9,19,21])
c
c      if the multigrid cycling results in a second-order
estimate (i.e.,
c      discretization level error is reached) then this
can be improved to a
c      fourth-order estimate using the technique of
"deferred corrections"
c      the values in the solution array are used to
generate a fourth-order
c      approximation to the truncation error. second-
order finite difference
c      formula are used to approximate third and fourth
partial derivatives
c      of the solution function [3]. the truncation
error estimate is
c      transferred down to all grid levels using weighted
averaging where
c      it serves as a new right hand side. the default
multigrid options
c      are used to compute the fourth-order correction
term which is added
c      to the original solution array.
c
c
c ... references (partial)
c
c
c      [1] J. Adams, "MUDPACK: Multigrid Fortran Software
for the Efficient
c      Solution of Linear Elliptic Partial Differential
Equations,"
c      Applied Math. and Comput. vol.34, Nov 1989,
pp.113-146.
c
c      [2] J. Adams, "FMG Results with the Multigrid
```

```
Software Package MUDPACK,"  
c      proceedings of the fourth Copper Mountain  
Conference on Multigrid, SIAM,  
c      1989, pp.1-12.  
c      .  
c      .  
c      .  
c      [7] J. Adams, R. Garcia, B. Gross, J. Hack, D.  
Haidvogel, and V. Pizzo,  
c      "Applications of Multigrid Software in the  
Atmospheric Sciences,"  
c      Mon. Wea. Rev., vol. 120 # 7, July 1992, pp. 1447-  
1458.  
c      .  
c      .  
c      .  
c      [9] J. Adams, "Recent Enhancements in MUDPACK, a  
Multigrid Software  
c      package for Elliptic Partial Differential  
Equations," Applied Math.  
c      and Comp., 1991, vol. 43, May 1991, pp. 79-94.  
c  
c      [10]J. Adams, "MUDPACK-2: Multigrid Software for  
Approximating  
c      Elliptic Partial Differential Equations on Uniform  
Grids with  
c      any Resolution," Applied Math. and Comp., 1993,  
vol. 53, February  
c      1993, pp. 235-249  
c      .  
c      .  
c      .  
c  
c ... error parameter  
c  
c      = 0 if no error is detected  
c  
c      = 30 if min0(nx,ny,nz) < 6 where nx,ny,nz are the  
fine grid sizes  
c              in the x,y,z directions.  
c  
c  
c  
*****
```

```
*****
C
*****
C      end of muh34 documentation
C
C
*****
C
*****
C
```

---

## RESC2

```
C
C      file resc2.f
C
C      * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK version 5.0.1
*
```



```
C
C      subroutine resc2(nx,ny,work,res)
C
C
C ... purpose
C
C
C      subroutine resc2 computes the fine grid residual
in the nx by ny array
C      res after calling cud2 or cuh2 or cud2sa.  if
C
C          L * p = f
C
C      is the n by n (n = nx*ny) block tri-diagonal
linear system resulting
C      from the pde discretization (done internally in
the mudpack solver)
C      then resc2 computes the nx by ny residual array
C
C          res = f - L * p
C
C      one of the vector norms of res,
C
C          || res ||
C
C      can be computed as a "measure" of how well the
approximation satisfies
C      the discretization equations.
C
C *** please note:
C
C      let pe be the exact continuous solution to
the elliptic pde
C          evaluated on the nx by ny discretization grid
C
C      let p be the exact solution to the linear
discretization
C
C      let phi be the approximation to p generated
by the mudpack solver
C
C      then discretization level error is defined by the
condition
C
```

```

c      || phi - p || < || p - pe || .
c          =
c
c      a common measure of multigrid efficiency is that
discretization level
c      error is reached in one full multigrid cycle (see
references [2,9] in
c      the mudpack file "readme"). this can happen
before the residual is
c      reduced to the level of roundoff error.
consequently, || res || is
c      a conservative measure of accuracy which can be
wasteful if multi-
c      grid cycles are executed until it reaches the
level of roundoff error.
c
c      || res || can be used to estimate the convergence
rate of multigrid
c      iteration. let r(n) be the residual and e(n) be
the error after
c      executing n cycles. they are related by the
residual equation
c
c      L * e(n) = r(n).
c
c      it follows that the ratio
c
c      || r(n+1) || / || r(n) ||
c
c      estimates
c
c      || e(n+1) || / || e(n) ||
c
c      which in turn estimates the convergence rate
c
c      c = max_k || e(k+1) || / || e(k) || .
c
c
c      notice
c
c      || e(n) || < c^n || e(0) || .
c
c
c ... assumptions (see cud2.d or cuh2.d or cud2sa.d)

```

```

c
c      (1) nx,ny have the same values as
iparm(10),iparm(11) (used
c          to set the fine grid resolution)
c
c      (2) work is the same argument used in calling the
solver.
c
c      (3) work have not changed since the last call to
the solver.
c
c      (3) assures a copy of the last approximation phi
is contained in work.
c      if these assumptions are not true then resc2
cannot compute the
c      residual in res.
c
      subroutine resc2(nx,ny,work,res)
      implicit none
      integer nx,ny,ic
      complex work(*),res(nx,ny)
c
c      set approximation and coefficient pointers
c
      ic = 1+(nx+2)*(ny+2)
      call rec2(nx,ny,work,work(ic),res)
      return
      end

      subroutine rec2(nx,ny,phi,cof,res)
      implicit none
      integer nx,ny,i,j
      complex phi(0:nx+1,0:ny+1),cof(nx,ny,6),res(nx,ny)
c
c      compute residual over entire x,y grid
c
      do j=1,ny
      do i=1,nx
        res(i,j) = cof(i,j,6)-
          + cof(i,j,1)*phi(i-1,j)+ 
          + cof(i,j,2)*phi(i+1,j)+ 
          + cof(i,j,3)*phi(i,j-1)+ 
          + cof(i,j,4)*phi(i,j+1)+ 
          + cof(i,j,5)*phi(i,j))

```

```
    end do
    end do
    return
end
```

---

## RESC2CR

```
C
C      file resc2cr.f
C
C      * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *               University Corporation for Atmospheric
Research           *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK version 5.0.1
*
C      *
*
C      *               A Fortran Package of Multigrid
*
C      *
*
C      *               Subroutines and Example Programs
*
C      *
```



```

c      l * p = f
c
c      is the n by n (n = nx*ny) block tri-diagonal
linear system resulting
c      from the pde discretization (done internally in
cud2cr or cuh2cr) and
c      phi is the approximation to p obtained by calling
cud2cr or cuh2cr,
c      then resc2cr computes the nx by ny residual array
c
c      res = f - l * phi.
c
c      one of the vector norms of res,
c
c      || res ||
c
c      can be computed as a "measure" of how well phi
satisfies the
c      discretization equations. for example, the
following statements
c      will compute the location and size of the maximum
residual in res
c      on cray computers:
c
c      ij = isamax(nx*ny,res,1)
c
c      jmax = (ij-1)/nx + 1
c
c      imax = ij - (jmax-1)*nx
c
c      resmax = abs(res(imax,jmax))
c
c
c *** please note:
c
c      let pe be the exact continuous solution to
the elliptic pde
c      evaluated on the nx by ny discretization grid
c
c      let p be the exact solution to the linear
discretization
c
c      let phi be the approximation to p generated
by the mudpack solver

```

```

c
c      then discretization level error is defined by the
condition
c
c          || phi - p || < || p - pe || .
c                      =
c
c      a common measure of multigrid efficiency is that
discretization level
c      error is reached in one full multigrid cycle (see
references [2,9] in
c      the mudpack file "readme"). this can happen
before the residual is
c      reduced to the level of roundoff error.
consequently, || res || is
c      a conservative measure of accuracy which can be
wasteful if multi-
c      grid cycles are executed until it reaches the
level of roundoff error.
c
c      || res || can be used to estimate the convergence
rate of multigrid
c      iteration. let r(n) be the residual and e(n) be
the error after
c      executing n cycles. they are related by the
residual equation
c
c          l * e(n) = r(n) .
c
c      it follows that the ratio
c
c          || r(n+1) || / || r(n) ||
c
c      estimates
c
c          || e(n+1) || / || e(n) ||
c
c      which in turn estimates the convergence rate
c
c          c = max_k || e(k+1) || / || e(k) || .
c
c      notice
c                      n

```

```

c      || e(n) || < c || e(0) || .
c
c
c ... assumptions (see cuh2cr.d or cud2cr.d)
c
c      (1) nx,ny have the same values as
iparm(10),iparm(11) (used
c          to set the fine grid resolution when calling
cud2cr or cuh2cr)
c
c      (2) work is the same argument used in calling
cud2cr or cuh2cr.
c
c      (3) work has not changed since the last call to
cud2cr or cuh2cr.
c
c      if these assumptions are not true then resc2cr
cannot compute the
c      residual in res. (3) assures a copy of the last
approximation phi
c      is contained in work.
c
      subroutine resc2cr(nx,ny,work,res)
      implicit none
      integer nx,ny,ic
      real work(*),res(nx,ny)
c
c      set pointer for fine grid coefficients in work
c
      ic = 1+(nx+2)*(ny+2)
      call rec2cr(nx,ny,work,work(ic),res)
      return
      end

      subroutine rec2cr(nx,ny,phi,cof,res)
c
c      compute residual
c
      implicit none
      integer nx,ny,i,j
      real phi(0:nx+1,0:ny+1),cof(nx,ny,10),res(nx,ny)
      do j=1,ny
      do i=1,nx
      res(i,j) = cof(i,j,10) - (cof(i,j,9)*phi(i,j) +

```

```

+           cof(i,j,1)*phi(i+1,j) +
cof(i,j,2)*phi(i+1,j+1) +
+           cof(i,j,3)*phi(i,j+1) +
cof(i,j,4)*phi(i-1,j+1) +
+           cof(i,j,5)*phi(i-1,j) +
cof(i,j,6)*phi(i-1,j-1) +
+           cof(i,j,7)*phi(i,j-1) +
cof(i,j,8)*phi(i+1,j-1) )
end do
end do
return
end

```

RESC2SP

```
C      file resc2sp.f
C
C      * * * * * * * * * * * * * * * *
C      *
C      *
C      *
C      *
C      *               copyright (c) 2008 by UCAR
C      *
C      *
C      *               University Corporation for Atmospheric
Research          *
C      *
C      *
C      *               all rights reserved
C      *
C      *
C      *               MUDPACK version 5.0.1
C      *
C      *
```

C \* A Fortran Package of Multigrid  
\*  
C \*  
\*  
C \* Subroutines and Example Programs  
\*  
C \*  
\*  
C \* for Solving Elliptic Partial Differential  
Equations \*  
C \*  
\*  
C \* by  
\*  
C \*  
\*  
C \* John Adams  
\*  
C \*  
\*  
C \* of  
\*  
C \*  
\*  
C \*  
\*  
C \* the National Center for Atmospheric  
Research \*  
C \*  
\*  
C \* Boulder, Colorado (80307)  
U.S.A. \*  
C \*  
\*  
C \* which is sponsored by  
\*  
C \*  
\*  
C \* the National Science Foundation  
\*  
C \*  
\*  
C \*  
\* \* \* \* \* \* \* \*  
C  
C subroutine resc2sp(nx,ny,nxa,nxb,nyc,nyd,work,res)

```

C
C
C ... purpose
C
C
C      subroutine resc2sp computes the fine grid residual
in the nx by ny array
C      res after calling cud2sp.  if
C
C          l * p = f
C
C      is the n by n (n = nx*ny) block tri-diagonal
linear system resulting
C      from the pde discretization (done internally in
cud2sp) and phi is the
C      approximation to p obtained by calling cud2sp,
then resc2sp computes the
C      nx by ny residual array
C
C          res = f - l * phi.
C
C      one of the vector norms of res,
C
C          || res ||
C
C      can be computed as a "measure" of how well phi
satisfies the
C      discretization equations.  for example, the
following statements
C      will compute the location and size of the maximum
residual in res
C      on cray computers:
C
C          ij = isamax(nx*ny,res,1)
C
C          jmax = (ij-1)/nx + 1
C
C          imax = ij - (jmax-1)*nx
C
C          resmax = abs(res(imax,jmax))
C
C
C *** please note:
C

```

```

c           let pe be the exact continuous solution to
the elliptic pde
c           evaluated on the nx by ny discretization grid
c
c           let p be the exact solution to the linear
discretization
c
c           let phi be the approximation to p generated
by the mudpack solver
c
c           then discretization level error is defined by the
condition
c
c           || phi - p || < || p - pe ||
c           =
c
c           a common measure of multigrid efficiency is that
discretization level
c           error is reached in one full multigrid cycle (see
references [2,9] in
c           the mudpack file "readme"). this can happen
before the residual is
c           reduced to the level of roundoff error.
consequently, || res || is
c           a conservative measure of accuracy which can be
wasteful if multi-
c           grid cycles are executed until it reaches the
level of roundoff error.
c
c           || res || can be used to estimate the convergence
rate of multigrid
c           iteration. let r(n) be the residual and e(n) be
the error after
c           executing n cycles. they are related by the
residual equation
c
c           l * e(n) = r(n).
c
c           it follows that the ratio
c
c           || r(n+1) || / || r(n) ||
c
c           estimates
c

```

```

c           || e(n+1) || / || e(n) ||
c
c   which in turn estimates the convergence rate
c
c   c = max || e(k+1) || / || e(k) || .
c           k
c
c   notice
c           n
c   || e(n) || < c || e(0) || .
c
c
c   ... assumptions (see cud2sp.d)
c
c   (1) nx,ny have the same values as
iparm(10),iparm(11) (used
c           to set the fine grid resolution when calling
cud2sp)
c
c   (2) nxa,nxb,nyc,nyd have the same values as
iparm(2),iparm(3),
c           iparm(4),iparm(5) (boundary condition flags)
used to call
c           cud2sp
c
c   (3) work is the same work space argument used in
calling cud2sp.
c
c   (4) work has not changed since the last call to
cud2sp.
c
c   If (1)-(4) are not true then resc2sp cannot
compute the residual
c   in res. (3),(4) assure a copy of the last
computed phi is in work.
c
subroutine resc2sp(nx,ny,nxa,nxb,nyc,nyd,work,res)
implicit none
integer nx,ny,nxa,nxb,nyc,nyd,irh,icx,icy
complex work(*),res(nx,ny)
c
c   set pointer for fine grid coefficients in work
c
irh = 1 + (nx+2)*(ny+2)

```

```

icx = irh + nx*ny
icy = icx + 3*nx
call rec2sp(nx,ny,nxa,nxb,nyc,nyd,work,work(irh),
+           work(icx),work(icy),res)
return
end

subroutine
rec2sp(nx,ny,nxa,nxb,nyc,nyd,phi,rhs,cofx,cofy,res)
implicit none
integer nx,ny,nxa,nxb,nyc,nyd,i,j,ist,ifn,jst,jfn
complex phi(0:nx+1,0:ny+1),rhs(nx,ny)
complex cofx(nx,3),cofy(ny,3),res(nx,ny)

C
C      initialize residual to zero and set limits
C
do j=1,ny
do i=1,nx
  res(i,j) = (0.0,0.0)
end do
  end do
C
C      set limits
C
ist = 1
if (nxa.eq.1) ist = 2
ifn = nx
if (nxb.eq.1) ifn = nx-1
jst = 1
if (nyc.eq.1) jst = 2
jfn = ny
if (nyd.eq.1) jfn = ny-1
C
C      compute residual on nonspecified grid points
C
do j=jst,jfn
do i=ist,ifn
  res(i,j) =  rhs(i,j)-
+            cofx(i,1)*phi(i-1,j)+ 
+            cofx(i,2)*phi(i+1,j)+ 
+            cofy(j,1)*phi(i,j-1)+ 
+            cofy(j,2)*phi(i,j+1)+ 
+            (cofx(i,3)+cofy(j,3))*phi(i,j))
end do

```

```
end do  
return  
end
```

---

## RESC3

```
C  
C      file resc3.f  
C  
C      * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C      *  
*  
C      *          copyright (c) 2008 by UCAR  
*  
C      *  
*  
C      *          University Corporation for Atmospheric  
Research           *  
C      *  
*  
C      *          all rights reserved  
*  
C      *  
*  
C      *          MUDPACK  version 5.0.1  
*  
C      *  
*  
C      *          A Fortran Package of Multigrid  
*  
C      *  
*  
C      *          Subroutines and Example Programs  
*  
C      *  
*  
C      *          for Solving Elliptic Partial Differential
```

```
Equations      *
C      *
*
C      *                                by
*
C      *
*
C      *                                John Adams
*
C      *
*
C      *                                of
*
C      *
*
C      *                                the National Center for Atmospheric
Research      *
C      *
*
C      *                                Boulder, Colorado (80307)
U.S.A.      *
C      *
*
C      *                                which is sponsored by
*
C      *
*
C      *                                the National Science Foundation
*
C      *
*
C      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *
*      *      *      *      *      *      *
C
C      subroutine resc3(nx,ny,nz,work,res)
C
C
C      ... purpose
C
C
C      subroutine resc3 computes the fine grid residual
in the nx by ny by nz
C      array res after calling cuh3 or cud3 or cud3sa.
if
```

```

c
c           L * p = f
c
c           is the n by n (n = nx*ny*nz) block tri-diagonal
c           linear system resulting
c           from the pde discretization (done internally in
c           cud3 or cuh3) and phi
c           is the approximation to p obtained by calling
c           cud3, then resc3 computes
c           the nx by ny by nz residual array
c
c           res = f - L * phi.
c
c           one of the vector norms of res,
c
c           || res ||
c
c           can be computed as a "measure" of how well phi
c           satisfies the
c           discretization equations. for example, the
c           following statements
c           will compute the location and size of the maximum
c           residual in res
c           on cray computers:
c
c           ijk = isamax(nx*ny*nz,res,1)
c
c           kmax = (ijk-1)/(nx*ny) + 1
c
c           jmax = (ijk-(kmax-1)*nx*ny-1)/nx + 1
c
c           imax = ij - nx*((kmax-1)*ny-jmax+1)
c
c           resmax = abs(res(imax,jmax,kmax))
c
c
c *** please note:
c
c           let pe be the exact continuous solution to
c           the elliptic pde
c           evaluated on the nx by ny by nz
c           discretization grid
c
c           let p be the exact solution to the linear

```

```

discretization
c
c           let phi be the approximation to p generated
by the mudpack solver
c
c           then discretization level error is defined by the
condition
c
c           || phi - p || < || p - pe ||.
c                   =
c
c           a common measure of multigrid efficieny is that
discretization level
c           error is reached in one full multigrid cycle (see
references [2,9] in
c           the mudpack file "readme"). this can happen
before the residual is
c           reduced to the level of roundoff error.
consequently, || res || is
c           a conservative measure of accuracy which can be
wasteful if multi-
c           grid cycles are executed until it reaches the
level of roundoff error.
c
c           || res || can be used to estimate the convergence
rate of multigrid
c           iteration. let r(n) be the residual and e(n) be
the error after
c           executing n cycles. they are related by the
residual equation
c
c           L * e(n) = r(n).
c
c           it follows that the ratio
c
c           || r(n+1) || / || r(n) ||
c
c           estimates
c
c           || e(n+1) || / || e(n) ||
c
c           which in turn estimates the convergence rate
c
c           c = max || e(k+1) || / || e(k) ||.

```

```

c          k
c
c      notice
c          n
c      || e(n) || < c || e(0) || .
c
c
c ... assumptions (see cud3.d or cud3sa.d or cuh3.d)
c
c      (1) nx,ny,nz have the same values as
iparm(14),iparm(15),iparm(16)
c          (used to set the fine grid resolution when
calling cud3 or cuh3 or cud3sa)
c
c      (2) work,phi are the same parameters used in
calling cud3 or cuh3 or cud3sa
c
c      (3) work,phi have not changed since the last call
to cud3 or cud3sa or cuh3
c
c      (3) assures a copy of the last approximation phi
is contained in work.
c      if these assumptions are not true then resc3
cannot compute the
c      residual in res.
c
subroutine resc3(nx,ny,nz,work,res)
c
c      compute fine grid residual in res after calling
cud3 or cud3sa or cuh3
c
implicit none
integer nx,ny,nz,ic
complex work(*),res(nx,ny,nz)
c
c      set coefficient pointer
c
ic = 1+(nx+2)*(ny+2)*(nz+2)
ic = 1+(nx+2)*(ny+2)*(nz+2)
call rec3(nx,ny,nz,work,work(ic),res)
return
end

```

```

subroutine rec3(nx,ny,nz,phi,cof,res)
implicit none
integer nx,ny,nz,i,j,k
complex
cof(nx,ny,nz,8),phi(0:nx+1,0:ny+1,0:nz+1),res(nx,ny,nz)
do k=1,nz
do j=1,ny
do i=1,nx
    res(i,j,k) = cof(i,j,k,8) - (
        cof(i,j,k,1)*phi(i-1,j,k)+
        cof(i,j,k,2)*phi(i+1,j,k)+
        cof(i,j,k,3)*phi(i,j-1,k)+
        cof(i,j,k,4)*phi(i,j+1,k)+
        cof(i,j,k,5)*phi(i,j,k-1)+
        cof(i,j,k,6)*phi(i,j,k+1)+
        cof(i,j,k,7)*phi(i,j,k) )
end do
end do
end do
return
end

```

## RESC3SP

```

C
C      file resc3sp.f
C
C      * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *                               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *

```

\*  
C \* all rights reserved  
\*  
C \*  
\*  
C \* MUDPACK version 5.0.1  
\*  
C \*  
\*  
C \* A Fortran Package of Multigrid  
\*  
C \*  
\*  
C \* Subroutines and Example Programs  
\*  
C \*  
\*  
C \* for Solving Elliptic Partial Differential  
Equations \*  
C \*  
\*  
C \* by  
\*  
C \*  
\*  
C \* John Adams  
\*  
C \*  
\*  
C \* of  
\*  
C \*  
\*  
C \* the National Center for Atmospheric  
Research \*  
C \*  
\*  
C \* Boulder, Colorado (80307)  
U.S.A. \*  
C \*  
\*  
C \* which is sponsored by  
\*  
C \*

```

*
C      *                               the National Science Foundation
*
C      *
*
C      * * * * * * * * * * * * * * * *
* * * * * * * *
C
C      subroutine
resc3sp(nx,ny,nz,nxa,nxb,nyc,nyd,nze,nzf,work,res)
C
C
C ... purpose
C
C
C      subroutine resc3sp computes the fine grid residual
in the nx by ny by nz
C      array res after calling cud3sp.  if
C
C          l * p = f
C
C      is the n by n (n = nx*ny*nz) block tri-diagonal
linear system resulting
C      from the pde discretization (done internally in
cud3sp) and phi is the
C      approximation to p obtained by calling cud3sp,
then resc3sp computes
C      the nx by ny by nz residual array
C
C      res = f - l * phi.
C
C      one of the vector norms of res,
C
C          || res ||
C
C      can be computed as a "measure" of how well phi
satisfies the
C      discretization equations.  for example, the
following statements
C      will compute the location and size of the maximum
residual in res
C      on cray computers:
C
C          ijk = isamax(nx*ny*nz,res,1)

```

```

C
C           kmax = (ijk-1) / (nx*ny) + 1
C
C           jmax = (ijk-(kmax-1)*nx*ny-1)/nx + 1
C
C           imax = ij - nx* ((kmax-1)*ny-jmax+1)
C
C           resmax = abs(res(imax,jmax,kmax))
C
C
C *** please note:
C
C           let pe be the exact continuous solution to
the elliptic pde
C           evaluated on the nx by ny by nz
discretization grid
C
C           let p be the exact solution to the linear
discretization
C
C           let phi be the approximation to p generated
by the mudpack solver
C
C           then discretization level error is defined by the
condition
C
C           || phi - p || < || p - pe ||
C           =
C
C           a common measure of multigrid efficiency is that
discretization level
C           error is reached in one full multigrid cycle (see
references [2,9] in
C           the mudpack file "readme"). this can happen
before the residual is
C           reduced to the level of roundoff error.
consequently, || res || is
C           a conservative measure of accuracy which can be
wasteful if multi-
C           grid cycles are executed until it reaches the
level of roundoff error.
C
C           || res || can be used to estimate the convergence
rate of multigrid

```

```

c      iteration. let r(n) be the residual and e(n) be
the error after
c      executing n cycles. they are related by the
residual equation
c
c          l * e(n) = r(n).
c
c      it follows that the ratio
c
c          || r(n+1) || / || r(n) ||
c
c      estimates
c
c          || e(n+1) || / || e(n) ||
c
c      which in turn estimates the convergence rate
c
c          c = max_k || e(k+1) || / || e(k) ||.
c
c
c      notice
c
c          || e(n) || < c^n || e(0) ||.
c
c
c ... assumptions (see cud3sp.d)
c
c      (1) nx,ny,nz have the same values as
iparm(14),iparm(15),iparm(16)
c          (used to set the fine grid resolution when
calling cud3sp)
c
c      (2) nxa,nxb,nyc,nyd,nze,nzf have the same values
as iparm(2),
c          iparm(3),iparm(4),iparm(5),iparm(6),iparm(7)
(used to flag
c          boundary conditions when calling cud3sp).
c
c      (3) work,phi are the same parameters used in
calling cud3sp.
c
c      (4) work,phi have not changed since the last call
to cud3sp.
c

```

```

c      (3) assures a copy of the last approximation phi
is contained in work.
c      if (1)-(4) are not true then resc3sp cannot
compute the residual.
c
      subroutine
resc3sp(nx,ny,nz,nxa,nxb,nyc,nyd,nze,nzf,wk,res)
      implicit none
      integer
nx,ny,nz,nxa,nxb,nyc,nyd,nze,nzf,ir,ix,iy,iz
      complex wk(*),res(*)
c
c      set pointers
c
      ir = 1+(nx+2)*(ny+2)*(nz+2)
      ix = ir+nx*ny*nz
      iy = ix+3*nx
      iz = iy+3*ny
      call
rec3sp(nx,ny,nz,nxa,nxb,nyc,nyd,nze,nzf,wk,wk(ir),wk(ix)
,
      +               wk(iy),wk(iz),res)
      return
      end

      subroutine
rec3sp(nx,ny,nz,nxa,nxb,nyc,nyd,nze,nzf,phi,
      +               rhs,cofx,cofy,cofz,resf)
      implicit none
      integer nx,ny,nz,nxa,nxb,nyc,nyd,nze,nzf
      integer i,j,k,ist,ifn,jst,jfn,kst,kfn
      complex cofx(nx,3),cofy(ny,3),cofz(nz,3)
      complex
rhs(nx,ny,nz),phi(0:nx+1,0:ny+1,0:nz+1),resf(nx,ny,nz)
c
c      intialize residual to zero and set loop limits
c
      do k=1,nz
      do j=1,ny
      do i=1,nx
          resf(i,j,k) = (0.0,0.0)
      end do
      end do
      end do

```

```

ist = 1
if (nxa.eq.1) ist = 2
ifn = nx
if (nxb.eq.1) ifn = nx-1
jst = 1
if (nyc.eq.1) jst = 2
jfn = ny
if (nyd.eq.1) jfn = ny-1
kst = 1
if (nze.eq.1) kst = 2
kfn = nz
if (nzf.eq.1) kfn = nz-1
c
c      compute residual
c
do k=kst,kfn
do j=jst,jfn
do i=ist,ifn
    resf(i,j,k) = rhs(i,j,k)-
    +      cofx(i,1)*phi(i-
1,j,k)+cofx(i,2)*phi(i+1,j,k) +
    +      cofy(j,1)*phi(i,j-
1,k)+cofy(j,2)*phi(i,j+1,k) +
    +      cofz(k,1)*phi(i,j,k-
1)+cofz(k,2)*phi(i,j,k+1) +
    +      (cofx(i,3)+cofy(j,3)+cofz(k,3))*phi(i,j,k)
)
end do
end do
end do
return
end

```

## RESM2

```

c
c      file resm2.f
c

```

```
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * * *  
C      *  
*  
C      *          copyright (c) 2008 by UCAR  
*  
C      *  
*  
C      *          University Corporation for Atmospheric  
Research      *  
C      *  
*  
C      *          all rights reserved  
*  
C      *  
*  
C      *          MUDPACK version 5.0.1  
*  
C      *  
*  
C      *          A Fortran Package of Multigrid  
*  
C      *  
*  
C      *          Subroutines and Example Programs  
*  
C      *  
*  
C      *          for Solving Elliptic Partial Differential  
Equations      *  
C      *  
*  
C      *          by  
*  
C      *  
*  
C      *          John Adams  
*  
C      *  
*  
C      *          of  
*  
C      *  
*
```

```
C      *          the National Center for Atmospheric
Research           *
C      *
*
C      *          Boulder, Colorado (80307)
U.S.A.           *
C      *
*
C      *          which is sponsored by
*
C      *
*
C      *          the National Science Foundation
*
C      *
*
C      *          ****
* * * * * * * *
C
C      subroutine resm2(nx,ny,work,res)
C
C
C ... purpose
C
C
C      subroutine resm2 computes the fine grid residual
in the nx by ny array
C      res after calling mud2 or muh2 or mud2sa. if
C
C          L * p = f
C
C      is the n by n (n = nx*ny) block tri-diagonal
linear system resulting
C      from the pde discretization (done internally in
the mudpack solver)
C      then resm2 computes the nx by ny residual array
C
C          res = f - L * p
C
C      one of the vector norms of res,
C
C          || res ||
C
C      can be computed as a "measure" of how well the
```

```

approximation satisfies
c      the discretization equations.  for example, the
following statements
c      will compute the location and size of the maximum
residual in res
c      on cray computers:
c
c          ij = isamax(nx*ny,res,1)
c
c          jmax = (ij-1)/nx + 1
c
c          imax = ij - (jmax-1)*nx
c
c          resmax = abs(res(imax,jmax))
c
c
c *** please note:
c
c      let pe be the exact continuous solution to
the elliptic pde
c          evaluated on the nx by ny discretization grid
c
c      let p be the exact solution to the linear
discretization
c
c      let phi be the approximation to p generated
by the mudpack solver
c
c      then discretization level error is defined by the
condition
c
c          || phi - p || < || p - pe ||
c                  =
c
c      a common measure of multigrid efficiency is that
discretization level
c      error is reached in one full multigrid cycle (see
references [2,9] in
c      the mudpack file "readme").  this can happen
before the residual is
c      reduced to the level of roundoff error.
consequently, || res || is
c      a conservative measure of accuracy which can be
wasteful if multi-

```

```

c      grid cycles are executed until it reaches the
level of roundoff error.

c
c      || res || can be used to estimate the convergence
rate of multigrid
c      iteration. let r(n) be the residual and e(n) be
the error after
c      executing n cycles. they are related by the
residual equation
c
c          L * e(n) = r(n).
c
c      it follows that the ratio
c
c          || r(n+1) || / || r(n) ||
c
c      estimates
c
c          || e(n+1) || / || e(n) ||
c
c      which in turn estimates the convergence rate
c
c          c = max_k || e(k+1) || / || e(k) ||.
c
c      notice
c
c          || e(n) || < cn || e(0) ||.
c
c
c ... assumptions (see mud2.d or muh2.d or mud2sa.d)
c
c      (1) nx,ny have the same values as
iparm(10),iparm(11) (used
c          to set the fine grid resolution)
c
c      (2) work is the same argument used in calling the
solver.
c
c      (3) work have not changed since the last call to
the solver.
c
c      (3) assures a copy of the last approximation phi
is contained in work.

```

```

c      if these assumptions are not true then resm2
cannot compute the
c      residual in res.
c
      subroutine resm2(nx,ny,work,res)
      implicit none
      integer nx,ny,ic
      real work(*),res(nx,ny)
c
c      set approximation and coefficient pointers
c
      ic = 1+(nx+2)*(ny+2)
      call rem2(nx,ny,work,work(ic),res)
      return
      end

      subroutine rem2(nx,ny,phi,cof,res)
      implicit none
      integer nx,ny,i,j
      real phi(0:nx+1,0:ny+1),cof(nx,ny,6),res(nx,ny)
c
c      compute residual over entire x,y grid
c
      do j=1,ny
      do i=1,nx
         res(i,j) = cof(i,j,6)-
            + cof(i,j,1)*phi(i-1,j)+ 
            + cof(i,j,2)*phi(i+1,j)+ 
            + cof(i,j,3)*phi(i,j-1)+ 
            + cof(i,j,4)*phi(i,j+1)+ 
            + cof(i,j,5)*phi(i,j))
      end do
      end do
      return
      end

```

---

## RESM2CR

```
C
C      file resm2cr.f
C
C      * * * * *
* * * * *
C      *
*
C      *           copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *           all rights reserved
*
C      *
*
C      *           MUDPACK version 5.0.1
*
C      *
*
C      *           A Fortran Package of Multigrid
*
C      *
*
C      *           Subroutines and Example Programs
*
C      *
*
C      *           for Solving Elliptic Partial Differential
Equations          *
C      *
*
C      *           by
*
C      *
*
C      *           John Adams
*
C      *
```

```
C      *
      of
*
C      *
*
C      *           the National Center for Atmospheric
Research          *
C      *
*
C      *           Boulder, Colorado (80307)
U.S.A.          *
C      *
*
C      *           which is sponsored by
*
C      *
*
C      *           the National Science Foundation
*
C      *
*
C      *           ****
* * * * * * *
C
C      subroutine resm2cr(nx,ny,work,res)
C
C
C ... purpose
C
C
C      subroutine resm2cr computes the fine grid residual
in the nx by ny array
C      res after calling mud2cr or muh2cr.  if
C
C          l * p = f
C
C      is the n by n (n = nx*ny) block tri-diagonal
linear system resulting
C      from the pde discretization (done internally in
mud2cr or muh2cr) and
C      phi is the approximation to p obtained by calling
mud2cr or muh2cr,
C      then resm2cr computes the nx by ny residual array
C
C          res = f - l * phi.
```

```

c
c      one of the vector norms of res,
c
c      || res ||
c
c      can be computed as a "measure" of how well phi
satisfies the
c      discretization equations. for example, the
following statements
c      will compute the location and size of the maximum
residual in res
c      on cray computers:
c
c          ij = isamax(nx*ny,res,1)
c
c          jmax = (ij-1)/nx + 1
c
c          imax = ij - (jmax-1)*nx
c
c          resmax = abs(res(imax,jmax))
c
c
c *** please note:
c
c      let pe be the exact continuous solution to
the elliptic pde
c          evaluated on the nx by ny discretization grid
c
c      let p be the exact solution to the linear
discretization
c
c      let phi be the approximation to p generated
by the mudpack solver
c
c      then discretization level error is defined by the
condition
c
c          || phi - p || < || p - pe ||.
c                      =
c
c      a common measure of multigrid efficiency is that
discretization level
c      error is reached in one full multigrid cycle (see
references [2,9] in

```

```

c      the mudpack file "readme") .  this can happen
before the residual is
c      reduced to the level of roundoff error.
consequently, || res || is
c      a conservative measure of accuracy which can be
wasteful if multi-
c      grid cycles are executed until it reaches the
level of roundoff error.
c
c      || res || can be used to estimate the convergence
rate of multigrid
c      iteration.  let r(n) be the residual and e(n) be
the error after
c      executing n cycles.  they are related by the
residual equation
c
c          l * e(n) = r(n).
c
c      it follows that the ratio
c
c          || r(n+1) || / || r(n) ||
c
c      estimates
c
c          || e(n+1) || / || e(n) ||
c
c      which in turn estimates the convergence rate
c
c          c = max_k || e(k+1) || / || e(k) ||.
c
c      notice
c
c          || e(n) || <_n c || e(0) ||.
c
c
c ... assumptions (see muh2cr.d or mud2cr.d)
c
c      (1) nx,ny have the same values as
iparm(10),iparm(11) (used
c          to set the fine grid resolution when calling
cud2cr or cuh2cr)
c
c      (2) work is the same argument used in calling

```

```

cud2cr or cuh2cr.

c
c      (3) work has not changed since the last call to
cud2cr or cuh2cr.
c
c      if these assumptions are not true then resc2cr
c cannot compute the
c      residual in res. (3) assures a copy of the last
c approximation phi
c      is contained in work.
c
c
      subroutine resm2cr(nx,ny,work,res)
      implicit none
      integer nx,ny,ic
      real work(*),res(nx,ny)
c
c      set pointer for fine grid coefficients in work
c
      ic = 1+(nx+2)*(ny+2)
      call rem2cr(nx,ny,work,work(ic),res)
      return
      end

      subroutine rem2cr(nx,ny,phi,cof,res)
c
c      compute residual
c
      implicit none
      integer nx,ny,i,j
      real phi(0:nx+1,0:ny+1),cof(nx,ny,10),res(nx,ny)
      do j=1,ny
      do i=1,nx
         res(i,j) = cof(i,j,10) - (cof(i,j,9)*phi(i,j) +
         +                      cof(i,j,1)*phi(i+1,j) +
         cof(i,j,2)*phi(i+1,j+1) +
         +                      cof(i,j,3)*phi(i,j+1) +
         cof(i,j,4)*phi(i-1,j+1) +
         +                      cof(i,j,5)*phi(i-1,j) +
         cof(i,j,6)*phi(i-1,j-1) +
         +                      cof(i,j,7)*phi(i,j-1) +
         cof(i,j,8)*phi(i+1,j-1) )
      end do
      end do

```

```
    return  
end
```

---

## RESM2SP

```
C  
C      file resm2sp.f  
C  
C      * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C      *  
*  
C      *          copyright (c) 2008 by UCAR  
*  
C      *  
*  
C      *          University Corporation for Atmospheric  
Research           *  
C      *  
*  
C      *          all rights reserved  
*  
C      *  
*  
C      *          MUDPACK version 5.0.1  
*  
C      *  
*  
C      *          A Fortran Package of Multigrid  
*  
C      *  
*  
C      *          Subroutines and Example Programs  
*  
C      *  
*  
C      *          for Solving Elliptic Partial Differential  
Equations       *
```

```
C      *
*
C      *                                by
*
C      *
*
C      *                                John Adams
*
C      *
*
C      *                                of
*
C      *
*
C      *                                the National Center for Atmospheric
Research          *
C      *
*
C      *                                Boulder, Colorado (80307)
U.S.A.          *
C      *
*
C      *                                which is sponsored by
*
C      *
*
C      *                                the National Science Foundation
*
C      *
*
C      *      *      *      *      *      *      *      *      *      *      *
*      *      *      *      *      *
C
C      subroutine resm2sp(nx,ny,nxa,nxb,nyc,nyd,work,res)
C
C
C ... purpose
C
C
C      subroutine resm2sp computes the fine grid residual
in the nx by ny array
C      res after calling mud2sp.  if
C
C              l * p = f
```

```

c
c      is the n by n (n = nx*ny) block tri-diagonal
linear system resulting
c      from the pde discretization (done internally in
mud2sp) and phi is the
c      approximation to p obtained by calling mud2sp,
then resm2sp computes the
c      nx by ny residual array
c
c          res = f - l * phi.
c
c      one of the vector norms of res,
c
c          || res ||
c
c      can be computed as a "measure" of how well phi
satisfies the
c      discretization equations. for example, the
following statements
c      will compute the location and size of the maximum
residual in res
c      on cray computers:
c
c          ij = isamax(nx*ny,res,1)
c
c          jmax = (ij-1)/nx + 1
c
c          imax = ij - (jmax-1)*nx
c
c          resmax = abs(res(imax,jmax))
c
c
c *** please note:
c
c      let pe be the exact continuous solution to
the elliptic pde
c      evaluated on the nx by ny discretization grid
c
c      let p be the exact solution to the linear
discretization
c
c      let phi be the approximation to p generated
by the mudpack solver
c

```

```

c      then discretization level error is defined by the
condition
c
c      || phi - p || < || p - pe || .
c          =
c
c      a common measure of multigrid efficiency is that
discretization level
c      error is reached in one full multigrid cycle (see
references [2,9] in
c      the mudpack file "readme"). this can happen
before the residual is
c      reduced to the level of roundoff error.
consequently, || res || is
c      a conservative measure of accuracy which can be
wasteful if multi-
c      grid cycles are executed until it reaches the
level of roundoff error.
c
c      || res || can be used to estimate the convergence
rate of multigrid
c      iteration. let r(n) be the residual and e(n) be
the error after
c      executing n cycles. they are related by the
residual equation
c
c      l * e(n) = r(n) .
c
c      it follows that the ratio
c
c      || r(n+1) || / || r(n) ||
c
c      estimates
c
c      || e(n+1) || / || e(n) ||
c
c      which in turn estimates the convergence rate
c
c      c = max_k || e(k+1) || / || e(k) || .
c
c      notice
c
c      || e(n) || < c^n || e(0) || .

```

```

c
c
c ... assumptions (see mud2sp.d)
c
c      (1) nx,ny have the same values as
iparm(10),iparm(11) (used
c          to set the fine grid resolution when calling
mud2sp)
c
c      (2) nxa,nxb,nyc,nyd have the same values as
iparm(2),iparm(3),
c          iparm(4),iparm(5) (boundary condition flags)
used to call
c          mud2sp
c
c      (3) work is the same work space argument used in
calling mud2sp.
c
c      (4) work has not changed since the last call to
mud2sp.
c
c      If (1)-(4) are not true then resm2sp cannot
compute the residual
c      in res. (3),(4) assure a copy of the last
computed phi is in work.
c
subroutine resm2sp(nx,ny,nxa,nxb,nyc,nyd,work,res)
implicit none
integer nx,ny,nxa,nxb,nyc,nyd,irh,icx,icy
real work(*),res(nx,ny)
c
c      set pointer for fine grid coefficients in work
c
irh = 1 + (nx+2)*(ny+2)
icx = irh + nx*ny
icy = icx + 3*nx
call rem2sp(nx,ny,nxa,nxb,nyc,nyd,work,work(irh),
+           work(icx),work(icy),res)
return
end

subroutine
rem2sp(nx,ny,nxa,nxb,nyc,nyd,phi,rhs,cofx,cofy,res)
implicit none

```

```

integer nx,ny,nxa,nxb,nyc,nyd,i,j,ist,ifn,jst,jfn
real phi(0:nx+1,0:ny+1),rhs(nx,ny)
real cofx(nx,3),cofy(ny,3),res(nx,ny)

C
C      initialize residual to zero and set limits
C
do j=1,ny
do i=1,nx
  res(i,j) = 0.0
end do
  end do

C
C      set limits
C
ist = 1
if (nxa.eq.1) ist = 2
ifn = nx
if (nxb.eq.1) ifn = nx-1
jst = 1
if (nyc.eq.1) jst = 2
jfn = ny
if (nyd.eq.1) jfn = ny-1

C
C      compute residual on nonspecified grid points
C
do j=jst,jfn
do i=ist,ifn
  res(i,j) = rhs(i,j)-
    + cofx(i,1)*phi(i-1,j)+ 
    + cofx(i,2)*phi(i+1,j)+ 
    + cofy(j,1)*phi(i,j-1)+ 
    + cofy(j,2)*phi(i,j+1)+ 
    + (cofx(i,3)+cofy(j,3))*phi(i,j))
end do
  end do
return
end

```

---

## RESM3

```
C
C      file resm3.f
C
C      * * * * *
* * * * *
C      *
*
C      *           copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *           all rights reserved
*
C      *
*
C      *           MUDPACK version 5.0.1
*
C      *
*
C      *           A Fortran Package of Multigrid
*
C      *
*
C      *           Subroutines and Example Programs
*
C      *
*
C      *           for Solving Elliptic Partial Differential
Equations          *
C      *
*
C      *           by
*
C      *
*
C      *           John Adams
*
C      *
```

```
C      *
      of
*
C      *
*
C      *           the National Center for Atmospheric
Research          *
C      *
*
C      *           Boulder, Colorado (80307)
U.S.A.          *
C      *
*
C      *           which is sponsored by
*
C      *
*
C      *           the National Science Foundation
*
C      *
*
C      *           ****
* * * * * * *
C
C      subroutine resm3(nx,ny,nz,work,res)
C
C
C ... purpose
C
C
C      subroutine resm3 computes the fine grid residual
in the nx by ny by nz
c      array res after calling muh3 or mud3 or mud3sa.
if
C
C      L * p = f
C
C      is the n by n (n = nx*ny*nz) block tri-diagonal
linear system resulting
c      from the pde discretization (done internally in
mud3 or mud3sa) and phi
c      is the approximation to p obtained by calling
mud3, then resm3 computes
c      the nx by ny by nz residual array
C
```

```

c           res = f - L * phi.
c
c           one of the vector norms of res,
c
c           || res ||
c
c           can be computed as a "measure" of how well phi
satisfies the
c           discretization equations. for example, the
following statements
c           will compute the location and size of the maximum
residual in res
c           on cray computers:
c
c           ijk = isamax(nx*ny*nz,res,1)
c
c           kmax = (ijk-1)/(nx*ny) + 1
c
c           jmax = (ijk-(kmax-1)*nx*ny-1)/nx + 1
c
c           imax = ij - nx*((kmax-1)*ny-jmax+1)
c
c           resmax = abs(res(imax,jmax,kmax))
c
c
c *** please note:
c
c           let pe be the exact continuous solution to
the elliptic pde
c           evaluated on the nx by ny by nz
discretization grid
c
c           let p be the exact solution to the linear
discretization
c
c           let phi be the approximation to p generated
by the mudpack solver
c
c           then discretization level error is defined by the
condition
c
c           || phi - p || < || p - pe ||
c           =
c

```

```

c      a common measure of multigrid efficiency is that
discretization level
c      error is reached in one full multigrid cycle (see
references [2,9] in
c      the mudpack file "readme"). this can happen
before the residual is
c      reduced to the level of roundoff error.
consequently, || res || is
c      a conservative measure of accuracy which can be
wasteful if multi-
c      grid cycles are executed until it reaches the
level of roundoff error.

c
c      || res || can be used to estimate the convergence
rate of multigrid
c      iteration. let r(n) be the residual and e(n) be
the error after
c      executing n cycles. they are related by the
residual equation
c
c      L * e(n) = r(n).
c
c      it follows that the ratio
c
c      || r(n+1) || / || r(n) ||
c
c      estimates
c
c      || e(n+1) || / || e(n) ||
c
c      which in turn estimates the convergence rate
c
c      c = max || e(k+1) || / || e(k) ||.
c          k
c
c      notice
c
c          n
c      || e(n) || < c || e(0) ||.
c
c
c ... assumptions (see mud3.d or mud3sa.d)
c
c      (1) nx,ny,nz have the same values as
iparm(14),iparm(15),iparm(16)

```

```

c      (used to set the fine grid resolution when
calling mud3 or mud3sa)
c
c      (2) work,phi are the same parameters used in
calling mud3 or mud3sa.
c
c      (3) work,phi have not changed since the last call
to mud3 or mud3sa.
c
c
c      (3) assures a copy of the last approximation phi
is contained in work.
c      if these assumptions are not true then resm3
cannot compute the
c      residual in res.
c
c      subroutine resm3(nx,ny,nz,work,res)
c
c      compute fine grid residual in res after calling
mud3 or mud3sa
c
c      implicit none
      integer nx,ny,nz,ic
      real work(*),res(nx,ny,nz)
c
c      set coefficient pointer
c
      ic = 1+(nx+2)*(ny+2)*(nz+2)
      ic = 1+(nx+2)*(ny+2)*(nz+2)
      call rem3(nx,ny,nz,work,work(ic),res)
      return
      end

      subroutine rem3(nx,ny,nz,phi,cof,res)
      implicit none
      integer nx,ny,nz,i,j,k
      real
cof(nx,ny,nz,8),phi(0:nx+1,0:ny+1,0:nz+1),res(nx,ny,nz)
      do k=1,nz
      do j=1,ny
      do i=1,nx
          res(i,j,k) = cof(i,j,k,8) - (
          + cof(i,j,k,1)*phi(i-1,j,k)+
          + cof(i,j,k,2)*phi(i+1,j,k)+
```

```

+
+           cof(i,j,k,3)*phi(i,j-1,k) +
+           cof(i,j,k,4)*phi(i,j+1,k) +
+           cof(i,j,k,5)*phi(i,j,k-1) +
+           cof(i,j,k,6)*phi(i,j,k+1) +
+           cof(i,j,k,7)*phi(i,j,k) )
end do
end do
end do
return
end

```

RESM3SP

```
C      file resm3sp.f
C
C      * * * * * * * * * * * * * * * * * * * * * *
*
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *               University Corporation for Atmospheric
Research          *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK version 5.0.1
*
C      *
*
C      *               A Fortran Package of Multigrid
```

```
C      *
*
C      *                               Subroutines and Example Programs
*
C      *
*
C      *               for Solving Elliptic Partial Differential
Equations      *
C      *
*
C      *                               by
*
C      *
*
C      *                               John Adams
*
C      *
*
C      *                               of
*
C      *
*
C      *               the National Center for Atmospheric
Research      *
C      *
*
C      *               Boulder, Colorado (80307)
U.S.A.      *
C      *
*
C      *               which is sponsored by
*
C      *
*
C      *               the National Science Foundation
*
C      *
*
C      *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *
*   *   *   *   *   *   *
C
C      subroutine
resm3sp(nx,ny,nz,nxa,nxb,nyc,nyd,nze,nzf,work,res)
C
```

```

c
c ... purpose
c
c
c      subroutine resm3sp computes the fine grid residual
c      in the nx by ny by nz
c      array res after calling mud3sp.  if
c
c          l * p = f
c
c      is the n by n (n = nx*ny*nz) block tri-diagonal
c      linear system resulting
c      from the pde discretization (done internally in
c      mud3sp) and phi is the
c      approximation to p obtained by calling mud3sp,
c      then resm3sp computes
c          the nx by ny by nz residual array
c
c          res = f - l * phi.
c
c          one of the vector norms of res,
c
c          || res ||
c
c      can be computed as a "measure" of how well phi
c      satisfies the
c      discretization equations.  for example, the
c      following statements
c      will compute the location and size of the maximum
c      residual in res
c      on cray computers:
c
c          ijk = isamax(nx*ny*nz,res,1)
c
c          kmax = (ijk-1) / (nx*ny) + 1
c
c          jmax = (ijk-(kmax-1)*nx*ny-1)/nx + 1
c
c          imax = ij - nx*((kmax-1)*ny-jmax+1)
c
c          resmax = abs(res(imax,jmax,kmax))
c
c
c *** please note:

```

```

c
c           let pe be the exact continuous solution to
the elliptic pde
c           evaluated on the nx by ny by nz
discretization grid
c
c           let p be the exact solution to the linear
discretization
c
c           let phi be the approximation to p generated
by the mudpack solver
c
c           then discretization level error is defined by the
condition
c
c           || phi - p || < || p - pe ||
c           =
c
c           a common measure of multigrid efficiency is that
discretization level
c           error is reached in one full multigrid cycle (see
references [2,9] in
c           the mudpack file "readme"). this can happen
before the residual is
c           reduced to the level of roundoff error.
consequently, || res || is
c           a conservative measure of accuracy which can be
wasteful if multi-
c           grid cycles are executed until it reaches the
level of roundoff error.
c
c           || res || can be used to estimate the convergence
rate of multigrid
c           iteration. let r(n) be the residual and e(n) be
the error after
c           executing n cycles. they are related by the
residual equation
c
c           l * e(n) = r(n).
c
c           it follows that the ratio
c
c           || r(n+1) || / || r(n) ||
c

```

```

c      estimates
c
c      || e(n+1) || / || e(n) ||
c
c      which in turn estimates the convergence rate
c
c      c = max_k || e(k+1) || / || e(k) ||.
c
c
c      notice
c
c      || e(n) || < cn || e(0) ||.
c
c
c ... assumptions (see mud3sp.d)
c
c      (1) nx,ny,nz have the same values as
iparm(14),iparm(15),iparm(16)
c          (used to set the fine grid resolution when
calling mud3sp)
c
c      (2) nxa,nxb,nyc,nyd,nze,nzf have the same values
as iparm(2),
c          iparm(3),iparm(4),iparm(5),iparm(6),iparm(7)
(used to flag
c          boundary conditions when calling mud3sp).
c
c      (3) work,phi are the same parameters used in
calling mud3sp.
c
c      (4) work,phi have not changed since the last call
to mud3sp.
c
c      (3) assures a copy of the last approximation phi
is contained in work.
c      if (1)-(4) are not true then resm3sp cannot
compute the residual.
c
      subroutine
resm3sp(nx,ny,nz,nxa,nxb,nyc,nyd,nze,nzf,wk,res)
      implicit none
      integer
nx,ny,nz,nxa,nxb,nyc,nyd,nze,nzf,ir,ix,iy,iz
      real wk(*),res(*)
```

```

C
C      set pointers
C
        ir = 1+(nx+2)*(ny+2)*(nz+2)
        ix = ir+nx*ny*nz
        iy = ix+3*nx
        iz = iy+3*ny
        call
rem3sp(nx,ny,nz,nxa,nxb,nyc,nyd,nze,nzf,wk,wk(ir),wk(ix)
,
        +           wk(iy),wk(iz),res)
        return
        end

        subroutine
rem3sp(nx,ny,nz,nxa,nxb,nyc,nyd,nze,nzf,phi,
        +           rhs,cofx,cofy,cofz,resf)
        implicit none
        integer nx,ny,nz,nxa,nxb,nyc,nyd,nze,nzf
        integer i,j,k,ist,ifn,jst,jfn,kst,kfn
        real cofx(nx,3),cofy(ny,3),cofz(nz,3)
        real
rhs(nx,ny,nz),phi(0:nx+1,0:ny+1,0:nz+1),resf(nx,ny,nz)
C
C      intialize residual to zero and set loop limits
C
        do k=1,nz
        do j=1,ny
        do i=1,nx
            resf(i,j,k) = 0.0
        end do
        end do
        end do
        ist = 1
        if (nxa.eq.1) ist = 2
        ifn = nx
        if (nxb.eq.1) ifn = nx-1
        jst = 1
        if (nyc.eq.1) jst = 2
        jfn = ny
        if (nyd.eq.1) jfn = ny-1
        kst = 1
        if (nze.eq.1) kst = 2
        kfn = nz

```

```

if (nzf.eq.1) kfn = nz-1
C
C      compute residual
C
      do k=kst,kfn
      do j=jst,jfn
      do i=ist,ifn
          resf(i,j,k) =  rhs(i,j,k)-
          +      cofx(i,1)*phi(i-
1,j,k)+cofx(i,2)*phi(i+1,j,k) +
          +      cofy(j,1)*phi(i,j-
1,k)+cofy(j,2)*phi(i,j+1,k) +
          +      cofz(k,1)*phi(i,j,k-
1)+cofz(k,2)*phi(i,j,k+1) +
          +      (cofx(i,3)+cofy(j,3)+cofz(k,3))*phi(i,j,k)
      )
      end do
      end do
      end do
      return
      end

```

## TCUD2

```

C
C      file tcud2.f
C
C      * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *                               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research           *
C      *

```

```
*  
C      *          all rights reserved  
*  
C      *  
*  
C      *          MUDPACK  version 5.0.1  
*  
C      *  
*  
C      *          A Fortran Package of Multigrid  
*  
C      *  
*  
C      *          Subroutines and Example Programs  
*  
C      *  
*  
C      *          for Solving Elliptic Partial Differential  
Equations      *  
C      *  
*  
C      *          by  
*  
C      *  
*  
C      *          John Adams  
*  
C      *  
*  
C      *          of  
*  
C      *  
*  
C      *          the National Center for Atmospheric  
Research      *  
C      *  
*  
C      *          Boulder, Colorado (80307)  
U.S.A.      *  
C      *  
*  
C      *          which is sponsored by  
*  
C      *
```

```
*
```

```
C      *          the National Science Foundation
```

```
*
```

```
C      *
```

```
*
```

```
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
* * * * * * * *
```

```
C
```

```
C ... purpose
```

```
C
```

```
C      test program for the complex MUDPACK solver cud2
```

```
C
```

```
C ... required MUDPACK files
```

```
C
```

```
C      cud2.f, cudcom.f
```

```
C
```

```
C
```

```
C
```

```
*****
```

```
*
```

```
C
```

```
*****
```

```
*
```

```
C
```

```
C      sample program/test driver for cud2
```

```
C
```

```
C
```

```
*****
```

```
**
```

```
C
```

```
*****
```

```
**
```

```
C
```

```
C      a sample program/test driver for cud2 is below.
```

```
it can be
```

```
C      executed as an initial test. the output is listed
```

```
for the
```

```
C      test case described.
```

```
C
```

```
C      test the driver below by solving the complex
```

```
linear elliptic pde
```

```
C
```

```
C      cmplx(1.+x*x,1.+y*y)*pxx +
```

```
C
```

```

c      cmplx(exp(-x),exp(-y))* (pyy - py) +
c
c      cmplx(y,x)*p(x,y) = r(x,y)
c
c      on the unit square with specified boundary
c      conditions at
c      xb = 1.0, yc = 1.0 and mixed boundary conditions
c
c      dp/dx - cmplx(y,y)*p(xa,y) = g(y) at x = xa
c      and
c
c      dp/dy + cmplx(x,x)*p(x,yd) = h(x) at y = yd.
c
c      the exact solution
c
c      p(x,y) = cmplx(x**5,y**5) + 1.0
c
c      is used for testing. one full multigrid cycle (no
c      initial guess)
c      with red/black gauss-seidel point relaxation and
c      the default multigrid
c      options is sufficient to reach discretization
c      level error on a
c      49 x 65 grid.
c
c
c ****
c      output (32 bit floating point arithmetic)
c
c ****
c
c      cud2 test
c
c      integer input arguments
c      intl = 0 nxa = 2 nxb = 1 nyc = 1 nyd = 2
c      ixp = 3 jyq = 2 iex = 5 jey = 6
c      nx = 49 ny = 65 iguess = 0 maxcy = 1
c      method = 0 work space estimate = 42776
c
c      multigrid option arguments
c      kcycle = 2
c      iprer = 2
c      ipost = 1
c      interpol = 3

```

```

C
C      floating point input parameters
C      xa =  0.000 xb =  1.000 yc =  0.000 yd =  1.000
C      tolerance (error control) =      0.000E+00
C
C      discretization call to cud2 intl =  0
C      ierror =  0 minimum work space =  34192
C
C      approximation call to cud2
C      intl =  1 method =  0 iguess =  0
C      ierror =  0
C      maximum error =  0.615E-03
C
C
C ****
C      end of output
C
C ****
C ****
C
C      program tcud2
C      implicit none
C      integer iixp,jjyq,iiex,jjey,nnx,nny,llw
C
C      set grid sizes with parameter statements
C
C      parameter (iixp = 3 , jjyq = 2 , iiex =5, jjey =
6)
C      parameter (nnx=iixp*2** (iiex-1)+1,
C      nny=jjyq*2** (jjey-1)+1)
C
C      estimate work length approximation for method=0
C      (see cud2.d)
C
C      parameter (llw=(40*nnx*nny+8* (nnx+nny+2)) /3)
C
C      dimension solution,right hand side, and work
C      arrays
C
C      complex p(nnx,nny),r(nnx,nny),w(llw)
C      put integer and floating point parameter names in
C      contiguous
C      storeage for labelling purposes

```

```

c
      integer iprm(16),mgopt(4)
      real fprm(6)
      integer
      intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny,
      +
      iguess,maxcy,method,nwork,lwrkqd,itero

common/itcud2/intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny
,
      +
      iguess,maxcy,method,nwork,lwrkqd,itero
      real xa,xb,yc,yd,tolmax,relmax
      common/ftcud2/xa,xb,yc,yd,tolmax,relmax
      equivalence(intl,iprm)
      equivalence(xa,fprm)
      integer i,j,ierror
      complex pe,px,py,pxx,pyy,cxx,cyy,cx,cy,ce
      real dlx,dly,x,y,errmax

c
c      declare coefficient and boundary condition input
subroutines external
c
      external cof,bndc
c
c      set input integer arguments
c
      intl = 0
c
c      set boundary condition flags
c
      nxa = 2
      nxb = 1
      nyc = 1
      nyd = 2
c
c      set grid sizes from parameter statements
c
      ixp = iixp
      jyq = jjyq
      iex = iiex
      jey = jjey
      nx = nnx

```

```

ny = nny
c
c      set for one multigrid cycle
c
maxcy = 1
c
c      set work space length approximation from parameter
statement
c
nwork = llw
c
c      set point relaxation
c
method = 0
c
c      flag no initial guess (this sets full multigrid
cycling)
c
iguess = 0
c
c      set end points of solution rectangle in (x,y)
space
c
xa = 0.0
xb = 1.0
yc = 0.0
yd = 1.0
c
c      set mesh increments
c
dlx = (xb-xa)/float(nx-1)
dly = (yd-yc)/float(ny-1)
c
c      set for no error control
c
tolmax = 0.0
c
c      set right hand side in r
c      initialize p to zero
c
do i=1,nx
x = xa+float(i-1)*dlx
do j=1,ny
y = yc+float(j-1)*dly

```

```

call cof(x,y,cxx,cyy,cx,cy,ce)
call exact(x,y,pxx,pyy,px,py,pe)
r(i,j) = cxx*pxx+cyy*pyy+cx*px+cy*py+ce*pe
p(i,j) = (0.0,0.0)
end do
    end do
c
c      set specified boundaries in p
c
x = xb
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,pxx,pyy,px,py,pe)
p(nx,j) = pe
end do
y = yc
do i=1,nx
x = xa+float(i-1)*dlx
call exact(x,y,pxx,pyy,px,py,pe)
p(i,1) = pe
end do
c
c      set default multigrid options
c
mgopt(1) = 2
mgopt(2) = 2
mgopt(3) = 1
mgopt(4) = 3
c
c      print input parameters (except multigrid options
which are default)
c
write(6,100)
100 format(//' cud2 test ')
        write (*,101) (iprm(i),i=1,15)
101 format(/' integer input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =
',i2,' nyd = ',i2,
+/' ixp = ',i2,' jyq = ',i2,' iex = ',i2,' jey =
',i2
+/' nx = ',i3,' ny = ',i3,' iguess = ',i2,' maxcy =
',i2,
+/' method = ',i2, ' work space estimate = ',i7)
        write (*,102) (mgopt(i),i=1,4)

```

```

102 format(/' multigrid option arguments ',
+/' kcycle = ',i2,
+/' iprer = ',i2,
+/' ipost = ',i2
+/' interpol = ',i2)
      write(*,103) xa,xb,yc,yd,tolmax
103 format(/' floating point input parameters ',
+/' xa = ',f6.3,' xb = ',f6.3,' yc = ',f6.3,' yd =
',f6.3,
+/' tolerance (error control) =    ',e10.3)
C
C      initialization call
C
      write(*,104) intl
104 format(/' discretization call to cud2', ' intl =
', i2)
      call cud2(iprm,fprm,w,cof,bndc,r,p,mgopt,ierror)
C
C      print error parameter and minimum work space
requirement
C
      write (*,105) ierror,iprm(16)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)
C
C      attempt solution
C
      intl = 1
      write(*,106) intl,method,iguess
106 format(/' approximation call to cud2',
+/' intl = ',i2, ' method = ',i2,' iguess = ',i2)
      call cud2(iprm,fprm,w,cof,bndc,r,p,mgopt,ierror)
      write (*,107) ierror
107 format(' ierror = ',i2)
C
C      compute and print exact maximum error
C
      errmax = 0.0
      do j=1,ny
      y = yc+(j-1)*dly
      do i=1,nx
      x = xa+(i-1)*dlx
      call exact(x,y,pxx,pyy,px,py,pe)

```

```

errmax = amax1(errmax,cabs( (p(i,j)-pe) ))
end do
    end do
    write(*,108) errmax
108 format(' maximum error = ',e10.3)
    end

    subroutine cof(x,y,cxx,cyy,cx,cy,ce)
c
c      input pde coefficients at any grid point (x,y) in
the solution region
c      (xa.le.x.le.xb,yc.le.y.le.yd) to cud2
c
        implicit none
        real x,y
        complex cxx,cyy,cx,cy,ce
        cxx = cmplx(1.+x*x,1.+y*y)
        cyy = cmplx(exp(-x),exp(-y))
        cx = (0.,0.)
        cy = -cyy
        ce = -cmplx(y,x)
        return
        end

    subroutine bndc(kbdy,xory,alfa,gbdy)
c
c      input mixed derivative b.c. to cud2
c
        implicit none
        integer kbdy
        real xory,x,y
        complex alfa,gbdy
        real xa,xb,yc,yd,tolmax,relmax
        common/ftcud2/xa,xb,yc,yd,tolmax,relmax
        complex pe,px,py,pxx,pyy
        if (kbdy.eq.1) then
c
c      x=xa boundary (nxa must equal 2)
c      b.c. has the form px + alfxa(y)*pe = gbdxa(y)
c      alfa and gbdy corresponding to alfxa(y),gbdxa(y)
c      must be output
c
        y = xory
        x = xa

```

```

call exact(x,y,pxx,pyy,px,py,pe)
alfa = -cmplx(y,y)
gbdy = px + alfa*pe
return
end if
if (kbdy.eq.4) then
C
C      y = yd boundary (nyd must equal 2)
C      b.c. has the form py + alfyd(x)*pe = gbdy(x)
C      alfa and gbdy corresponding to alfyd(x),gbdy(x)
C      must be output
C
y = yd
x = xory
call exact(x,y,pxx,pyy,px,py,pe)
alfa = cmplx(x,x)
gbdy = py + alfa*pe
return
end if
end

subroutine exact(x,y,pxx,pyy,px,py,pe)
C
C      this subroutine is used to set an exact solution
for testing cud2
C
implicit none
real x,y
complex pxx,pyy,px,py,pe
pe = cmplx(x**5,y**5)+1.0
px = cmplx(5*x**4,0.)
py = cmplx(0.,5*y**4)
pxx = cmplx(20.*x**3,0.)
pyy = cmplx(0.,20.*y**3)
return
end

```

```
C
C      file tcud24.f
C
C      * * * * *
* * * * *
C      *
*
C      *           copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *           all rights reserved
*
C      *
*
C      *           MUDPACK version 5.0.1
*
C      *
*
C      *           A Fortran Package of Multigrid
*
C      *
*
C      *           Subroutines and Example Programs
*
C      *
*
C      *           for Solving Elliptic Partial Differential
Equations          *
C      *
*
C      *           by
*
C      *
*
C      *           John Adams
*
C      *
```

C \* of  
\*  
C \*  
\*  
C \* the National Center for Atmospheric  
Research \*  
C \*  
\*  
C \* Boulder, Colorado (80307)  
U.S.A. \*  
C \*  
\*  
C \* which is sponsored by  
\*  
C \*  
\*  
C \* the National Science Foundation  
\*  
C \*  
\*  
C \*  
\* \* \* \* \* \* \* \*  
C  
C ... purpose  
C  
C test program for the complex MUDPACK solver cud2  
C  
C ... required MUDPACK files  
C  
C cud2.f, cudcom.f  
C  
C  
C \*\*\*\*\*  
\*  
C \*\*\*\*\*  
\*  
C \*\*\*\*\*  
\*  
C sample program/test driver for cud2  
C  
C  
\*\*\*\*\*  
\*\*

```

C
*****
** 
C
c      a sample program/test driver for cud2 is below.
it can be
c      executed as an initial test.  the output is listed
for the
c      test case described.
C
c      test the driver below by solving the complex
linear elliptic pde
C
c      cmplx(1.+x*x,1.+y*y)*pxx +
C
c      cmplx(exp(-x),exp(-y))* (pyy - py) +
C
c      cmplx(y,x)*p(x,y) = r(x,y)
C
c      on a 49 X 65 grid superimposed on the unit square
with
c      specified boundary conditions at xb=1.0, yc=1.0
and
c      mixed boundary conditions
C
c      dp/dx - cmplx(y,y)*p(xa,y) = g(y) at x = xa
C      and
C
c      dp/dy + cmplx(x,x)*p(x,yd) = h(x) at y = yd.
C
c      the exact solution
C
c      p(x,y) = cmplx(x**5,y**5) + 1.0
C
c      is used for testing.  three multigrid cycles (no
initial guess)
c      with red/black gauss-seidel point relaxation and
the default multigrid
c      options are used to compute the second-order
approximation with cud2.
c      Then cud24 is called to produce a fourth order
estimate.
C
C ****

```

```

C      output (64 bit floating point arithmetic)
C
*****
C
C      cud2 test
C
C      integer input arguments
C      intl = 0 nxa = 2 nxb = 1 nyc = 1 nyd = 2
C      ixp = 3 jyq = 2 iex = 5 jey = 6
C      nx = 49 ny = 65 iguess = 0 maxcy = 3
C      method = 0 work space estimate = 34192
C
C      multigrid option arguments
C      kcycle = 2
C      iprer = 2
C      ipost = 1
C      interpol = 3
C
C      floating point input parameters
C      xa = 0.000 xb = 1.000 yc = 0.000 yd = 1.000
C      tolerance (error control) = 0.000E+00
C
C      discretization call to cud2 intl = 0
C      ierror = 0 minimum work space = 34192
C
C      approximation call to cud2
C      intl = 1 method = 0 iguess = 0 maxcy = 3
C      ierror = 0
C      maximum error = 0.640E-03
C
C      cud24 test ierror = 0
C      maximum error = 0.346E-05
C
C
*****
C      end of output
C
*****
C
C      program tcud24
C      implicit none
C      integer iixp,jjyq,iiex,jjey,nx,ny,llw

```

```

C
C      set grid sizes with parameter statements
C
C      parameter (iixp = 3 , jjyq = 2 , iiex =5, jjey =
6)
C          parameter (nnx=iixp*2** (iiex-1)+1,
C                      nny=jjyq*2** (jjey-1)+1)
C
C      set exact minimal work space required (see
C      tcud2.f)
C
C      parameter (llw=34192)
C
C      dimension solution,right hand side, and work
C      arrays
C
C      complex p(nnx,nny),r(nnx,nny),w(llw)
C      put integer and floating point parameter names in
C      contiguous
C      storeage for labelling purposes
C
C      integer iprm(16),mgopt(4)
C      real fprm(6)
C      integer
C      intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny,
C      +
C      iguess,maxcy,method,nwork,lwrkqd,itero
C
common/itcud2/intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny
,
+
C      iguess,maxcy,method,nwork,lwrkqd,itero
C      real xa,xb,yc,yd,tolmax,relmax
C      common/ftcud2/xa,xb,yc,yd,tolmax,relmax
C      equivalence(intl,iprm)
C      equivalence(xa,fprm)
C      integer i,j,ierror
C      complex cxx,cyy,cx,cy,ce,pxx,pyy,px,py,pe
C      real dlx,dly,x,y,errmax
C
C      declare coefficient and boundary condition input
C      subroutines external
C
C      external cof,bndc

```

```
C
C
C      set input integer arguments
C
C      intl = 0
C
C      set boundary condition flags
C
C      nxa = 2
C      nxb = 1
C      nyc = 1
C      nyd = 2
C
C      set grid sizes from parameter statements
C
C      ixp = iixp
C      jyq = jjyq
C      iex = iiex
C      jey = jjey
C      nx = nnx
C      ny = nny
C
C      set for three multigrid cycle
C
C      maxcy = 3
C
C      set work space length approximation from parameter
C      statement
C
C      nwork = llw
C
C      set point relaxation
C
C      method = 0
C
C      flag no initial guess (this sets full multigrid
C      cycling)
C
C      iguess = 0
C
C      set end points of solution rectangle in (x,y)
C      space
C
C      xa = 0.0
```

```

    xb = 1.0
    yc = 0.0
    yd = 1.0
C
C      set mesh increments
C
C      dlx = (xb-xa)/float(nx-1)
C      dly = (yd-yc)/float(ny-1)
C
C      set for no error control
C
C      tolmax = 0.0
C
C      set right hand side in r
C      initialize p to zero
C
do i=1,nx
x = xa+float(i-1)*dlx
do j=1,ny
  y = yc+float(j-1)*dly
  call cof(x,y,cxx,cyy,cx,cy,ce)
  call exact(x,y,pxx,pyy,px,py,pe)
  r(i,j) = cxx*pxx+cyy*pyy+cx*px+cy*py+ce*pe
  p(i,j) = (0.0,0.0)
end do
  end do
C
C      set specified boundaries in p
C
x = xb
do j=1,ny
  y = yc+float(j-1)*dly
  call exact(x,y,pxx,pyy,px,py,pe)
  p(nx,j) = pe
  end do
  y = yc
  do i=1,nx
x = xa+float(i-1)*dlx
  call exact(x,y,pxx,pyy,px,py,pe)
  p(i,1) = pe
  end do
C
C      set default multigrid options
C

```

```

mgopt(1) = 2
mgopt(2) = 2
mgopt(3) = 1
mgopt(4) = 3
C
C      print input parameters (except multigrid options
which are default)
C
      write(6,100)
100 format(//' cud2 test ')
      write (*,101) (iprm(i),i=1,15)
101 format(' integer input arguments ',
      +' intl = ',i2,' nxa = ',i2,' nxbo = ',i2,' nyc =
',i2,' nyd = ',i2,
      +' ixp = ',i2,' jyq = ',i2,' iex = ',i2,' jey =
',i2,
      +' nx = ',i3,' ny = ',i3,' iguess = ',i2,' maxcy =
',i2,
      +' method = ',i2, ' work space estimate = ',i7)
      write (*,102) (mgopt(i),i=1,4)
102 format(' multigrid option arguments ',
      +' kcycle = ',i2,
      +' iprer = ',i2,
      +' ipost = ',i2
      +' interpol = ',i2)
      write(*,103) xa,xb,yc,yd,tolmax
103 format(' floating point input parameters ',
      +' xa = ',f6.3,' xb = ',f6.3,' yc = ',f6.3,' yd =
',f6.3,
      +' tolerance (error control) =    ',e10.3)
C
C      initialization call
C
      write(*,104) intl
104 format(' discretization call to cud2', ' intl =
', i2)
      call cud2(iprm,fprm,w,cof,bndc,r,p,mgopt,ierror)
C
C      print error parameter and minimum work space
requirement
C
      write (*,105) ierror,iprm(16)
105 format(' ierror = ',i2, ' minimum work space =
',i7)

```

```

    if (ierror.gt.0) call exit(0)
C
C      attempt solution
C
    intl = 1
    write(*,106) intl,method,iguess,maxcy
106 format(/' approximation call to cud2',
     +' intl = ',i2, ' method = ',i2,' iguess = ',i2, '
maxcy = ',i2)
    call cud2(iprm,fprm,w,cof,bndc,r,p,mgopt,ierror)
    write (*,107) ierror
107 format(' ierror = ',i2)
C
C      compute and print exact maximum error
C
    errmax = 0.0
    do j=1,ny
        y = yc+(j-1)*dly
    do i=1,nx
        x = xa+(i-1)*dlx
        call exact(x,y,pxx,pyy,px,py,pe)
        errmax = amax1(errmax,cabs((p(i,j)-pe)))
    end do
    end do
    write(*,108) errmax
108 format(' maximum error = ',e10.3)
C
C      attempt to improve approximation to fourth order
C
    call cud24(w,p,ierror)
    write (*,109) ierror
109 format(/' cud24 test ', ' ierror = ',i2)
    if (ierror.gt.0) call exit(0)
    errmax = 0.0
    do j=1,ny
        y = yc+(j-1)*dly
    do i=1,nx
        x = xa+(i-1)*dlx
        call exact(x,y,pxx,pyy,px,py,pe)
        errmax = amax1(errmax,abs((p(i,j)-pe)))
    end do
    end do
    write(*,108) errmax
end

```

```

    subroutine cof(x,y,cxx,cyy,cx,cy,ce)
C
C      input pde coefficients at any grid point (x,y) in
the solution region
C      (xa.le.x.le.xb,yc.le.y.le.yd) to cud2
C
        implicit none
        real x,y
        complex cxx,cyy,cx,cy,ce
        cxx = cmplx(1.+x*x,1.+y*y)
        cyy = cmplx(exp(-x),exp(-y))
        cx = (0.,0.)
        cy = -cyy
        ce = -cmplx(y,x)
        return
        end

    subroutine bndc(kbdy,xory,alfa,gbdy)
C
C      input mixed derivative b.c. to cud2
C
        implicit none
        integer kbdy
        real xory,x,y
        complex alfa,gbdy
        real xa,xb,yc,yd,tolmax,relmax
        common/ftcud2/xa,xb,yc,yd,tolmax,relmax
        complex pe,px,py,pxx,pyy
        if (kbdy.eq.1) then
C
C      x=xa boundary (nxa must equal 2)
C      b.c. has the form px + alfxa(y)*pe = gbdxa(y)
C      alfa and gbdy corresponding to alfxa(y),gbdxa(y)
C      must be output
C
        y = xory
        x = xa
        call exact(x,y,pxx,pyy,px,py,pe)
        alfa = -cmplx(y,y)
        gbdy = px + alfa*pe
        return
        end if
        if (kbdy.eq.4) then

```

```

c      y = yd boundary (nyd must equal 2)
c      b.c. has the form py + alfyd(x)*pe = gbdy(x)
c      alfa and gbdy corresponding to alfyd(x),gbdy(x)
c      must be output
c

y = yd
x = xory
call exact(x,y,pxx,pyy,px,py,pe)
alfa = cmplx(x,x)
gbdy = py + alfa*pe
      return
      end if
      end

subroutine exact(x,y,pxx,pyy,px,py,pe)
c
c      this subroutine is used to set an exact solution
for testing cud2
c

implicit none
real x,y
complex pxx,pyy,px,py,pe
pe = cmplx(x**5,y**5)+1.0
px = cmplx(5*x**4,0.)
py = cmplx(0.,5*y**4)
pxx = cmplx(20.*x**3,0.)
pyy = cmplx(0.,20.*y**3)
      return
      end

```

TCUD24CR

```
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *               University Corporation for Atmospheric
Research          *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK version 5.0.1
*
C      *
*
C      *               A Fortran Package of Multigrid
*
C      *
*
C      *               Subroutines and Example Programs
*
C      *
*
C      *               for Solving Elliptic Partial Differential
Equations        *
C      *
*
C      *               by
*
C      *
*
C      *               John Adams
*
C      *
*
C      *               of
*
C      *
*
C      *               the National Center for Atmospheric
Research          *
```

```
C      *
*
C      *                      Boulder, Colorado (80307)
U.S.A.      *
C      *
*
C      *                      which is sponsored by
*
C      *
*
C      *                      the National Science Foundation
*
C      *
*
C      *      *      *      *      *      *      *      *      *      *      *
*      *      *      *      *      *      *      *      *      *      *      *
C
C ... purpose
C
C      test program for the complex MUDPACK solver
cud24cr
C
C ... required MUDPACK files
C
C      cud24cr.f, cud2cr.f, cudcom.f
C
C
*****
*
C
*****
*
C
*****
```

sample program/test driver for cud24cr

\*\*

C

\*\*\*\*\*

\*\*

C

a sample program/test driver for cud24cr is below.  
it can

```

c      be executed as an initial test.  the output is
listed for
c      the test case described.
c
c      test the driver below by solving the elliptic pde
c
c      cmplx(1.+y*y,1.-y*y)*pxx + cmplx(x*y,-x*y)*pxy +
c
c      cmplx(1.+x*x,1.-x*x)*pyy + cmplx(y,-y)*px +
c
c      cmplx(x,-x)*py + cmplx(x+y,-x-y)*p(x,y) = r(x,y)
c
c      on the unit square with specified boundary
conditions at
c      xa=0.0, xb=1.0, yc=1.0 and mixed boundary
conditions at yd=1.0
c      of the form:
c
c      cmplx(-x,x)*dp/dx + cmplx(1.+x,1.-x)*dp/dy +
c
c      cmplx(-x,x)*p(x,1.0) = gbdy(x)
c
c      choose a 49 x 65 grid and use line relaxation in
the y direction.
c      forr testing purposes use the exact solution
c
c      p(x,y) = cmplx((x*y)**3,-(x*y)**3) + (1.,1.)
c
c      First cud2cr is called with iguess=0 and maxcy=1
(see tcud2sp.f)
c      to execute one full multigrid cycle with no
initial guess.  Then
c      cud2cr is called again with iguess=1 and maxcy=2
to ensure second
c      order discretization level error is reached (a
requirement for
c      fourth order mudpack solvers).  Finally cud24cr is
called to
c      produce a fourth-order estimate.
c
c ****
c      output (64 bit floating point arithmetic)
c
*****

```

```
C
C      cud2cr test
C
C      integer input arguments
C      intl = 0 nxa = 1 nxb = 1 nyc = 1 nyd = 2
C      ixp = 3 jyq = 2 iex = 5 jey = 6
C      nx = 49 ny = 65 iguess = 0 maxcy = 1
C      method = 2 work space estimate = 64474
C
C      multigrid option arguments
C      kcycle = 2
C      iprer = 2
C      ipost = 1
C      interpol = 3
C
C      floating point input parameters
C      xa = 0.000 xb = 1.000 yc = 0.000 yd = 1.000
C      tolerance (error control) = 0.000E+00
C
C      discretization call to cud2cr intl = 0
C      ierror = 0 minimum work space = 64474
C
C      approximation call to cud2cr
C      intl = 1 method = 2 iguess = 0 maxcy = 1
C      ierror = 0
C      maximum error = 0.192E-03
C
C      approximation call to cud2cr
C      intl = 1 method = 2 iguess = 1 maxcy = 2
C      ierror = 0
C      maximum error = 0.194E-03
C
C      cud24cr test ierror = 0
C      maximum error = 0.209E-06
C
C
C ****
C      end of output
C
C ****
C
C      program tcud24cr
```

```

    implicit none
    integer iixp,jjyq,iiex,jjey,nnx,nny,llw
C
C      set grid sizes with parameter statements
C
      parameter (iixp = 3 , jjyq = 2 , iiex =5, jjey =
6)
      parameter (nnx=iixp*2** (iiex-1)+1,
nny=jjyq*2** (jjey-1)+1)
C
C      set minimum required work space (see tcud2cr.f)
C
      parameter (llw = 64474)
C
C      dimension solution,right hand side, and work
arrays
C
      complex p(nnx,nny),r(nnx,nny),w(llw)
C      put integer and floating point parameter names in
contiguous
C      storeage for labelling purposes
C
      integer iprm(16),mgopt(4)
      real fprm(6)
      integer
intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero

common/itcud2cr/intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,
ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero
      real xa,xb,yc,yd,tolmax,relmax
      common/ftcur2/xa,xb,yc,yd,tolmax,relmax
      equivalence(intl,iprm)
      equivalence(xa,fprm)
      integer i,j,ierror
      complex cxx,cxy,cyy,cx,cy,ce,pxx,pxy,pyy,px,py,pe
      real dlx,dly,x,y,errmax
C
C      declare coefficient and boundary condition input
subroutines external
C

```

```
external cofcr,bndcr
c
c
c      set input integer arguments
c
c      intl = 0
c
c      set boundary condition flags
c
c      nxa = 1
c      nxb = 1
c      nyc = 1
c      nyd = 2
c
c      set grid sizes from parameter statements
c
c      ixp = iixp
c      jyq = jjyq
c      iex = iiex
c      jey = jjey
c      nx = nnx
c      ny = nny
c
c      set for one multigrid cycle
c
c      maxcy = 1
c
c      set work space length approximation from parameter
c      statement
c
c      nwork = llw
c
c      set point relaxation
c
c      method = 2
c
c      flag no initial guess (this sets full multigrid
c      cycling)
c
c      iguess = 0
c
c      set end points of solution rectangle in (x,y)
c      space
c
```

```

xa = 0.0
xb = 1.0
yc = 0.0
yd = 1.0
c
c      set mesh increments
c
c      dlx = (xb-xa)/float(nx-1)
c      dly = (yd-yc)/float(ny-1)
c
c      set for no error control
c
tolmax = 0.0
c
c      set right hand side in r
c      initialize p to zero
c
do i=1,nx
x = xa+float(i-1)*dlx
do j=1,ny
y = yc+float(j-1)*dly
call cofcr(x,y,cxx,cxy,cyy,cx,cy,ce)
call exact(x,y,pxx,pxy,pyy,px,py,pe)
r(i,j) = cxx*pxx+cxy*pxy+cyy*pyy+cx*px+cy*py+ce*pe
p(i,j) = (0.0,0.0)
end do
end do
c
c      set specified boundaries in p
c
x = xa
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,pxx,pxy,pyy,px,py,pe)
p(1,j) = pe
end do
x = xb
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,pxx,pxy,pyy,px,py,pe)
p(nx,j) = pe
end do
y = yc
do i=1,nx

```

```

x = xa+float(i-1)*dlx
call exact(x,y,pxx,pxy,pyy,px,py,pe)
p(i,1) = pe
end do
c
c      set default multigrid opitons
c
      mgopt(1) = 2
      mgopt(2) = 2
      mgopt(3) = 1
      mgopt(4) = 3
c
c      print input parameters (except multigrid options
which are default)
c
      write(6,100)
100 format(//' cud2cr test ')
      write (*,101) (iprm(i),i=1,15)
101 format(/' integer input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =
',i2,' nyd = ',i2,
+/' ixp = ',i2,' jyq = ',i2,' iex = ',i2,' jey =
',i2
+/' nx = ',i3,' ny = ',i3,' iguess = ',i2,' maxcy =
'i2,
+/' method = ',i2, ' work space estimate = ',i7)
      write (*,102) (mgopt(i),i=1,4)
102 format(/' multigrid option arguments ',
+/' kcycle = ',i2,
+/' iprer = ',i2,
+/' ipost = ',i2
+/' interpol = ',i2)
      write(*,103) xa,xb,yc,yd,tolmax
103 format(/' floating point input parameters ',
+/' xa = ',f6.3,' xb = ',f6.3,' yc = ',f6.3,' yd =
',f6.3,
+/' tolerance (error control) =    ',e10.3)
c
c      intitialization call
c
      write(*,104) intl
104 format(/' discretization call to cud2cr', ' intl =
', i2)
      call

```

```

cud2cr(iprm,fprm,w,cofc,r,bndcr,r,p,mgopt,ierror)
C
C      print error parameter and minimum work space
requirement
C
      write (*,105) ierror,iprm(16)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)
C
C      attempt solution
C
      intl = 1
      write(*,106) intl,method,iguess,maxcy
106 format(/' approximation call to cud2cr',
+/ ' intl = ',i2, ' method = ',i2,' iguess = ',i2, '
maxcy = ',i2)
      call
cud2cr(iprm,fprm,w,cofc,r,bndcr,r,p,mgopt,ierror)
      write (*,107) ierror
107 format(' ierror = ',i2)
      if (ierror.gt.0) call exit(0)
C
C      compute and print exact maximum error
C
      errmax = 0.0
      do j=1,ny
      y = yc+(j-1)*dly
      do i=1,nx
      x = xa+(i-1)*dlx
      call exact(x,y,pxx,pxy,pyy,px,py,pe)
      errmax = amax1(errmax,cabs((p(i,j)-pe)))
      end do
      end do
      write(*,108) errmax
108 format(' maximum error = ',e10.3)
C
C      recall with iguess=1,maxcy=2
C
      maxcy = 2
      iguess = 1
      write(*,106) intl,method,iguess,maxcy
      call
cud2cr(iprm,fprm,w,cofc,r,bndcr,r,p,mgopt,ierror)

```

```

        write (*,107) ierror
        if (ierror.gt.0) call exit(0)
        errmax = 0.0
        do j=1,ny
        y = yc+(j-1)*dly
        do i=1,nx
        x = xa+(i-1)*dlx
        call exact(x,y,pxx,pxy,pyy,px,py,pe)
        errmax = amax1(errmax,cabs((p(i,j)-pe)))
        end do
        end do
        write(*,108) errmax
C
C      now attempt fourth order estimate
C
        call cud24cr(w,cofcr,bndcr,p,ierror)
        write(*,109) ierror
109 format(/' cud24cr test ', ' ierror = ',i2)
        if (ierror.gt.0.0) call exit(0)
        errmax = 0.0
        do j=1,ny
        y = yc+(j-1)*dly
        do i=1,nx
        x = xa+(i-1)*dlx
        call exact(x,y,pxx,pxy,pyy,px,py,pe)
        errmax = amax1(errmax,cabs((p(i,j)-pe)))
        end do
        end do
        write(*,108) errmax
        end

        subroutine cofcr(x,y,cxx,cxy,cyy,cx,cy,ce)
C
C      input pde coefficients at any point (x,y) in the
solution region
C      (xa.le.x.le.xb,yc.le.y.le.yd) to cud2cr
C
        implicit none
        complex cxx,cxy,cyy,cx,cy,ce
        real x,y,xa,xb,yc,yd,tolmax,relmax
        common/ftcur2/xa,xb,yc,yd,tolmax,relmax
        cxx = cmplx(1.+y**2,1.-y**2)
        cxy = cmplx(x*y,-x*y)
        cyy = cmplx(1.+x**2,1.-x*x)

```

```

cx = cmplx(y,-y)
cy = cmplx(x,-x)
ce = -cmplx( (x+y) ,-x-y)
return
end

subroutine bndcr(kbdy,xory,alfa,beta,gama,gbdy)
c
c      input mixed "oblique" derivative b.c. to cud2cr
c
implicit none
integer kbdy
real xory,x,y,xa,xb,yc,yd,tolmax,relmax
complex alfa,beta,gama,gbdy,pe,px,py,pxx,pxy,pyy
common/ftcur2/xa,xb,yc,yd,tolmax,relmax
if (kbdy.eq.4) then
c
c      y=yd boundary (nyd must equal 2 if this code is to
be executed) .
c      b.c. has the form
alfydyd(x)*px+betydyd(x)*py+gamydyd(x)*pe = gbdydyd(x)
c      where x = yorx.    alfa,beta,gama,gbdy
corresponding to alfydyd(x),
c      betydyd(x),gamydyd(x),gbdydyd(y) must be output.
c
y = yd
x = xory
alfa = -cmplx(x,-x)
beta = cmplx(1.+x,1.-x)
gama = -cmplx(x,-x)
call exact(x,y,pxx,pxy,pyy,px,py,pe)
gbdy = alfa*px + beta*py + gama*pe
return
end if
end

subroutine exact(x,y,pxx,pxy,pyy,px,py,pe)
c
c      this subroutine is used for setting an exact
solution in order
c      to test subroutine cud2cr.
c
implicit none
real x,y,xy,xy2,xy3

```

```
complex pxx,pxy,pyy,px,py,pe
xy = x*y
xy2 = xy*xy
xy3 = xy2*xy
pe = cmplx(xy3,-xy3)
px = 3.*y*cmplx(xy2,-xy2)
py = 3.*x*cmplx(xy2,-xy2)
pxx = 6.*y*y*cmplx(xy,-xy)
pxy = 9.*xy*cmplx(xy,-xy)
pyy = 6.*x*x*cmplx(xy,-xy)
return
end
```

---

## TCUD24SP

```
C
C      file tcud24sp.f
C
C      * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *                               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *                               all rights reserved
*
C      *
*
C      *           MUDPACK   version 5.0.1
*
C      *
*
```



```

C
C      test program for the complex MUDPACK solver
cud24sp
C
C ... required MUDPACK files
C
C      cud24sp.f, cud2sp.f, cudcom.f
C
C
C
C
***** ****
*
C
***** ****
*
C
***** ****
sample program/test driver for cud24sp
C
C
***** ****
**
C
***** ****
**
C
C
C      a sample program/test driver for cud24sp is listed
below. it
C      can be executed as an initial test. the output is
listed for
C      the test case described.
C
C      test the driver below by solving the complex
separable elliptic pde
C
C      cmplx(1.+x*x,1-x*x)*pxx + cmplx(exp(1.-
y),exp(y))*ppy - 
C
C      (cmplx(x,x) + cmplx(y,y))*pe(x,y) = r(x,y)
C
C      on the (x,y) region
C
C      [1/4,3/4] X [1/3,2/3]

```

```

c
c      with specified boundaries at upper x boundry xb =
3/4 and
c      the lower y boundary yc = 1/3 and mixed boundary
conditions
c      at the lower x boundry xa = 1/4
c
c            dp/dx - p = g(y)
c
c      and upper y boundary yd = 2/3
c
c            dp/dy + p = h(x)
c
c      Use point relaxation, the default multigrid
options and
c      the exact solution
c
c            pe(x,y) = cmplx(y**5,x**4+1.0)
c
c      to set the righthand side, boundary conditions,
and compute
c      error. the approximation is generated on a 129 by
97 grid.
c      First cud2sp is called to render a second-order
approximation.
c      then cud24sp is called for a fourth-order estimate
c
c
c
*****
*****  

*****  

c      output (64 bit floating point arithmetic)
c
*****
*****  

*****  

c
c      cud2sp test
c
c      integer input arguments
c      intl = 0 nxa = 2 nxb = 1 nyc = 1 nyd = 2
c      ixp = 2 jyq = 3 iex = 7 jey = 6
c      nx = 129 ny = 97 iguess = 0 maxcy = 3
c      method = 0 work space estimate = 48976
c

```

```

c      multigrid option arguments
c      kcycle = 0
c      iprer = 0
c      ipost = 0
c      interpol = 0
c
c      floating point input parameters
c      xa = 0.000 xb = 1.000 yc = 0.000 yd = 1.000
c      tolerance (error control) = 0.000E+00
c
c      discretization call to cud2sp intl = 0
c      ierror = 0 minimum work space = 48976
c
c      approximation call to cud2sp
c      intl = 1 method = 0 iguess = 0
c      ierror = 0
c      maximum error = 0.317E-03
c
c      cud24sp test ierror = 0
c      maximum error = 0.621E-06
c
c
c ****
c      end of output
c
c ****
c ****
c
c      program tcud24sp
c      implicit none
c      integer iixp,jjyq,iiex,jjey,nnx,nny,llw
c
c      set grid sizes with parameter statements
c
c      parameter (iixp = 2 , jjyq = 3 , iiex =7, jjey =
6)
c              parameter (nnx=iixp*2** (iiex-1)+1,
nny=jjyq*2** (jjey-1)+1)
c
c      set minimum required work space (see tmud2sp.f)
c
c      parameter (llw = 48976)
c

```

```

c      dimension solution, right hand side, and work
arrays
c
c      complex p(nnx,nny),r(nnx,nny),w(llw)
c      put integer and floating point parameter names in
contiguous
c      storeage for labelling purposes
c
c      integer iprm(16),mgopt(4)
c      real fprm(6)
c      integer
intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero

common/itcud2sp/intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,
ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero
      real xa,xb,yc,yd,tolmax,relmax
      common/ftcud2sp/xa,xb,yc,yd,tolmax,relmax
      equivalence(intl,iprm)
      equivalence(xa,fprm)
      integer i,j,ierror
      complex cxx,cyy,cx,cy,ce,pxx,pyy,px,py,pe
      complex pe,px,py,pxx,pyy,cxx,cyy,cx,cy,cex,cey,ce
      real dlx,dly,x,y,errmax

c
c      declare coefficient and boundary condition input
subroutines external
c
c      external cfx,cfy,bndc
c
c
c      set input integer arguments
c
c      intl = 0
c
c      set boundary condition flags
c
nxa = 2
nxb = 1
nyc = 1
nyd = 2

```

```
C
C      set grid sizes from parameter statements
C
C      ixp = iixp
C      jyq = jjyq
C      iex = iiex
C      jey = jjey
C      nx = nnx
C      ny = nny
C
C      set three multigrid cycles
C
C      maxcy = 3
C
C      set work space length approximation from parameter
C      statement
C
C      nwork = llw
C
C      set point relaxation
C
C      method = 0
C
C      flag no initial guess (this sets full multigrid
C      cycling)
C
C      iguess = 0
C
C      set end points of solution rectangle in (x,y)
C      space
C
C      xa = 0.0
C      xb = 1.0
C      yc = 0.0
C      yd = 1.0
C
C      set mesh increments
C
C      dlx = (xb-xa)/float(nx-1)
C      dly = (yd-yc)/float(ny-1)
C
C      set for no error control
C
C      tolmax = 0.0
```

```

c
c      set right hand side in r
c      initialize p to zero
c
do i=1,nx
x = xa+float(i-1)*dlx
call cfx(x,cxx,cx,cex)
do j=1,ny
y = yc+float(j-1)*dly
call cfy(y,cyy,cy,cey)
ce = cex+cey
call exact(x,y,pxx,pyy,px,py,pe)
r(i,j) = cxx*pxx+cyy*pyy+cx*px+cy*py+ce*pe
p(i,j) = (0.0,0.0)
end do
end do
c
c      set specified boundaries in p
c
x = xb
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,pxx,pyy,px,py,pe)
p(nx,j) = pe
end do
y = yc
do i=1,nx
x = xa+float(i-1)*dlx
call exact(x,y,pxx,pyy,px,py,pe)
p(i,1) = pe
end do
c
c      set default multigrid options
c
mgopt(1) = 0
c
c      print input parameters (except multigrid options
which are default)
c
write(6,100)
100 format(//' cud2sp test ')
      write (*,101) (iprm(i),i=1,15)
101 format(/' integer input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =

```

```

',i2,' nyd = ',i2,
      +' ixp = ',i2,' jyq = ',i2,' iex = ',i2,' jey =
',i2
      +' nx = ',i3,' ny = ',i3,' iguess = ',i2,' maxcy =
'i2,
      +' method = ',i2, ' work space estimate = ',i7)
      write (*,102) (mgopt(i),i=1,4)
102 format(/' multigrid option arguments ',
      +' kcycle = ',i2,
      +' iprer = ',i2,
      +' ipost = ',i2
      +' interpol = ',i2)
      write(*,103) xa,xb,yc,yd,tolmax
103 format(/' floating point input parameters ',
      +' xa = ',f6.3,' xb = ',f6.3,' yc = ',f6.3,' yd =
',f6.3,
      +' tolerance (error control) =    ',e10.3)
C
C      intitialization call
C
      write(*,104) intl
104 format(/' discretization call to cud2sp', ' intl =
', i2)
      call
cud2sp(iprm,fprm,w,cfx,cfy,bndc,r,p,mgopt,ierror)
C
C      print error parameter and minimum work space
requirement
C
      write (*,105) ierror,iprm(16)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)
C
C      attempt solution
C
      intl = 1
      write(*,106) intl,method,iguess
106 format(/' approximation call to cud2sp ',
      +' intl = ',i2, ' method = ',i2,' iguess = ',i2)
      call
cud2sp(iprm,fprm,w,cfx,cfy,bndc,r,p,mgopt,ierror)
      write (*,107) ierror
107 format(' ierror = ',i2)

```

```

C
C      compute and print exact maximum error
C
C      errmax = 0.0
      do j=1,ny
      y = yc+(j-1)*dly
      do i=1,nx
      x = xa+(i-1)*dlx
      call exact(x,y,pxx,pyy,px,py,pe)
      errmax = amax1(errmax,cabs((p(i,j)-pe)))
      end do
      end do
      write(*,108) errmax
108 format(' maximum error = ',e10.3)
C
C      fourth-order estimate
C
      call cud24sp(w,p,ierror)
      write(*,109) ierror
109 format(/' cud24sp test ', ' ierror = ',i2)
      errmax = 0.0
      do j=1,ny
      y = yc+(j-1)*dly
      do i=1,nx
      x = xa+(i-1)*dlx
      call exact(x,y,pxx,pyy,px,py,pe)
      errmax = amax1(errmax,cabs((p(i,j)-pe)))
      end do
      end do
      write(*,108) errmax
      end

      subroutine cfx(x,cxx,cx,cex)
C
C      input x dependent complex coefficients
C
      implicit none
      real x
      complex cxx,cx,cex
      cxx = cmplx(1.+x*x,1.-x*x)
      cx = (0.,0.)
      cex = -cmplx(x,x)
      return
      end

```

```

subroutine cfy(y,cyy,cy,cey)
C
C      input y dependent complex coefficients
C
      implicit none
      real y
      complex cyy, cy, cey
      cyy = cmplx(exp(1.0-y),exp(y) )
      cy = -cyy
      cey = -cmplx(y,y)
      return
      end

subroutine bndc(kbdy,xory,cons,gbdy)
C
C      input mixed complex derivative b.c. to cud2sp
C
      implicit none
      integer kbdy
      real xory,x,y
      complex cons,gbdy,pe,px,py,pxx,pyy
      real xa,xb,yc,yd,tolmax,relmax
      common/ftcud2sp/xa,xb,yc,yd,tolmax,relmax
      if (kbdy.eq.1) then
C
C      x=xa boundary
C
      y = xory
      x = xa
      cons =(-1.0,0.0)
      call exact(x,y,pxx,pyy,px,py,pe)
      gbdy = px + cons*pe
      return
      end if
      if (kbdy.eq.4) then
C
C      y=yd boundary
C
      y = yd
      x = xory
      cons = (1.0,0.0)
      call exact(x,y,pxx,pyy,px,py,pe)
      gbdy = py + cons*pe

```

```

        return
    end if
end

subroutine exact(x,y,pxx,pyy,px,py,pe)
C
C      set exact solution used for testing cud2sp
C
      implicit none
      real x,y
      complex pe,px,py,pxx,pyy
      pe = cmplx(y**5.,x**4+1.)
      px = cmplx(0.,4.*x**3)
      pxx = cmplx(0.,12.*x**2)
      py = cmplx(5.*y**4,0.)
      pyy = cmplx(20.*y**3,0.)
      return
end

```

## TCUD2CR

```

C
C      file tcud2cr.f
C
C      * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *                               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *                               all rights reserved
*

```

C \*  
\*  
C \* MUDPACK version 5.0.1  
\*  
C \*  
\*  
C \* A Fortran Package of Multigrid  
\*  
C \*  
\*  
C \* Subroutines and Example Programs  
\*  
C \*  
\*  
C \* for Solving Elliptic Partial Differential  
Equations \*  
C \*  
\*  
C \* by  
\*  
C \*  
\*  
C \* John Adams  
\*  
C \*  
\*  
C \* of  
\*  
C \*  
\*  
C \* the National Center for Atmospheric  
Research \*  
C \*  
\*  
C \* Boulder, Colorado (80307)  
U.S.A. \*  
C \*  
\*  
C \* which is sponsored by  
\*  
C \*  
\*  
C \* the National Science Foundation  
\*

```

C      *
*
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C
C ... purpose
C
C      test program for the complex MUDPACK solver cud2cr
C
C ... required MUDPACK files
C
C      cud2cr.f, cudcom.f
C
C
*****
* *
C
*****
* *
C
C      sample program/test driver for cud2cr
C
C
*****
** *
C
*****
** *
C
C      a sample program/test driver for cud2cr is below.
it can
c      be executed as an initial test.  the output is
listed for
c      the test case described.
C
C      test the driver below by solving the elliptic pde
C
C      cmplx(1.+y*y,1.-y*y)*d(dp/dx)/dx + cmplx(x*y,-
x*y)*d(dp/dx)/dy
C
C      cmplx(1.+x*x,1.-x*x)*d(dp/dy)/dy + cmplx(y,-
y)*dp/dx +
C
C      cmplx(x,-x)*dp/dy + cmplx(x+y,-x-y)*p(x,y) =

```

```

r(x,y)
c
c      on the unit square with specified boundary
conditions at
c      xa=0.0, xb=1.0, yc=0.0 and mixed boundary
conditions at yd=1.0
c      of the form:
c
c      cmplx(-x,x)*dp/dx+cmplx(1.+x,1.-x)*dp/dy+cmplx(-
x,x)*p(x,yd)=gbdy(x)
c
c      choose a 49 x 65 grid and use line relaxation in
the y direction.
c      forr testing purposes use the exact solution
c
c      p(x,y) = cmplx((x*y)**3,-(x*y)**3) + (1.,1.)
c
c ****
c      output (32 bit floating point arithmetic)
c
*****
c
c      cud2cr test
c
c      integer input arguments
c      intl = 0 nxa = 1 nxb = 1 nyc = 1 nyd = 2
c      ixp = 3 jyq = 2 iex = 5 jey = 6
c      nx = 49 ny = 65 iguess = 0 maxcy = 1
c      method = 2 work space estimate = 67426
c
c      multigrid option arguments
c      kcycle = 2
c      iprer = 2
c      ipost = 1
c      interpol = 3
c
c      floating point input parameters
c      xa = 0.000 xb = 1.000 yc = 0.000 yd = 1.000
c      tolerance (error control) = 0.000E+00
c
c      discretization call to cud2cr intl = 0
c      ierror = 0 minimum work space = 64474
c
c      approximation call to cud2cr

```

```

c      intl = 1 method = 2 iguess = 0
c      ierror = 0
c      maximum error = 0.188E-03
c
c
***** *****
**
c      end of output
c
***** *****
**
c
program tcud2cr
implicit none
integer iixp,jjyq,iiex,jjey,nnx,nny,llw
c
c      set grid sizes with parameter statements
c
parameter (iixp = 3 , jjyq = 2 , iiex =5, jjey =
6)
parameter (nnx=iixp*2** (iiex-1)+1,
nny=jjyq*2** (jjey-1)+1)
c
c      estimate work length approximation for method=0
(see cud2cr.d)
c
parameter (llw=(7* (nnx+2) * (nny+2)+56*nnx*nny) /3)
c
c      dimension solution,right hand side, and work
arrays
c
complex p(nnx,nny),r(nnx,nny),w(llw)
c      put integer and floating point parameter names in
contiguous
c      storeage for labelling purposes
c
integer iprm(16),mgopt(4)
real fprm(6)
integer
intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero

common/itcud2cr/intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,

```

```
ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero
    real xa,xb,yc,yd,tolmax,relmax
    common/ftcur2/xa,xb,yc,yd,tolmax,relmax
    equivalence(intl,iprm)
    equivalence(xa,fprm)
    integer i,j,ierror
    complex cxx,cxy,cyy,cx,cy,ce,pxx,pxy,pyy,px,py,pe
    real dlx,dly,x,y,errmax
C
C      declare coefficient and boundary condition input
subroutines external
C
        external cofcr,bndcr
C
C      set input integer arguments
C
        intl = 0
C
C      set boundary condition flags
C
        nxa = 1
        nxb = 1
        nyc = 1
        nyd = 2
C
C      set grid sizes from parameter statements
C
        ixp = iixp
        jyq = jjyq
        iex = iiex
        jey = jjey
        nx = nnx
        ny = nny
C
C      set for one multigrid cycle
C
        maxcy = 1
C
C      set work space length approximation from parameter
statement
C
```

```

nwork = llw
c
c      set point relaxation
c
method = 2
c
c      flag no initial guess (this sets full multigrid
cycling)
c
iguess = 0
c
c      set end points of solution rectangle in (x,y)
space
c
xa = 0.0
xb = 1.0
yc = 0.0
yd = 1.0
c
c      set mesh increments
c
dlx = (xb-xa)/float(nx-1)
dly = (yd-yc)/float(ny-1)
c
c      set for no error control
c
tolmax = 0.0
c
c      set right hand side in r
c      initialize p to zero
c
do i=1,nx
x = xa+float(i-1)*dlx
do j=1,ny
y = yc+float(j-1)*dly
call cofcr(x,y,cxx,cxy,cyy,cx,cy,ce)
call exact(x,y,pxx,pxy,pyy,px,py,pe)
r(i,j) = cxx*pxx+cxy*pxy+cyy*pyy+cx*px+cy*py+ce*pe
p(i,j) = (0.0,0.0)
end do
end do
c
c      set specified boundaries in p
c

```

```

x = xa
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,pxx,pxy,pyy,px,py,pe)
p(1,j) = pe
end do
x = xb
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,pxx,pxy,pyy,px,py,pe)
p(nx,j) = pe
end do
y = yc
do i=1,nx
x = xa+float(i-1)*dlx
call exact(x,y,pxx,pxy,pyy,px,py,pe)
p(i,1) = pe
end do
c
c      set default multigrid opitons
c
mgopt(1) = 2
mgopt(2) = 2
mgopt(3) = 1
mgopt(4) = 3
c
c      print input parameters (except multigrid options
which are default)
c
      write(6,100)
100 format(//' cud2cr test ')
      write (*,101) (iprm(i),i=1,15)
101 format(/' integer input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =
',i2,' nyd = ',i2,
+/' ixp = ',i2,' jyq = ',i2,' iex = ',i2,' jey =
',i2
+/' nx = ',i3,' ny = ',i3,' iguess = ',i2,' maxcy =
',i2,
+/' method = ',i2, ' work space estimate = ',i7)
      write (*,102) (mgopt(i),i=1,4)
102 format(/' multigrid option arguments ',
+/' kcycle = ',i2,
+/' iprer = ',i2,

```

```

+/' ipost = ',i2
+/' interpol = ',i2)
    write(*,103) xa,xb,yc,yd,tolmax
103 format(/' floating point input parameters ',
+/' xa = ',f6.3,' xb = ',f6.3,' yc = ',f6.3,' yd =
',f6.3,
+/' tolerance (error control) =     ',e10.3)
C
C      intialization call
C
        write(*,104) intl
104 format(/' discretization call to cud2cr', ' intl =
', i2)
        call
cud2cr(iprm,fprm,w,cofcr,bndcr,r,p,mgopt,ierror)
C
C      print error parameter and minimum work space
requirement
C
        write (*,105) ierror,iprm(16)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
        if (ierror.gt.0) call exit(0)
C
C      attempt solution
C
        intl = 1
        write(*,106) intl,method,iguess
106 format(/' approximation call to cud2cr',
+/' intl = ',i2, ' method = ',i2,' iguess = ',i2)
        call
cud2cr(iprm,fprm,w,cofcr,bndcr,r,p,mgopt,ierror)
        write (*,107) ierror
107 format(' ierror = ',i2)
C
C      compute and print exact maximum error
C
        errmax = 0.0
        do j=1,ny
y = yc+(j-1)*dly
        do i=1,nx
            x = xa+(i-1)*dlx
            call exact(x,y,pxx,pxy,pyy,px,py,pe)
            errmax = amax1(errmax,cabs((p(i,j)-pe)))

```

```

    end do
    end do
    write(*,108) errmax
108 format(' maximum error = ',e10.3)
    end

    subroutine cofcr(x,y,cxx,cxy,cyy,cx,cy,ce)
C
C      input pde coefficients at any point (x,y) in the
solution region
C      (xa.le.x.le.xb,yc.le.y.le.yd) to cud2cr
C
        implicit none
        complex cxx,cxy,cyy,cx,cy,ce
        real x,y,xa,xb,yc,yd,tolmax,relmax
        common/ftcur2/xa,xb,yc,yd,tolmax,relmax
        cxx = cmplx(1.+y**2,1.-y**2)
        cxy = cmplx(x*y,-x*y)
        cyy = cmplx(1.+x**2,1.-x*x)
        cx = cmplx(y,-y)
        cy = cmplx(x,-x)
        ce = -cmplx((x+y),-x-y)
        return
        end

    subroutine bndcr(kbdy,xory,alfa,beta,gama,gbdy)
C
C      input mixed "oblique" derivative b.c. to cud2cr
C
        implicit none
        integer kbdy
        real xory,x,y,xa,xb,yc,yd,tolmax,relmax
        complex alfa,beta,gama,gbdy,pe,px,py,pzx,pzy,pyy
        common/ftcur2/xa,xb,yc,yd,tolmax,relmax
        if (kbdy.eq.4) then
C
C      y=yd boundary (nyd must equal 2 if this code is to
be executed) .
C      b.c. has the form
        alfyd(x)*px+betyd(x)*py+gamyd(x)*pe = gbdyd(x)
C      where x = yorx.  alfa,beta,gama,gbdy
corresponding to alfyd(x),
C      betyd(x),gamyd(x),gbdyd(y) must be output.
C

```

```

y = yd
x = xory
alfa = -cmplx(x,-x)
beta = cmplx(1.+x,1.-x)
gama = -cmplx(x,-x)
call exact(x,y,pxx,pxy,pyy,px,py,pe)
gbdy = alfa*px + beta*py + gama*pe
return
end if
end

subroutine exact(x,y,pxx,pxy,pyy,px,py,pe)
c
c      this subroutine is used for setting an exact
solution in order
c      to test subroutine cud2cr.
c
implicit none
real x,y,xy,xy2,xy3
complex pxx,pxy,pyy,px,py,pe
xy = x*y
xy2 = xy*xy
xy3 = xy2*xy
pe = cmplx(xy3,-xy3)
px = 3.*y*cmplx(xy2,-xy2)
py = 3.*x*cmplx(xy2,-xy2)
pxx = 6.*y*y*cmplx(xy,-xy)
pxy = 9.*xy*cmplx(xy,-xy)
pyy = 6.*x*x*cmplx(xy,-xy)
return
end

```

TCUD2SP

```
* * * * * *
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK version 5.0.1
*
C      *
*
C      *               A Fortran Package of Multigrid
*
C      *
*
C      *               Subroutines and Example Programs
*
C      *
*
C      *               for Solving Elliptic Partial Differential
Equations        *
C      *
*
C      *               by
*
C      *
*
C      *               John Adams
*
C      *
*
C      *               of
*
C      *
*
C      *               the National Center for Atmospheric
```

```
Research          *
C      *
*
C      *                      Boulder, Colorado (80307)
U.S.A.          *
C      *
*
C      *                      which is sponsored by
*
C      *
*
C      *                      the National Science Foundation
*
C      *
*
C      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *
*      *      *      *      *      *
C
C ... purpose
C
C      test program for the complex MUDPACK solver cud2sp
C
C ... required MUDPACK files
C
C      cud2sp.f, cudcom.f
C
C
C
C
*****
*
C
*****
*
C
C      sample program/test driver for cud2sp
C
C
*****
**
C
*****
**
C
```

```

c
c      a sample program/test driver for cud2sp is listed
below. it
c      can be executed as an initial test. the output is
listed for
c      the test case described.
c
c      test the driver below by solving the complex
separable elliptic pde
c
c            cmplx(1.+x*x,1-x*x)*pxx + cmplx(exp(1.-
y),exp(y))* (pyy-py) -
c
c            (cmplx(x,x) + cmplx(y,y))*pe(x,y) = r(x,y)
c
c      on the (x,y) region
c
c      [1/4,3/4] X [1/3,2/3]
c
c      with specified boundaries at upper x boundry xb =
3/4 and
c      the lower y boundary yc = 1/3 and mixed boundary
conditions
c      at the lower x boundry xa = 1/4
c
c            dp/dx - p = g(y)
c
c      and upper y boundary yd = 2/3
c
c            dp/dy + p = h(x)
c
c      Use point relaxation, the default multigrid
options and
c      the exact solution
c
c            pe(x,y) = cmplx(y**5,x**4+1.0)
c
c      to set the righthand side, boundary conditions,
and compute
c      error. the approximation is generated on a 129 by
97 grid.
c
c
*****
```

```

*****
C      output (32 bit floating point arithmetic)
C
***** *****
C
C      cud2sp test
C
C      integer input arguments
C      intl = 0 nxa = 2 nxb = 1 nyc = 1 nyd = 2
C      ixp = 2 jyq = 3 iex = 7 jey = 6
C      nx = 129 ny = 97 iguess = 0 maxcy = 3
C      method = 0 work space estimate = 64825
C
C      multigrid option arguments
C      kcycle = 0
C      iprer = 0
C      ipost = 0
C      interpol = 0
C
C      floating point input parameters
C      xa = 0.000 xb = 1.000 yc = 0.000 yd = 1.000
C      tolerance (error control) = 0.000E+00
C
C      discretization call to cud2sp intl = 0
C      ierror = 0 minimum work space = 48976
C
C      approximation call to cud2sp
C      intl = 1 method = 0 iguess = 0
C      ierror = 0
C      maximum error = 0.328E-03
C
C
C
***** *****
C      end of output
C
***** *****
C
C      program tcud2sp
C      implicit none
C      integer iixp,jjyq,iiex,jjey,nnx,nny,llw
C
```

```

c      set grid sizes with parameter statements
c
c      parameter (iixp = 2 , jjyq = 3 , iiex =7, jjey =
6)
c          parameter (nnx=iixp*2** (iiex-1)+1,
nny=jjyq*2** (jjey-1)+1)
c
c      estimate work length approximation for method=0
(see cud2sp.d)
c
c      parameter (llw = 5* (nnx*nny+2* (nnx+nny)) )
c
c      dimension solution,right hand side, and work
arrays
c
c      complex p(nnx,nny),r(nnx,nny),w(llw)
c      put integer and floating point parameter names in
contiguous
c      storeage for labelling purposes
c
c      integer iprm(16),mgopt(4)
c      real fprm(6)
c      integer
intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero

common/itcud2sp/intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,
ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero
      real xa,xb,yc,yd,tolmax,relmax
      common/ftcud2sp/xa,xb,yc,yd,tolmax,relmax
      equivalence(intl,iprm)
      equivalence(xa,fprm)
      integer i,j,ierror
      complex cxx,cyy,cx,cy,ce,pxx,pyy,px,py,pe
      complex pe,px,py,pxx,pyy,cxx,cyy,cx,cy,cex,cey,ce
      real dlx,dly,x,y,errmax
c
c      declare coefficient and boundary condition input
subroutines external
c
      external cfx,cfy,bndc

```

```
C
C
C      set input integer arguments
C
C      intl = 0
C
C      set boundary condition flags
C
C      nxa = 2
C      nxb = 1
C      nyc = 1
C      nyd = 2
C
C      set grid sizes from parameter statements
C
C      ixp = iixp
C      jyq = jjyq
C      iex = iiex
C      jey = jjey
C      nx = nnx
C      ny = nny
C
C      set three multigrid cycles
C
C      maxcy = 3
C
C      set work space length approximation from parameter
C      statement
C
C      nwork = llw
C
C      set point relaxation
C
C      method = 0
C
C      flag no initial guess (this sets full multigrid
C      cycling)
C
C      iguess = 0
C
C      set end points of solution rectangle in (x,y)
C      space
C
C      xa = 0.0
```

```

    xb = 1.0
    yc = 0.0
    yd = 1.0
C
C      set mesh increments
C
        dlx = (xb-xa)/float(nx-1)
        dly = (yd-yc)/float(ny-1)
C
C      set for no error control
C
        tolmax = 0.0
C
C      set right hand side in r
C      initialize p to zero
C
        do i=1,nx
          x = xa+float(i-1)*dlx
          call cfx(x,cxx,cx,cex)
          do j=1,ny
            y = yc+float(j-1)*dly
            call cfy(y,cyy,cy,cey)
            ce = cex+cey
            call exact(x,y,pxx,pyy,px,py,pe)
            r(i,j) = cxx*pxx+cyy*pyy+cx*px+cy*py+ce*pe
            p(i,j) = (0.0,0.0)
          end do
        end do
C
C      set specified boundaries in p
C
        x = xb
        do j=1,ny
          y = yc+float(j-1)*dly
          call exact(x,y,pxx,pyy,px,py,pe)
          p(nx,j) = pe
        end do
        y = yc
        do i=1,nx
          x = xa+float(i-1)*dlx
          call exact(x,y,pxx,pyy,px,py,pe)
          p(i,1) = pe
        end do
C

```

```

c      set default multigrid opitons
c
c      mgopt(1) = 0
c
c      print input parameters (except multigrid options
c which are default)
c
c      write(6,100)
100 format(//' cud2sp test ')
      write (*,101) (iprm(i),i=1,15)
101 format(/' integer input arguments ',
      +' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =
',i2,' nyd = ',i2,
      +' ixp = ',i2,' jyq = ',i2,' iex = ',i2,' jey =
',i2
      +' nx = ',i3,' ny = ',i3,' iguess = ',i2,' maxcy =
',i2,
      +' method = ',i2, ' work space estimate = ',i7)
      write (*,102) (mgopt(i),i=1,4)
102 format(/' multigrid option arguments ',
      +' kcycle = ',i2,
      +' iprer = ',i2,
      +' ipost = ',i2
      +' interpol = ',i2)
      write(*,103) xa,xb,yc,yd,tolmax
103 format(/' floating point input parameters ',
      +' xa = ',f6.3,' xb = ',f6.3,' yc = ',f6.3,' yd =
',f6.3,
      +' tolerance (error control) =     ',e10.3)
c
c      initialization call
c
c      write(*,104) intl
104 format(/' discretization call to cud2sp', ' intl =
', i2)
      call
cud2sp(iprm,fprm,w,cfx,cfy,bndc,r,p,mgopt,ierror)
c
c      print error parameter and minimum work space
c requirement
c
c      write (*,105) ierror,iprm(16)
105 format(' ierror = ',i2, ' minimum work space =
',i7)

```

```

    if (ierror.gt.0) call exit(0)
C
C      attempt solution
C
    intl = 1
    write(*,106) intl,method,iguess
106 format(' approximation call to cud2sp ',
     +' intl = ',i2, ' method = ',i2,' iguess = ',i2)
    call
cud2sp(iprm,fprm,w,cfx,cfy,bndc,r,p,mgopt,ierror)
    write (*,107) ierror
107 format(' ierror = ',i2)
C
C      compute and print exact maximum error
C
    errmax = 0.0
    do j=1,ny
        y = yc+(j-1)*dly
    do i=1,nx
        x = xa+(i-1)*dlx
        call exact(x,y,pxx,pyy,px,py,pe)
        errmax = amax1(errmax,cabs((p(i,j)-pe)))
    end do
    end do
    write(*,108) errmax
108 format(' maximum error = ',e10.3)
end

subroutine cfx(x,cxx,cx,cex)
C
C      input x dependent complex coefficients
C
    implicit none
    real x
    complex cxx,cx,cex
    cxx = cmplx(1.+x*x,1.-x*x)
    cx = (0.,0.)
    cex = -cmplx(x,x)
    return
end

subroutine cfy(y,cyy,cy,cey)
C
C      input y dependent complex coefficients

```

```

c
    implicit none
    real y
    complex cyy, cy, cey
    cyy = cmplx(exp(1.0-y),exp(y) )
    cy = -cyy
    cey = -cmplx(y,y)
    return
    end

    subroutine bndc(kbdy,xory,cons,gbdy)
c
c      input mixed complex derivative b.c. to cud2sp
c
        implicit none
        integer kbdy
        real xory, x, y
        complex cons, gbdy, pe, px, py, pxx, pyy
        real xa, xb, yc, yd, tolmax, relmax
        common/ftcud2sp/xa, xb, yc, yd, tolmax, relmax
        if (kbdy.eq.1) then
c
c      x=xa boundary
c
        y = xory
        x = xa
        cons = (-1.0,0.0)
        call exact(x,y,pxx,pyy,px,py,pe)
        gbdy = px + cons*pe
        return
        end if
        if (kbdy.eq.4) then
c
c      y=yd boundary
c
        y = yd
        x = xory
        cons = (1.0,0.0)
        call exact(x,y,pxx,pyy,px,py,pe)
        gbdy = py + cons*pe
        return
        end if
        end

```

```
C subroutine exact(x,y,pxx,pyy,px,py,pe)
C
C set exact solution used for testing cud2sp
C
implicit none
real x,y
complex pe,px,py,pxx,pyy
pe = cmplx(y**5.,x**4+1.)
px = cmplx(0.,4.*x**3)
pxx = cmplx(0.,12.*x**2)
py = cmplx(5.*y**4,0.)
pyy = cmplx(20.*y**3,0.)
return
end
```

TCUD3

```
C      file tcud3.f
C
C      * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
C      *
C      *               copyright (c) 2008 by UCAR
*
C      *
C      *
C      *       University Corporation for Atmospheric
Research           *
C      *
C      *
C      *               all rights reserved
*
C      *
C      *
C      *               MUDPACK version 5.0.1
*
```



```

C
C ... purpose
C
C      test program for the MUDPACK solver cud3
C
C ... required MUDPACK files
C
C      cud3.f, cudcom.f, cud3ln.f, cud3pn.f
C
C
C
C ****
*
C
C ****
*
C
C      sample program/test driver for cud3
C
C
C ****
**
C
C ****
**
C
C
C      a sample program/test driver for cud3 is below. it
can be
C      executed as an initial test. the output is listed
for the
C      test case described.
C
C      test the driver below by solving the complex
nonseparable
C      linear elliptic pde
C
C      cmplx(y,z)*d(dp/dx)/dx +
cmplx(x,z)*d(dp/dy)/dy +
C
C      cmplx(x,y)*d(dp/dz)/dz -
cmplx(y+z,x+y)*pe(x,y,z)
C
C      = r(x,y,z)

```

```

c
c      on the box [0.5,1.0] x [0.5,1.0] x [0.0,1.0] with
dirchlet
c      boundary conditions at the x and y boundaries and
periodic
c      boundary conditions in the z direction.  use point
relaxation
c      and choose a grid 33 by 33 by 97 grid.  use the
exact solution
c
c          pe(x,y,z) =
exp(x*y)*cmplx(cos(tpi*z),sin(tpi*z))
c
c      for testing (tpi = 8.0*atan(1.0)).
c
c
c ****
c      output (32 bit floating point arithmetic)
c
*****
c
c      cud3 test
c
c      input arguments
c      intl = 0 nxa = 1 nxb = 1 nyc = 1 nyd = 1
c      nze = 0 nzf = 0
c      ixp = 2 jyq = 2 kxr = 3
c      iex = 5 jey = 5 kez = 6
c      nx = 33 ny = 33 nz = 97 iguess = 0 maxcy = 1
c      method = 3 work space length input = 1940400
c      xa = 0.50 xb = 1.00
c      yc = 0.50 yd = 1.00
c      ze = 0.00 zf = 1.00
c      tolmax = 0.000e+00
c
c      multigrid options
c      kcycle = 2
c      iprer = 2
c      ipost = 1
c      interpol = 3
c
c      discretization call to cud3 intl = 0
c      ierror = 0 minimum work space = 1854498
c

```

```

c      approximation call to cud3
c      intl = 1 method = 3 iguess = 0 maxcy = 1
c      ierror = 0
c      maximum error = 0.361e-03
c
c
c ****
c ****
c      end of output
c
c ****
c ****
c
c      program tcud3
c      implicit none
c
c      set grid sizes with parameter statements
c
c      integer
iixp,jjyq,kkzr,iiex,jjey,kkez,llwork,nnx,nny,nnz
      parameter(iixp=2,jjyq=2,kkzr=3)
      parameter(iiex=5,jjey=5,kkez=6)
      parameter (nnx = iixp*2** (iiex-1)+1)
      parameter (nny = jjyq*2** (jjey-1)+1)
      parameter (nnz = kkzr*2** (kkez-1)+1)
c
c      set work space length estimate for method=3 (see
c cud3.d) .
c      this will probably overestimate required space
c
c      parameter (llwork = 16* (nnx+2) * (nny+2) * (nnz+2) )
c
c
c      dimension solution,right hand side, and work
arrays
c
c      complex
phi(nnx,nny,nnz),rhs(nnx,nny,nnz),work(llwork)
      integer iprm(23),mgopt(4)
      real fprm(8)
c
c      put integer and floating point arguments names in
contiguous
c      storeage labelling

```

```

c
      integer
intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex,jey,
+
kez,nx,ny,nz,iguess,maxcy,method,meth2,nwork,
+
           lwrkqd,itero

common/itcud3/intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,i
ex,jey,
+
kez,nx,ny,nz,iguess,maxcy,method,meth2,nwork,
+
           lwrkqd,itero

      real xa,xb,yc,yd,ze,zf,tolmax,relmax,tpi
      common/ftcud3/xa,xb,yc,yd,ze,zf,tolmax,relmax,tpi
      real dlx,dly,dlz,x,y,z,errm
      complex cxx,cyy,czz,cx,cy,cz,ce,
pxx,pyy,pzz,px,py,pz,pe
      integer i,j,k,ierror
      equivalence(intl,iprm)
      equivalence(xa,fprm)

c
c      declare coefficient and boundary condition input
subroutines external
c
      external cof,bndc
      tpi = 8.0*atan(1.0)
c
c      set for initial call
c
      intl = 0
c
c      set boundary condition flags
c
      nxa = 1
      nxb = 1
      nyc = 1
      nyd = 1
      nze = 0
      nzf = 0
c
c      set grid sizes from parameter statements
c
      ixp = iixp

```

```
jyq = jjyq
kzr = kkzr
iex = iiex
jey = jjey
kez = kkez
nx = nnx
ny = nny
nz = nnz

c
c      set for one multigrid cycles
c
c      maxcy = 1
c
c      set work space length approximation from parameter
statement
c
nwork = llwork
c
c      set method of relaxation--line in the z direction
c
method = 3
c
c      meth2 only used in planar relaxation--but set
c
meth2 = 0
c
c      set full multigrid cycling by flagging no initial
guess at the finest
c      grid level
c
iguess = 0
c
c      set end points of solution region in (x,y,z) space
c
xa = 0.5
xb = 1.0
yc = 0.5
yd = 1.0
ze = 0.0
zf = 1.0
c
c      set default multigrid options
c
mgopt(1) = 2
```

```

mgopt(2) = 2
mgopt(3) = 1
mgopt(4) = 3
C
C      set mesh increments
C
dlx = (xb-xa)/float(nx-1)
dly = (yd-yc)/float(ny-1)
dlz = (zf-ze)/float(nz-1)
C
C      set for no error control
C
tolmax = 0.0
C
C      set right hand side in rhs and phi to zero
C
do k=1,nz
z = ze+(k-1)*dlz
do j=1,ny
y = yc+float(j-1)*dly
do i=1,nx
x = xa+float(i-1)*dlx
call cof(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
rhs(i,j,k) =
cxx*pxx+cyy*pyy+czz*pzz+cx*px+cy*py+cz*pz+ce*pe
phi(i,j,k) = (0.,0.)
end do
end do
end do
C
C      set specified values at x and y boundaries in phi
C
x = xa
do k=1,nz
z = ze+(k-1)*dlz
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(1,j,k) = pe
end do
end do
x = xb
do k=1,nz

```

```

z = ze+(k-1)*dlz
do j=1,ny
    y = yc+float(j-1)*dly
    call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
    phi(nx,j,k) = pe
end do
    end do
    y = yc
    do k=1,nz
z = ze+(k-1)*dlz
do i=1,nx
    x = xa+float(i-1)*dlx
    call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
    phi(i,1,k) = pe
end do
    end do
    y = yd
    do k=1,nz
z = ze+(k-1)*dlz
do i=1,nx
    x = xa+float(i-1)*dlx
    call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
    phi(i,ny,k) = pe
end do
    end do
    write(6,50)
50 format(//' cud3 test ')
c
c      print input arguments
c

write(6,100)intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex
,jey,kez,
+
nx,ny,nz,iguess,maxcy,method,nwork,xa,xb,yc,yd,ze,zf,
+
tolmax,mgopt(1),mgopt(2),mgopt(3),mgopt(4)
100 format(/' input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =
',i2,' nyd = 'i2,
+/' nze = ',i2,' nzf = ',i2,
+/' ixp = ',i2,' jyq = 'i2,' kzr = 'i2,
+/' iex = ',i2,' jey = 'i2,' kez = 'i2,
+/' nx = 'i3,' ny = 'i3,' nz = 'i3,' iguess =

```

```

'i2,' maxcy = 'i2,
+/' method = 'i2, ' work space length input = ',i7,
+/' xa = 'f5.2,' xb = 'f5.2,
+/' yc = 'f5.2,' yd = 'f5.2,
+/' ze = 'f5.2,' zf = 'f5.2,
+/' tolmax = ' ,e10.3
+/' multigrid options '
+/' kcycle = ',i2
+/' iprер = ',i2
+/' ipost = ',i2
+/' interpol = ',i2 )

C
C      discretize pde
C
      write(*,104) intl
104 format(/' discretization call to cud3', ' intl =
', i2)
      call
cud3(iprm,fprm,work,cof,bndc,rhs,phi,mgopt,ierror)
      write (*,105) ierror,iprm(22)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)
C
C      approximate pde
C
      intl = 1
      write(*,106) intl,method,iguess,maxcy
106 format(/' approximation call to cud3 ',
+/' intl = ',i2, ' method = ',i2,' iguess = ',i2, '
maxcy = ',i2)
      call
cud3(iprm,fprm,work,cof,bndc,rhs,phi,mgopt,ierror)
      write (*,107) ierror
107 format(' ierror = ',i2)
      if (ierror.gt.0) call exit(0)
C
C      compute and print maximum error
C
      call error(nx,ny,nz,phi,errm)
      write(*,108) errm
108 format(' maximum error  =  ',e10.3)
      end

```

```

subroutine error(nx,ny,nz,phi,errm)
C
C      compute the error in the estimate in phi
C
      implicit none
      integer nx,ny,nz
      complex phi(nx,ny,nz)
      real xa,xb,yc,yd,ze,zf,tolmax,relmax,errm,tpi
      common/ftcud3/xa,xb,yc,yd,ze,zf,tolmax,relmax,tpi
      real dlx,dly,dlz,x,y,z
      complex pxx,pyy,pzz,px,py,pz,pe
      integer i,j,k
      dlx = (xb-xa) / (nx-1)
      dly = (yd-yc) / (ny-1)
      dlz = (zf-ze) / (nz-1)
      errm = 0.0
      do k=1,nz
      z = ze+(k-1)*dlz
      do j=1,ny
      y = yc+float(j-1)*dly
      do i=1,nx
      x = xa+float(i-1)*dlx
      call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
      errm = amax1(errm,abs(phi(i,j,k)-pe))
      end do
      end do
      end do
      return
      end

subroutine cof(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
C
C      input pde coefficients at grid point (x,y,z) in
the solution region
C      to subroutine cud3
C
      implicit none
      complex cxx,cyy,czz,cx,cy,cz,ce
      real x,y,z
      cxx = cmplx(y,z)
      cyy = cmplx(x,z)
      czz = cmplx(x,y)
      cx = (0.,0.)
      cy = cx

```

```

cz = cz
ce = -cmplx(y+z,x+z)
return
end

subroutine bndc(kbdy,xory,yorz,alfa,gbdy)
C
C      dummy subroutine since no mixed derivative b.c.
C
implicit none
integer kbdy
real xory,yorz
complex alfa,gbdy
return
end

subroutine exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
C
C      this subroutine is used to set an exact solution
for testing cud3
C
implicit none
real x,y,z
complex pxx,pyy,pzz,px,py,pz,pe
real xa,xb,yc,yd,ze,zf,tolmax,relmax,tpi
common/ftcud3/xa,xb,yc,yd,ze,zf,tolmax,relmax,tpi
pe = exp(x*y)*cmplx(cos(tpi*z),sin(tpi*z))
px = y*pe
py = x*pe
pz = tpi*exp(x*y)*cmplx(-sin(tpi*z),cos(tpi*z))
pxx = y*y*pe
pyy = x*x*pe
pzz = -tpi*tpi*pe
return
end

```

```
C
C      file tcud34.f
C
C      * * * * *
* * * * *
C      *
*
C      *           copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *           all rights reserved
*
C      *
*
C      *           MUDPACK version 5.0.1
*
C      *
*
C      *           A Fortran Package of Multigrid
*
C      *
*
C      *           Subroutines and Example Programs
*
C      *
*
C      *           for Solving Elliptic Partial Differential
Equations          *
C      *
*
C      *           by
*
C      *
*
C      *           John Adams
*
C      *
```

C \* of  
\*  
C \*  
\*  
C \* the National Center for Atmospheric  
Research \*  
C \*  
\*  
C \* Boulder, Colorado (80307)  
U.S.A. \*  
C \*  
\*  
C \* which is sponsored by  
\*  
C \*  
\*  
C \* the National Science Foundation  
\*  
C \*  
\*  
C \*  
\* \* \* \* \* \* \* \*  
C  
C ... purpose  
C  
C test program for the MUDPACK solver cud34  
C  
C ... required MUDPACK files  
C  
C cud34.f, cud3.f, cudcom.f, cud3ln.f, cud3pn.f  
C  
C  
C \*\*\*\*\*  
\*  
C  
\*\*\*\*\*  
\*  
C  
\*\*\*\*\*  
\*  
C  
C sample program/test driver for cud34  
C  
C  
\*\*\*\*\*  
\*\*

```

C
*****
**
C
C
c      a sample program/test driver for cud34 is below.
it can be
c      executed as an initial test.  the output is listed
for the
c      test case described.
C
c      test the driver below by solving the complex
nonseparable
c      linear elliptic pde
C
c          cmplx(y,z)*d(dp/dx)/dx +
cmplx(x,z)*d(dp/dy)/dy +
C
c          cmplx(x,y)*d(dp/dz)/dz -
cmplx(y+z,x+y)*pe(x,y,z)
C
c          = r(x,y,z)
C
c      on the box [0.5,1.0] x [0.5,1.0] x [0.0,1.0] with
Dirchlet
c      boundary conditions at the x and y boundaries and
periodic
c      boundary conditions in the z direction.  use point
relaxation
c      and choose a grid 33 by 33 by 97 grid.  use the
exact solution
C
c          pe(x,y,z) =
exp(x*y)*cmplx(cos(tpi*z),sin(tpi*z))
C
c      for testing (tpi = 8.0*atan(1.0)).  First cud3 is
called with
c      iguess=0, maxcy=1 to force one full multigrid
cycle as in tcud3.f.
c      Then, to ensure second order discretization level
error is reached
c      (a requirement for fourth-order mudpack solvers),
3 additional
c      cycles are executed from the finest grid level

```

```
(iguess=1) with
c      cud3. Finally, cud34 is called for a fourth-order
estimate.
c
c ****
c      output (64 bit floating point arithmetic)
c
*****
c
c      cud3 test
c
c      input arguments
c      intl =  0 nxa =  1 nxb =  1 nyc =  1 nyd =  1
c      nze =  0 nzf =  0
c      ixp =  2 jyq =  2 kxr =  3
c      iex =  5 jey =  5 kez =  6
c      nx = 33 ny = 33 nz = 97 iguess = 0 maxcy = 1
c      method = 3 work space length input = 1854498
c      xa = 0.50 xb = 1.00
c      yc = 0.50 yd = 1.00
c      ze = 0.00 zf = 1.00
c      tolmax = 0.000E+00
c
c      multigrid options
c      kcyle = 2
c      iprер = 2
c      ipost = 1
c      interpol = 3
c
c      discretization call to cud3 intl = 0
c      ierror = 0 minimum work space = 1854498
c
c      approximation call to cud3
c      intl = 1 method = 3 iguess = 0 maxcy = 1
c      ierror = 0
c      maximum error = 0.380E-03
c
c      approximation call to cud3
c      intl = 1 method = 3 iguess = 1 maxcy = 3
c      ierror = 0
c      maximum error = 0.381E-03
c
c      cud24 test ierror = 0
c      maximum error = 0.632E-06
```

```

C
C
*****
C      end of output
C
*****
C
C      program tcud34
C      implicit none
C
C      set grid sizes with parameter statements
C
C      integer
iixp,jjyq,kkzr,iiex,jjey,kkez,llwork,nnx,nny,nnz
      parameter(iixp=2,jjyq=2,kkzr=3)
      parameter(iiex=5,jjey=5,kkez=6)
      parameter (nnx = iixp*2** (iiex-1)+1)
      parameter (nny = jjyq*2** (jjey-1)+1)
      parameter (nnz = kkzr*2** (kkez-1)+1)
C
C      set minimal required work space (see tcud3.f) for
this grid
C
C      parameter (llwork = 1854498 )
C
C
C      dimension solution,right hand side, and work
arrays
C
C      complex
phi(nnx,nny,nnz),rhs(nnx,nny,nnz),work(llwork)
      integer iprm(23),mgopt(4)
      real fprm(8)
C
C      put integer and floating point arguments names in
contiguous
C      storeage labelling
C
      integer
intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex,jey,
+
kez,nx,ny,nz,iguess,maxcy,method,meth2,nwork,

```

```

+           lwrkqd,itero

common/itcud3/intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,i
ex,jey,
+
kez,nx,ny,nz,iguess,maxcy,method,meth2,nwork,
+           lwrkqd,itero

      real xa,xb,yc,yd,ze,zf,tolmax,relmax,tpi
      common/ftcud3/xa,xb,yc,yd,ze,zf,tolmax,relmax,tpi
      real dlx,dly,dlz,x,y,z,errm
      complex cxx,cyy,czz,cx,cy,cz,ce,
pxx,pyy,pzz,px,py,pz,pe
      integer i,j,k,ierror
      equivalence(intl,iprm)
      equivalence(xa,fprm)

C
C      declare coefficient and boundary condition input
subroutines external
C
      external cof,bndc
      tpi = 8.0*atan(1.0)
C
C      set for initial call
C
      intl = 0
C
C      set boundary condition flags
C
      nxa = 1
      nxb = 1
      nyc = 1
      nyd = 1
      nze = 0
      nzf = 0
C
C      set grid sizes from parameter statements
C
      ixp = iixp
      jyq = jjyq
      kzr = kkzr
      iex = iiex
      jey = jjey
      kez = kkez

```

```
nx = nnx
ny = nny
nz = nnz
C
C      set for one multigrid cycles
C
maxcy = 1
C
C      set work space length approximation from parameter
statement
C
nwork = llwork
C
C      set method of relaxation--line in the z direction
C
method = 3
C
C      meth2 only used in planar relaxation--but set
C
meth2 = 0
C
C      set full multigrid cycling by flagging no initial
guess at the finest
C      grid level
C
iguess = 0
C
C      set end points of solution region in (x,y,z) space
C
xa = 0.5
xb = 1.0
yc = 0.5
yd = 1.0
ze = 0.0
zf = 1.0
C
C      set default multigrid options
C
mgopt(1) = 2
mgopt(2) = 2
mgopt(3) = 1
mgopt(4) = 3
C
C      set mesh increments
```

```

c
dlx = (xb-xa)/float(nx-1)
dly = (yd-yc)/float(ny-1)
dlz = (zf-ze)/float(nz-1)

c
c      set for no error control
c
tolmax = 0.0

c
c      set right hand side in rhs and phi to zero
c
do k=1,nz
z = ze+(k-1)*dlz
do j=1,ny
y = yc+float(j-1)*dly
do i=1,nx
x = xa+float(i-1)*dlx
call cof(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
rhs(i,j,k) =
cxx*pxx+cyy*pyy+czz*pzz+cx*px+cy*py+cz*pz+ce*pe
phi(i,j,k) = (0.,0.)
end do
end do
end do

c
c      set specified values at x and y boundaries in phi
c
x = xa
do k=1,nz
z = ze+(k-1)*dlz
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(1,j,k) = pe
end do
end do
x = xb
do k=1,nz
z = ze+(k-1)*dlz
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(nx,j,k) = pe

```

```

    end do
    end do
    y = yc
    do k=1,nz
    z = ze+(k-1)*dlz
    do i=1,nx
        x = xa+float(i-1)*dlx
        call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
        phi(i,1,k) = pe
    end do
    end do
    y = yd
    do k=1,nz
    z = ze+(k-1)*dlz
    do i=1,nx
        x = xa+float(i-1)*dlx
        call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
        phi(i,ny,k) = pe
    end do
    end do
    write(6,50)
50 format(//' cud3 test ')
C
C      print input arguments
C

write(6,100)intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex,
,jey,kez,
+
nx,ny,nz,iguess,maxcy,method,nwork,xa,xb,yc,yd,ze,zf,
+
tolmax,mgopt(1),mgopt(2),mgopt(3),mgopt(4)
100 format(/' input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =
',i2,' nyd = 'i2,
+/' nze = ',i2,' nzf = ',i2,
+/' ixp = ',i2,' jyq = 'i2,' kzr = 'i2,
+/' iex = ',i2,' jey = 'i2,' kez = 'i2,
+/' nx = 'i3,' ny = 'i3,' nz = 'i3,' iguess =
'i2,' maxcy = 'i2,
+/' method = 'i2, ' work space length input = ',i7,
+/' xa = 'f5.2,' xb = 'f5.2,
+/' yc = 'f5.2,' yd = 'f5.2,
+/' ze = 'f5.2,' zf = 'f5.2,

```

```

+/' tolmax = ',e10.3
+/' multigrid options '
+/' kcycle = ',i2
+/' iprer = ',i2
+/' ipost = ',i2
+/' interpol = ',i2 )

C
C      discretize pde
C
      write(*,104) intl
104 format(/' discretization call to cud3', ' intl =
', i2)
      call
cud3(iprm,fprm,work,cof,bndc,rhs,phi,mgopt,ierror)
      write (*,105) ierror,iprm(22)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)
C
C      approximate pde
C
      intl = 1
      write(*,106) intl,method,iguess,maxcy
106 format(/' approximation call to cud3 ',
+/' intl = ',i2, ' method = ',i2,' iguess = ',i2, '
maxcy = ',i2)
      call
cud3(iprm,fprm,work,cof,bndc,rhs,phi,mgopt,ierror)
      write (*,107) ierror
107 format(' ierror = ',i2)
      if (ierror.gt.0) call exit(0)
C
C      compute and print maximum error
C
      call error(nx,ny,nz,phi,errm)
      write(*,108) errm
108 format(' maximum error  =  ',e10.3)
C
C      execut three more cycles from finest grid level to
ensure
C      second-order approximation is reached
C
      iguess = 1
      maxcy = 3

```

```

        write(*,106) intl,method,iguess,maxcy
        call
cud3(iprm,fprm,work,cof,bndc,rhs,phi,mgopt,ierror)
        write (*,107) ierror
        if (ierror.gt.0) call exit(0)
        call error(nx,ny,nz,phi,errm)
        write(*,108) errm
C
C      attempt to compute fourth-order estimate
C
        call cud34(work,phi,ierror)
        write(*,109) ierror
109 format (/ ' cud24 test ', ' ierror = ',i2)
        if (ierror.gt.0) call exit(0)
        call error(nx,ny,nz,phi,errm)
        write(*,108) errm
        end

        subroutine error(nx,ny,nz,phi,errm)
C
C      compute the error in the estimate in phi
C
        implicit none
        integer nx,ny,nz
        complex phi(nx,ny,nz)
        real xa,xb,yc,yd,ze,zf,tolmax,relmax,errm,tpi
        common/ftcud3/xa,xb,yc,yd,ze,zf,tolmax,relmax,tpi
        real dlx,dly,dlz,x,y,z
        complex pxx,pyy,pzz,px,py,pz,pe
        integer i,j,k
        dlx = (xb-xa) / (nx-1)
        dly = (yd-yc) / (ny-1)
        dlz = (zf-ze) / (nz-1)
        errm = 0.0
        do k=1,nz
          z = ze+(k-1)*dlz
          do j=1,ny
            y = yc+float(j-1)*dly
            do i=1,nx
              x = xa+float(i-1)*dlx
              call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
              errm = amax1(errm,abs(phi(i,j,k)-pe))
            end do
          end do
        end do
      end

```

```

    end do
    return
end

subroutine cof(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
c
c      input pde coefficients at grid point (x,y,z) in
the solution region
c      to subroutine cud3
c
implicit none
complex cxx,cyy,czz,cx,cy,cz,ce
real x,y,z
cxx = cmplx(y,z)
cyy = cmplx(x,z)
czz = cmplx(x,y)
cx = (0.,0.)
cy = cx
cz = cz
ce = -cmplx(y+z,x+z)
return
end

subroutine bndc(kbdy,xory,yorz,alfa,gbdy)
c
c      dummy subroutine since no mixed derivative b.c.
c
implicit none
integer kbdy
real xory,yorz
complex alfa,gbdy
return
end

subroutine exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
c
c      this subroutine is used to set an exact solution
for testing cud3
c
implicit none
real x,y,z
complex pxx,pyy,pzz,px,py,pz,pe
real xa,xb,yc,yd,ze,zf,tolmax,relmax,tpi
common/ftcud3/xa,xb,yc,yd,ze,zf,tolmax,relmax,tpi

```

```
pe = exp(x*y)*cmplx(cos(tpi*z),sin(tpi*z))
px = y*pe
py = x*pe
pz = tpi*exp(x*y)*cmplx(-sin(tpi*z),cos(tpi*z))
pxx = y*y*pe
pyy = x*x*pe
pzz = -tpi*tpi*pe
return
end
```

---

## TCUD34SP

```
C
C      file tcud34sp.f
C
C      * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *                               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *                               all rights reserved
*
C      *
*
C      *                               MUDPACK version 5.0.1
*
C      *
*
C      *                               A Fortran Package of Multigrid
*
C      *
```

```
*  
C      *          Subroutines and Example Programs  
*  
C      *  
*  
C      *      for Solving Elliptic Partial Differential  
Equations      *  
C      *  
*  
C      *          by  
*  
C      *  
*  
C      *          John Adams  
*  
C      *  
*  
C      *          of  
*  
C      *  
*  
C      *          the National Center for Atmospheric  
Research      *  
C      *  
*  
C      *          Boulder, Colorado (80307)  
U.S.A.      *  
C      *  
*  
C      *          which is sponsored by  
*  
C      *  
*  
C      *          the National Science Foundation  
*  
C      *  
*  
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C  
C ... purpose  
C  
C     test program for the MUDPACK solver cud34sp  
C
```

```

C ... required MUDPACK files
C
C      cud34sp.f, cud3sp.f, cudcom.f
C
C
C ****
*
C ****
*
C
C      sample program/test driver for cud34sp
C
C
C ****
**
*****
****

C
C      a sample program/test driver for cud34sp is below.
it can be
C      executed as an initial test.  the output is listed
for the
C      test case described.
C
C      test cud34sp below by solving the complex
separable elliptic pde
C
C      d(cmplx(x,-x)*dp/dx)/dx + d(cmplx(y,-
y)*dp/dy)/dy +
C
C      d(cmplx(z,-z)*dp/dz)/dz + cmplx(1.,-1.)*(dp/dx
+ dp/dy + dp/dz) +
C
C      cmplx(-(x+y+z),x+y+z)*p(x,y,z) = r(x,y,z)
C
C      on the (x,y,z) region
C
C      1/4 < x < 3/4, 1/3 < y < 2/3, 1/5 < z < 4/5
C
C      with dirchlet boundary conditions at the upper
x,y,z boundaries
C      and the mixed complex derivative boundary

```

```

conditions:
c
c          (1) dp/dx + cmplx(1.,1.)*p(0.5,y,z) =
gbdxa(y,z) at x = 1/4
c
c          (2) dp/dy + cmplx(-1.,-1.)*p(x,0.5,z) =
gbdyc(x,z) at y = 1/3
c
c          (3) dp/dz + cmplx(1.,-1.)*p(x,y,0.5) =
gbdze(x,y) at z = 1/5
c
c      note the complex number multiplying "p" must be
constant along
c      each surface.
c
c      use the exact solution
c
c      pe(x,y,z) = cmplx(exp(x+y),exp(y+z))
c
c      for testing and choose a 49 by 33 by 41 grid.
First cud3sp
c      is called with 5 cycles to ensure second-order
discretization
c      level error is reached (a requirement for mudpack
fourth-order
c      solvers). Then cud34sp is called for a fourth-
order estimate.
c
c ****
c      output (64 bit floating point arithmetic)
c
*****
c
c      cud3sp test
c
c      input arguments
c      intl = 0 nxa = 2 nxb = 1 nyc = 2 nyd = 1
c      nze = 2 nzf = 1
c      ixp = 3 jyq = 2 kzc = 3
c      iex = 5 jey = 5 kez = 5
c      nx = 49 ny = 33 nz = 49 iguess = 0 maxcy = 5
c      method = 0 work space length input = 291605
c      xa = 0.25 xb = 0.75
c      yc = 0.33 yd = 0.67

```

```

C      ze =  0.20  zf =  0.80
C      tolmax =  0.000E+00
C
C      multigrid options
C      kcycle =  2
C      iprер =  2
C      ipost =  1
C      interpol =  3
C
C      discretization call to cud3sp intl =  0
C      ierror =  0 minimum work space =  291605
C
C      approximation call to cud3sp
C      intl =  1 method =  0 iguess =  0 maxcy =  5
C      ierror =  0
C      maximum error =  0.287E-04
C
C      cud34sp test ierror =  0
C      maximum error =  0.592E-07
C
C
C ****
C ****
C      end of output
C
C ****
C ****
C
C      program tcud34sp
C      implicit none
C
C      set grid sizes with parameter statements
C
C      integer
iixp,jjyq,kkzr,iiex,jjey,kkez,llwork,nnx,nny,nnz
parameter(iixp=3,jjyq=2,kkzr=3)
parameter(iiex=5,jjey=5,kkez=5)
parameter (nnx = iixp*2** (iiex-1)+1)
parameter (nny = jjyq*2** (jjey-1)+1)
parameter (nnz = kkzr*2** (kkez-1)+1)
C
C      set minimal work space (see tcud3sp.f)
C

```

```

    parameter (llwork = 291605)
c
c      dimension solution,right hand side, and work
arrays
c
      complex
phi(nnx,nny,nnz),rhs(nnx,nny,nnz),work(llwork)
      integer iprm(23),mgopt(4)
      real fprm(8)
c
c      put integer and floating point arguments names in
contiguous
c      storeage for labelling purposes
c
      integer
intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex,jey,
+
kez,nx,ny,nz,iguess,maxcy,method,nwork,
+
lwrkqd,itero

common/itcd3sp/intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,
iex,jey,
+
kez,nx,ny,nz,iguess,maxcy,method,nwork,
+
lwrkqd,itero
      real xa,xb,yc,yd,ze,zf,tolmax,relmax
      common/ftcud3sp/xa,xb,yc,yd,ze,zf,tolmax,relmax
      real dlx,dly,dlz,x,y,z,errm
      complex cxx,cyy,czz,cx,cy,cz,ce,cex,cey,cez
      complex pxx,pyy,pzz,px,py,pz,pe
      integer i,j,k,ierror
      equivalence(intl,iprm)
      equivalence(xa,fprm)

c
c      declare coefficient and boundary condition input
subroutines external
c
      external cfx,cfy,cfz,bndc
c
c      set for initial call
c
      intl = 0
c
c      set boundary condition flags

```

```
c
nxa = 2
nxb = 1
nyc = 2
nyd = 1
nze = 2
nzf = 1
c
c      set grid sizes from parameter statements
c
ixp = iixp
jyq = jjyq
kzr = kkzr
iex = iiex
jey = jjey
kez = kkez
nx = nnx
ny = nny
nz = nnz
c
c      set 5 multigrid cycles
c
maxcy = 5
c
c      set work space length approximation from parameter
statement
c
nwork = llwork
c
c      set point relaxation (only choice with cud3sp)
c
method = 0
c
c      set full multigrid cycling beginning at coarsest
grid
c
iguess = 0
c
c      set end points of solution region in (x,y,z) space
c
xa = 0.25
xb = 0.75
yc = 1.0/3.0
yd = 2.0/3.0
```

```

ze = 0.20
zf = 0.80
C
C      set default multigrid options
C
        mgopt(1) = 2
        mgopt(2) = 2
        mgopt(3) = 1
        mgopt(4) = 3
C
C      set mesh increments
C
        dlx = (xb-xa)/float(nx-1)
        dly = (yd-yc)/float(ny-1)
        dlz = (zf-ze)/float(nz-1)
C
C      set for no error control
C
        tolmax = 0.0
C
C      set right hand side in rhs and phi to zero
C
        do k=1,nz
        z = ze+(k-1)*dlz
        call cfz(z,czz,cz,cez)
        do j=1,ny
        y = yc+float(j-1)*dly
        call cfy(y,cyy,cy,cey)
        do i=1,nx
        x = xa+float(i-1)*dlx
        call cfx(x,cxx,cx,cex)
        call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
        ce = cex+cey+cez
        rhs(i,j,k) =
cxx*pxx+cyy*pyy+czz*pzz+cx*px+cy*py+cz*pz+ce*pe
        phi(i,j,k) = (0.,0.)
        end do
        end do
        end do
C
C      set specified values at upper x,y,z boundaries in
phi
C
        x = xb

```

```

    do k=1,nz
    z = ze+(k-1)*dlz
    do j=1,ny
        y = yc+float(j-1)*dly
        call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
        phi(nx,j,k) = pe
    end do
    end do
    y = yd
    do k=1,nz
    z = ze+(k-1)*dlz
    do i=1,nx
        x = xa+float(i-1)*dlx
        call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
        phi(i,ny,k) = pe
    end do
    end do
    z = zf
    do j=1,ny
    y = yc+(j-1)*dly
    do i=1,nx
        x = xa+float(i-1)*dlx
        call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
        phi(i,j,nz) = pe
    end do
    end do
    write(6,50)
50 format(//' cud3sp test ')
C
C      print input arguments
C

write(6,100)intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex
,jey,kez,
+
nx,ny,nz,iguess,maxcy,method,nwork,xa,xb,yc,yd,ze,zf,
+
tolmax,mgopt(1),mgopt(2),mgopt(3),mgopt(4)
100 format(/' input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =
',i2,' nyd = 'i2,
+/' nze = ',i2,' nzf = ',i2,
+/' ixp = ',i2,' jyq = 'i2,' kzr = 'i2,
+/' iex = ',i2,' jey = 'i2,' kez = 'i2,

```

```

+/' nx = 'i3,' ny = 'i3,' nz = 'i3, ' iguess =
'i2,' maxcy = 'i2,
+/' method = 'i2, ' work space length input = ',i7,
+/' xa = 'f5.2,' xb = 'f5.2,
+/' yc = 'f5.2,' yd = 'f5.2,
+/' ze = 'f5.2,' zf = 'f5.2,
+/' tolmax = ',e10.3
+/' multigrid options '
+/' kcycle = ',i2
+/' iprer = ',i2
+/' ipost = ',i2
+/' interpol = ',i2 )

C
C      discretize pde only
C
      write(*,104) intl
104 format(/' discretization call to cud3sp', ' intl =
', i2)
      call
cud3sp(iprm,fprm,work,cfx,cfy,cfz,bndc,rhs,phi,mgopt,ier
ror)
      write (*,105) ierror,iprm(21)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)
C
C      approximate pde
C
      intl = 1
      write(*,106) intl,method,iguess,maxcy
106 format(/' approximation call to cud3sp ',
+/' intl = ',i2, ' method = ',i2,' iguess = ',i2, '
maxcy = ',i2)
      call
cud3sp(iprm,fprm,work,cfx,cfy,cfz,bndc,rhs,phi,mgopt,ier
ror)
      write (*,107) ierror
107 format(' ierror = ',i2)
      if (ierror.gt.0) call exit(0)
C
C      compute and print maximum error
C
      call error(nx,ny,nz,phi,errm)
      write(*,108) errm

```

```

108 format(' maximum error = ',e10.3)
c
c      attempt fourth-order estimate
c
      call cud34sp(work,phi,ierror)
      write(*,109) ierror
109 format(/ ' cud34sp test ', ' ierror = ',i2)
      if (ierror.gt.0) call exit(0)
      call error(nx,ny,nz,phi,errm)
      write(*,108) errm
      end

      subroutine error(nx,ny,nz,phi,errm)
c
c      compute the error in the estimate in phi
c
      implicit none
      integer nx,ny,nz
      complex phi(nx,ny,nz)
      real xa,xb,yc,yd,ze,zf,tolmax,relmax,errm
      common/ftcud3sp/xa,xb,yc,yd,ze,zf,tolmax,relmax
      real dlx,dly,dlz,x,y,z
      complex pxx,pyy,pzz,px,py,pz,pe
      integer i,j,k
      dlx = (xb-xa) / (nx-1)
      dly = (yd-yc) / (ny-1)
      dlz = (zf-ze) / (nz-1)
      errm = 0.0
      do k=1,nz
      z = ze+(k-1)*dlz
      do j=1,ny
      y = yc+float(j-1)*dly
      do i=1,nx
      x = xa+float(i-1)*dlx
      call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
      errm = amax1(errm,abs(phi(i,j,k)-pe))
      end do
      end do
      end do
      return
      end

c
c      complex coefficient subroutines
c

```

```

subroutine cfx(x,cxx,cx,cex)
implicit none
real x
complex cxx,cx,cex
cxx = cmplx(x,-x)
cx = (1.,-1.)
cex = cmplx(-x,x)
return
end
subroutine cfy(y,cyy,cy,cey)
implicit none
real y
complex cyy, cy, cey
cyy= cmplx(y,-y)
cy = (1.,-1.)
cey = cmplx(-y,y)
return
end
subroutine cfz(z,czz,cz,cez)
implicit none
real z
complex czz,cz,cez
czz = cmplx(z,-z)
cz = (1.,-1.)
cez = cmplx(-z,z)
return
end

subroutine bndc(kbdy,xory,yorz,cons,gbdy)
C
C      input complex derivative boundary conditions to
cud3sp
C
implicit none
integer kbdy
real xory,yorz,x,y,z
complex cons,gbdy,pe,px,py,pz,pzx,pyy,pzz
real xa,xb,yc,yd,ze,zf,tolmax,relmax
common/ftcud3sp/xa,xb,yc,yd,ze,zf,tolmax,relmax
if (kbdy.eq.1) then
C
C      x=xa surface
C
x = xa

```

```

y = xorY
z = yorz
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
cons = (1.,1.)
gbdy = px+cons*pe
return
end if
if (kbdf.eq.3) then
C
C      y=yc surface
C
y = yc
x = xorY
z = yorz
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
cons = (-1.,-1.)
gbdy = py+cons*pe
return
end if
if (kbdf.eq.5) then
C
C      z=ze surface
C
z = ze
x = xorY
y = yorz
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
cons = (1.,-1.)
gbdy = pz + cons*pe
return
end if
end

subroutine exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
C
C      this subroutine is used to set an exact solution
for testing cud3sp
C      (i.e., setting the rhs, boundary conditions and
computing the exact
C      error)
C
implicit none
real x,y,z,exy,eyz
complex pe,px,py,pz,pxx,pyy,pzz

```

```

exy = exp(x+y)
eyz = exp(y+z)
pe = cmplx(exy,eyz)
px = cmplx(exy,0.)
pxx = px
py = pe
pyy = pe
pz = cmplx(0.,eyz)
pzz = pz
return
end

```

TCUD3CR

```
C      file tcud3cr.f
C
C      * * * * * * * * * * * * * * * *
*
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *               University Corporation for Atmospheric
Research          *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK version 5.0.1
*
C      *
*
C      *               A Fortran Package of Multigrid
```



```

C      test program for the MUDPACK solver cud3cr
C
C ... required MUDPACK files
C
C      cud3cr.f, cudcom.f
C
C
C
***** ****
*
C
***** ****
*
C
C      sample program/test driver for cud3cr (see
cud3cr.d)
C
C
***** ****
**
C
***** ****
**
C
C      a sample program/test driver for cud3cr is below.
it can be
C      executed as an initial test. The output from
executing
C      the code in this file is listed after the problem
description.
C
C      Problem Description:
C
C      test cud3cr by solving the complex nonseparable 3-
d elliptic pde
C      with cross derivative terms:
C
C          cxx*pxx + cyy*pyy + czz*pzz + cx*px + cy*py +
cz*pz + ce*pe +
C
C          cxy*pxy + cxz*pxz + cyz*pyz = r(x,y,z) .
C
C      on the region 0 < x < 1, 0 < y < (pi+pi), 0 < z <
1.

```

```

c      let s = sin(y), c = cos(y).  the complex
coefficients are
c      given by:
c
c      cxx = cmplx(1.0+0.5*s*z,1.0+0.5*c*z)
c
c      cyy = (1.,1.)+cmplx(x,z)
c
c      czz = cmplx(1.0+0.5*s*x,1.0+0.5*c*x)
c
c      cx = cy = cz = (0.0,0.0)
c
c      ce = -cmplx(z,x)
c
c      cxy =
cmplx(sqrt(real(cxx)*real(cyy),sqrt(aimag(cxx)*aimag(cyy
)))
c
c      cxz = (0.0,0.0)
c
c      cyz =
cmplx(sqrt(real(cyy)*real(czz),sqrt(aimag(cyy)*aimag(czz
)))
c
c      assume the solution is periodic in y and is
specified at x=0 and z=0.
c      further assume mixed oblique derivative conditions
of the form
c
c      px + cmplx(z,1-z)*py + cmplx(s,c)*pz -
p(1,y,z) = g(y,z)
c
c      at x=1 and mixed normal derivative conditions of
the form
c
c      pz + cmplx(x,s*s)*p(x,y,1) = h(x,y)
c
c      at z=1.  for testing purposes, use the exact
solution
c
c      p(x,y,z) = exp(x*z)*cmplx(c,s)
c
c      to set boundary conditions, the right hand side,
and compute

```

```
c      exact error.  results from approximating this
problem on a
c      25 by 65 by 25 x-y-z grid using cud3cr are given
below.  point
c      relaxation and an error tolerance of .001 are
used.
c
c ****
c      output (32 bit floating point arithmetic)
c
***** ****
c
c      cud3cr test
c
c      input arguments
c      intl = 0
c      nxa = 1 nxb = 2
c      nyc = 0 nyd = 0
c      nze = 1 nzf = 2
c      ixp = 3 jyq = 2 kzs = 3
c      iex = 3 jey = 6 kez = 3
c      nx = 13 ny = 65 nz = 13
c      iguess = 0 maxcy = 2
c      method = 0 work space input = 173394
c      xa = 0.00 xb = 1.00
c      yc = 0.00 yd = 6.28
c      ze = 0.00 zf = 1.00
c      tolmax = 0.000E+00
c
c      multigrid options
c      mgopt(1) = 0
c
c      new cud3cr arguments
c      icrs
c          1    0    1
c      tol = 0.0010
c      maxit = 10
c
c      initial call
c      intl = 0
c      iguess = 0
c      ierror = 0
c      minimum required work space length = 173394
c
```

```

c      approximation call
c      intl = 1
c      iguess = 0
c      ierror = 0
c      number of outer iterations executed = 9
c      relative difference profile:
c      0.6567 0.1462 0.0638 0.0203 0.0099 0.0051
0.0016 0.0011 0.0004
c      exact least squares error = 0.127E-02
c
c
***** *****
c      end of output
c
*****
***** *****
*****
c
      program tcd3cr
      implicit none
c
c      set grid sizes and predetermined minimal required
equired work
c      with parameter statements
c
      integer
iixp,jjyq,kkzr,iiex,jjey,kkez,nnx,nny,nnz,llengt,mmaxit
      parameter(iixp=3,jjyq=2,kkzr=3)
      parameter (iiex=3,jjey=6,kkez=3)
      parameter(nnx = iixp*2** (iiex-1)+1)
      parameter(nny = jjyq*2** (jjey-1)+1)
      parameter(nnz = kkzr*2** (kkez-1)+1)
      parameter(llengt=173394)
      parameter(mmaxit = 10)
      complex
rhs(nnx,nny,nnz),phi(nnx,nny,nnz),work(llengt)
      integer iparm(23),mgopt(4),icrs(3)
      integer i,j,k,ierror,maxit,iouter
      real fparm(8),rmax(mmaxit)

c
c      use labelled common to identify integer and
floating point arguments
c

```

```

    integer
intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex,jey,
+kez,nx,ny,nz,iguess,maxcy,method,meth2,nwork,lwkmin,ite
ro
    common
/iprm/intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex,jey,
+kez,nx,ny,nz,iguess,maxcy,method,meth2,nwork,lwkmin,ite
ro
    real xa,xb,yc,yd,ze,zf,tolmax,relmax
    common/fprm/xa,xb,yc,yd,ze,zf,tolmax,relmax
    real pi,dx,dy,dz,c,s,x,y,z,tol,err2
    complex cxx,cyy,czz,cx,cy,cz,ce,cxy,cyz
    complex pxx,pyy,pzz,px,py,pz,pxy,pxz,pyz,pe
    equivalence(iparm,intl)
    equivalence(fparm,xa)
    external cof,bd3cr,cxyf,cxzf,cyzf
    pi = 4.*atan(1.)

C
C      set interval end points
C
    xa = 0.0
    xb = 1.0
    yc = 0.0
    yd = pi+pi
    ze = 0.0
    zf = 1.0
C
C      set required no error control within multigrid
cycling
C
    tolmax = 0.0
C
C      set integer input arguments
C
    intl = 0
C
C      set boundary condition flags
C
    nxa = 1
    nxb = 2
    nyc = 0
    nyd = 0

```

```

nze = 1
nzf = 2
C
C      set grid size arguments from parameter statements
C
ixp = iixp
jyq = jjyq
kzr = kkzr
iex = iiex
jey = jjey
kez = kkez
nx = nnx
ny = nny
nz = nnz
C
C      flag nonzero xy and yz and zero xz cross
derivatives terms
C
icrs(1) = 1
icrs(2) = 0
icrs(3) = 1
C
C      set two multigrid cycles per outer iteration
C
maxcy = 2
C
C      set point relaxation
C
method = 0
meth2 = 0
C
C      set work space length input
C
nwork = llengt
C
C      set uniform grid interval lengths in each
dimension
C
dx = (xb-xa) / (nx-1)
dy = (yd-yc) / (ny-1)
dz = (zf-ze) / (nz-1)
C
C      set right hand side and preset solution to zero
C      this also sets specified (Dirchlet) b.c. in phi

```

```

c
      do j=1,ny
y = (j-1)*dy
s = sin(y)
do k=1,nz
      z = (k-1)*dz
      do i=1,nx
          x = (i-1)*dx
          call
exact(x,y,z,pe,px,py,pz,pxx,pyy,pzz,pxy,pxz,pyz)
          call cof(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
          call cxyf(x,y,z,cxy)
          call cyzf(x,y,z,cyz)
          rhs(i,j,k) =
cxx*pss+cyy*pss+czz*pss+cxy*pss+cyz*pss+
          +           cx*px+cy*py+cz*pz+ce*pe
          phi(i,j,k) = 0.0
      end do
      end do
      end do
c
c      set default multigrid options
c
      mgopt(1) = 0
c
c      set error control tolerance and outer iteration
limit of 10
c
      tol = .001
      maxit = mmaxit
c
c      print input arguments shared with cud3
c
      write(*,50)
50 format(//' cud3cr test ' )

write(6,100)intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex
,jey,kez,
+
nx,ny,nz,iguess,maxcy,method,nwork,xa,xb,yc,yd,ze,zf,
      +           tolmax,mgopt(1)
100 format(/' input arguments ',
+/' intl = ',i2,
+/' nxa = ',i2,' nxb = ',i2,

```

```

+/' nyc = ',i2,' nyd = ',i2,
+/' nze = ',i2,' nzf = ',i2,
+/' ixp = ',i2,' jyq = 'i2,' kzs = 'i2,
+/' iex = ',i2, ' jey = 'i2, ' kez = 'i2,
+/' nx = 'i3,' ny = 'i3,' nz = 'i3,
+/' iguess = 'i2,' maxcy = 'i2,
+/' method = 'i2, ' work space input = ',i7,
+/' xa = 'f5.2,' xb = 'f5.2,
+/' yc = 'f5.2,' yd = 'f5.2,
+/' ze = 'f5.2,' zf = 'f5.2,
+/' tolmax = ' ,e10.3
+/// multigrid options '
+/' mgopt(1) = ',i2)

C
C      print new cud3cr arguments
C
C      write(*,101) (icrs(i),i=1,3),tol,maxit
101 format(/' new cud3cr arguments ', / ' icrs '/ 3i5,
           +' tol = ',f6.4 /' maxit = ',i3)
C
C *** initialization call
C
        intl = 0
        iguess = 0
        call
cud3cr(iparm,fparm,work,cof,bd3cr,rhs,phi,mgopt,icrs,
       +cxyf,cxzf,cyzf,tol,maxit,iouter,rmax,ierror)
        write (6,200) intl,iguess,ierror,iparm(22)
200 format(/' initial call', /' intl = ',i2, /' iguess
= ',i2,
           +' ierror = ',i2, /' minimum required work space
length = ',i8)
        if (ierror.gt.0) call exit(0)
C
C *** noninitial call
C
        intl = 1
        iguess = 0
        call
cud3cr(iparm,fparm,work,cof,bd3cr,rhs,phi,mgopt,icrs,
       +cxyf,cxzf,cyzf,tol,maxit,iouter,rmax,ierror)
        write (6,201)
intl,iguess,ierror,iouter,(rmax(i),i=1,iouter)
201 format(/' approximation call ', /' intl = ',i2, /

```

```

iguess = ',i2,
      +' ierror = ',i3,
      +' number of outer iterations executed = ',i3,
      +' relative difference profile:', / (10(f6.4,2x)) )

c
c      compute and print exact least squares error after
iouter iterations
c
      err2 = 0.
      do j=1,ny
      y = (j-1)*dy
      c = cos(y)
      do k=1,nz
      z = (k-1)*dz
      do i=1,nx
      x = (i-1)*dx
      pe = cmplx((x*c*z)**3,0.0)
      err2 = err2 + cabs(pe - phi(i,j,k))**2
      end do
      end do
      end do
      err2 = sqrt(err2/(nx*ny*nz))
      write(6,202) err2
202 format(' exact least squares error = ',e10.3)
      end

      subroutine
exact(x,y,z,pe,px,py,pz,pxx,pyy,pzz,pxy,pxz,pyz)
c
c      set exact solution and partial derivatives:
p(x,y,z) = (x*cos(y)*z)**3
c
      implicit none
      real x,y,z
      complex pe,px,py,pz,pxx,pyy,pzz,pxy,pxz,pyz
      real s,c,exz
      real xc,xz,cz,xcz
      c = cos(y)
      s = sin(y)
      xc = x*c
      xz = x*z
      cz = c*z
      xcz = xc*z
      pe = cmplx(xcz**3,0.0)

```

```

px = cmplx(3.*x*x*cz**3,0.0)
pxx = cmplx(6.*x*cz**3,0.0)
py = cmplx(-3.*c*c*s*xz**3,0.0)
pyy = cmplx(-3.*xz**3*(c**3-2.*c*s**2),0.0)
pz = cmplx(3.*z*z*xc**3,0.0)
pzz = cmplx(6.*z*xc**3,0.0)
pxy = cmplx(-9.*x**2*c*c*s*z**3,0.0)
pxz = cmplx(9.*xz**2*c**3,0.0)
pyz = cmplx(-9.*c*c*s*x**3*z**2,0.0)
return
exz = exp(x*z)
pe = exz*cmplx(c,z)
px = z*pe
py = -exz*s
pz = x*pe +exz*cmplx(c,1.0)
pxx = z*px
pyy = -exz*c
pzz = x*(pz+exz*cmplx(c,1.0))
return
end

subroutine cof(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
C
C set noncross derivative pde coefficients
C
implicit none
real x,y,z,s,c
complex cxx,cyy,czz,cx,cy,cz,ce
s = sin(y)
c = cos(y)
C cxx = 1.+0.5*s*z
C cyy = 1.+x*z
C czz = 1.+0.5*x*s
C cx = 0.0
C cy = -x*z
C cz = 0.0
C ce = -(x+z)
C return
cxx = cmplx(1.+0.5*s*z, 1.+0.5*c*z)
cyy = cmplx(1.+x,1.+z)
czz = cmplx(1+0.5*s*x,1.+0.5*c*x)
cx = (0.,0.)
cy = cx
cz = cz

```

```

ce = -cmplx(x+z,0.0)
return

cxx = cmplx(1.+0.5*s*z,0.0)
cyy = cmplx(1.+x*z,0.0)
czz = cmplx(1.+0.5*x*s,0.0)
cx = (0.0,0.0)
cy = cmplx(-x*z,0.0)
cz = (0.0,0.0)
ce = cmplx(-(x+z),0.0)
return

cxx = (1.0,1.0)+0.5*cmplx(s*z,c*z)
cyy = (1.0,1.0)+cmplx(x,z)
czz = (1.0,1.0)+0.5*cmplx(x*s,x*c)
cx = (0.0,0.0)
cy = cmplx(-x*z,x*z)
cz = (0.0,0.0)
ce = -cmplx((x+z),0.0)
return
end

subroutine cxyf(x,y,z,cxy)
C
C      set x-y cross term coefficient at (x,y,z)
C
implicit none
real x,y,z,cxxr,cxxi,cyyr,cyyi,cxyr,cxyi
complex cxx,cyy,czz,cx, cy, cz, ce, cxy
call cof(x,y,z,cxx,cyy,czz,cx, cy, cz, ce)
cxxr = cxx
cxxi = aimag(cxx)
cyyr = cyy
cyyi = aimag(cyy)
cxyr = sqrt(cxxr*cyyr)
cxyi = sqrt (cxxi*cyyi)
cxy = cmplx(cxyr,cxyi)
return
end

subroutine cxzf(x,y,z,cxz)
C
C      this is a dummy subroutine since cxz=0.0 for all
(x,y,z)

```

```

c
      return
end

subroutine cyzf(x,y,z,cyz)
c
c      set y-z cross term coefficient at (x,y,z)
c
      real x,y,z,czzr,czzi,cyyr,cyyi,cyxr,cyzi
      complex cxx,cyy,czz,cx,cy,cz,ce,cyz
      call cof(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
      cyxr = cyy
      cyyi = aimag(cyy)
      czzr = czz
      czzi = imag(czz)
      cyxr = sqrt(cyyr*cyyi)
      cyzi = sqrt (cyyi*czzi)
      cyz = cmplx(cyxr, cyzi)
      return
end

subroutine bd3cr(kbdy,xory,yorz,a,b,c,g)
c
c      pass mixed derivative boundary conditions to
cud3cr
c
      implicit none
      integer kbdy
      real xory, yorz
      complex a,b,c,g,x,y,z
      complex pe,px,py,pz,pxx,pyy,pzz,pxy,pxz,pyz
      if (kbdy.eq.2) then
c
c      upper x boundary (mixed oblique)
c
      x = 1.0
      y = xory
      z = yorz
      a = cmplx(z*(1.-z),0.0)
      b = cmplx(sin(y)*cos(y),0.0)
      c = -(1.0,0.0)
      a = cmplx(z,1.-z)
      b = cmplx(sin(y),cos(y))
      c = - (1.0,1.0)

```

```

call exact(x,y,z,pe,px,py,pz,pxx,pyy,pzz,pxy,pxz,pyz)
g = px + a*py + b*pz + c*pe
return
    else if (kbdy.eq.6) then
C
C      upper z boundary (mixed normal)
C
z = 1.0
x = xory
y = yorz
C      a = (0.0,0.0)
C      b = (0.0,0.0)
c = cmplx(x,sin(y)**2)
C      c = x*sin(y)**2
call exact(x,y,z,pe,px,py,pz,pxx,pyy,pzz,pxy,pxz,pyz)
g = pz + a*px + b*py + c*pe
return
    end if
end

```

## TCUD3SP

```

C
C      file tcud3sp.f
C
C      * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *                               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *                               all rights reserved

```

\*  
C \*  
\*  
C \* MUDPACK version 5.0.1  
\*  
C \*  
\*  
C \* A Fortran Package of Multigrid  
\*  
C \*  
\*  
C \* Subroutines and Example Programs  
\*  
C \*  
\*  
C \* for Solving Elliptic Partial Differential  
Equations \*  
C \*  
\*  
C \* by  
\*  
C \*  
\*  
C \* John Adams  
\*  
C \*  
\*  
C \* of  
\*  
C \*  
\*  
C \* the National Center for Atmospheric  
Research \*  
C \*  
\*  
C \* Boulder, Colorado (80307)  
U.S.A. \*  
C \*  
\*  
C \* which is sponsored by  
\*  
C \*  
\*  
C \* the National Science Foundation

```

*
C      *
*
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * *
C
C ... purpose
C
C      test program for the MUDPACK solver cud3sp
C
C ... required MUDPACK files
C
C      cud3sp.f, cudcom.f
C
C
C
C ****
*
C
C ****
*
C
C      sample program/test driver for cud3sp
C
C
C ****
**
C
C ****
**
C
C      a sample program/test driver for cud3sp is below.
it can be
C      executed as an initial test. the output is listed
for the
C      test case described.
C
C      test cud3sp below by solving the complex separable
elliptic pde
C
C      d(cmplx(x,-x)*dp/dx)/dx + d(cmplx(y,-
y)*dp/dy)/dy +
C
C      d(cmplx(z,-z)*dp/dz)/dz + cmplx(1.,-1.)*(dp/dx

```

```

+ dp/dy + dp/dz) +
C
C      cmplx(-(x+y+z),x+y+z)*p(x,y,z) = r(x,y,z)
C
C      on the (x,y,z) region
C
C      1/4 < x < 3/4, 1/3 < y < 2/3, 1/5 < z < 4/5
C
C      with dirchlet boundary conditions at the upper
x,y,z boundaries
C      and the mixed complex derivative boundary
conditions:
C
C      (1) dp/dx + cmplx(1.,1.)*p(0.5,y,z) =
gbdxa(y,z) at x = 1/4
C
C      (2) dp/dy + cmplx(-1.,-1.)*p(x,0.5,z) =
gbdyc(x,z) at y = 1/3
C
C      (3) dp/dz + cmplx(1.,-1.)*p(x,y,0.5) =
gbdze(x,y) at z = 1/5
C
C      note the complex number multiplying "p" must be
constant along
C      each surface.
C
C      use the exact solution
C
C      pe(x,y,z) = cmplx(exp(x+y),exp(y+z))
C
C      for testing and choose a 49 by 33 by 49 grid.
C
C ****
C      output (32 bit floating point arithmetic)
C
C ****
C
C      cud3sp test
C
C      input arguments
C      intl = 0 nxa = 2 nxb = 1 nyc = 2 nyd = 1
C      nze = 2 nzf = 1
C      ixp = 3 jyq = 2 kzs = 3
C      iex = 5 jey = 5 kez = 5

```

```

C      nx =  49 ny =  33 nz =  49 iguess =  0 maxcy =  3
C      method =  0 work space length input =  318622
C      xa =  0.25 xb =  0.75
C      yc =  0.33 yd =  0.67
C      ze =  0.20 zf =  0.80
C      tolmax =  0.000E+00
C
C      multigrid options
C      kcycle =  2
C      iprer =  2
C      ipost =  1
C      interpol =  3
C
C      discretization call to cud3sp intl =  0
C      ierror =  0 minimum work space =  291605
C
C      approximation call to cud3sp
C      intl =  1 method =  0 iguess =  0 maxcy =  3
C      ierror =  0
C      maximum error =  0.330E-04
C
C
C ****
C      end of output
C
C ****
C
C      program tcud3sp
C      implicit none
C
C      set grid sizes with parameter statements
C
C      integer
iixp,jjyq,kkzr,iiex,jjey,kkez,llwork,nnx,nny,nnz
      parameter(iixp=3,jjyq=2,kkzr=3)
      parameter(iiex=5,jjey=5,kkez=5)
      parameter (nnx = iixp*2** (iiex-1)+1)
      parameter (nny = jjyq*2** (jjey-1)+1)
      parameter (nnz = kkzr*2** (kkez-1)+1)
C
C      set work space length estimate (see cud3sp.d) .

```

```

c      this will probably overestimate required space
c
c      parameter (llwork = 7* (nnx+2) * (nny+2) * (nnz+2) /2 )
c
c
c      dimension solution,right hand side, and work
c      arrays
c
c      complex
phi(nnx,nny,nnz),rhs(nnx,nny,nnz),work(llwork)
      integer iprm(23),mgopt(4)
      real fprm(8)
c
c      put integer and floating point arguments names in
c      contiguous
c      storeage labelling
c
c      integer
intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex,jey,
      +
kez,nx,ny,nz,iguess,maxcy,method,nwork,
      +
lwrkqd,itero

common/itcd3sp/intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,
iex,jey,
      +
kez,nx,ny,nz,iguess,maxcy,method,nwork,
      +
lwrkqd,itero
      real xa,xb,yc,yd,ze,zf,tolmax,relmax
      common/ftcud3sp/xa,xb,yc,yd,ze,zf,tolmax,relmax
      real dlx,dly,dlz,x,y,z,errm
      complex cxx,cyy,czz,cx,cy,cz,ce,cex,cey,cez
      complex pxx,pyy,pzz,px,py,pz,pe
      integer i,j,k,ierror
      equivalence(intl,iprm)
      equivalence(xa,fprm)

c
c      declare coefficient and boundary condition input
c      subroutines external
c
c      external cfx,cfy,cfz,bndc
c
c      set for initial call
c

```

```
    intl = 0
c
c      set boundary condition flags
c
    nxa = 2
    nxb = 1
    nyc = 2
    nyd = 1
    nze = 2
    nzf = 1
c
c      set grid sizes from parameter statements
c
    ixp = iixp
    jyq = jjyq
    kzs = kkzs
    iex = iiex
    jey = jjey
    kez = kkez
    nx = nnx
    ny = nny
    nz = nnz
c
c      set for 3 multigrid cycles
c
    maxcy = 3
c
c      set work space length approximation from parameter
statement
c
    nwork = llwork
c
c      set point relaxation (only choice with cud3sp)
c
    method = 0
c
c      set full multigrid cycling by flagging no initial
guess at the finest
c      grid level
c
    iguess = 0
c
c      set end points of solution region in (x,y,z) space
c
```

```

xa = 0.25
xb = 0.75
yc = 1.0/3.0
yd = 2.0/3.0
ze = 0.20
zf = 0.80
C
C      set default multigrid options
C
mgopt(1) = 2
mgopt(2) = 2
mgopt(3) = 1
mgopt(4) = 3
C
C      set mesh increments
C
dlx = (xb-xa)/float(nx-1)
dly = (yd-yc)/float(ny-1)
dlz = (zf-ze)/float(nz-1)
C
C      set for no error control
C
tolmax = 0.0
C
C      set right hand side in rhs and phi to zero
C
do k=1,nz
z = ze+(k-1)*dlz
call cfz(z,czz,cz,cez)
do j=1,ny
y = yc+float(j-1)*dly
call cfy(y,cyy,cy,cey)
do i=1,nx
x = xa+float(i-1)*dlx
call cfx(x,cxx,cx,cex)
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
ce = cex+cey+cez
rhs(i,j,k) =
cxx*pxx+cyy*pyy+czz*pzz+cx*px+cy*py+cz*pz+ce*pe
phi(i,j,k) = (0.,0.)
end do
end do
end do
C

```

```

c      set specified values at upper x,y,z boundaries in
phi
c
x = xb
do k=1,nz
z = ze+(k-1)*dlz
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(nx,j,k) = pe
end do
end do
y = yd
do k=1,nz
z = ze+(k-1)*dlz
do i=1,nx
x = xa+float(i-1)*dlx
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(i,ny,k) = pe
end do
end do
z = zf
do j=1,ny
y = yc+ (j-1)*dly
do i=1,nx
x = xa+float(i-1)*dlx
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(i,j,nz) = pe
end do
end do
write(6,50)
50 format(//' cud3sp test ')
c
c      print input arguments
c

write(6,100)intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex
,jey,kez,
+
nx,ny,nz,iguess,maxcy,method,nwork,xa,xb,yc,yd,ze,zf,
+
tolmax,mgopt(1),mgopt(2),mgopt(3),mgopt(4)
100 format(/' input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =

```

```

',i2,' nyd = 'i2,
+/' nze = ',i2, ' nzf = ',i2,
+/' ixp = ',i2,' jyq = 'i2,' kxr = 'i2,
+/' iex = ',i2, ' jey = 'i2, ' kez = 'i2,
+/' nx = 'i3,' ny = 'i3,' nz = 'i3, ' iguess =
'i2,' maxcy = 'i2,
+/' method = 'i2, ' work space length input = ',i7,
+/' xa = 'f5.2,' xb = 'f5.2,
+/' yc = 'f5.2,' yd = 'f5.2,
+/' ze = 'f5.2,' zf = 'f5.2,
+/' tolmax = ' ,e10.3
+/' multigrid options '
+/' kcycle = ',i2
+/' iprер = ',i2
+/' ipost = ',i2
+/' interpol = ',i2 )

C
C      discretize pde
C
      write(*,104) intl
104 format(/' discretization call to cud3sp', ' intl =
', i2)
      call
cud3sp(iprm,fprm,work,cfx,cfy,cfz,bndc,rhs,phi,mgopt,ierr
ror)
      write (*,105) ierror,iprm(21)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)
C
C      approximate pde
C
      intl = 1
      write(*,106) intl,method,iguess,maxcy
106 format(/' approximation call to cud3sp ',
+/' intl = ',i2, ' method = ',i2,' iguess = ',i2, '
maxcy = ',i2)
      call
cud3sp(iprm,fprm,work,cfx,cfy,cfz,bndc,rhs,phi,mgopt,ierr
ror)
      write (*,107) ierror
107 format(' ierror = ',i2)
      if (ierror.gt.0) call exit(0)
C

```

```

c      compute and print maximum error
c
c      call error(nx,ny,nz,phi,errm)
c      write(*,108) errm
108 format(' maximum error = ',e10.3)
      end

      subroutine error(nx,ny,nz,phi,errm)
c
c      compute the error in the estimate in phi
c
      implicit none
      integer nx,ny,nz
      complex phi(nx,ny,nz)
      real xa,xb,yc,yd,ze,zf,tolmax,relmax,errm
      common/ftcud3sp/xa,xb,yc,yd,ze,zf,tolmax,relmax
      real dlx,dly,dlz,x,y,z
      complex pxx,pyy,pzz,px,py,pz,pe
      integer i,j,k
      dlx = (xb-xa) / (nx-1)
      dly = (yd-yc) / (ny-1)
      dlz = (zf-ze) / (nz-1)
      errm = 0.0
      do k=1,nz
      z = ze+(k-1)*dlz
      do j=1,ny
      y = yc+float(j-1)*dly
      do i=1,nx
      x = xa+float(i-1)*dlx
      call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
      errm = amax1(errm,abs(phi(i,j,k)-pe))
      end do
      end do
      end do
      return
      end

c
c      complex coefficient subroutines
c
      subroutine cfx(x,cxx,cx,cex)
      implicit none
      real x
      complex cxx,cx,cex
      cxx = cmplx(x,-x)

```

```

cx = (1.,-1.)
cex = cmplx(-x,x)
return
end
subroutine cfy(y,cyy,cy,cey)
implicit none
real y
complex cyy, cy, cey
cyy= cmplx(y,-y)
cy = (1.,-1.)
cey = cmplx(-y,y)
return
end
subroutine cfz(z,czz,cz,cez)
implicit none
real z
complex czz, cz, cez
czz = cmplx(z,-z)
cz = (1.,-1.)
cez = cmplx(-z,z)
return
end

subroutine bndc(kbdy,xory,yorz,cons,gbdy)
C
C      input complex derivatives of the form
C
C          dp/dw + cons*p = gbdy   (w=x,y,z)
C
C      to cud3sp. "cons" must be a complex constant at
each surface
C
implicit none
integer kbdy
real xory,yorz,x,y,z
complex cons,gbdy,pe,px,py,pz,pxx,pyy,pzz
real xa,xb,yc,yd,ze,zf,tolmax,relmax
common/ftcud3sp/xa,xb,yc,yd,ze,zf,tolmax,relmax
if (kbdy.eq.1) then
C
C      x=xa surface
C
x = xa
y = xory

```

```

z = yorz
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
cons = (1.,1.)
gbdy = px+cons*pe
return
end if
if (kbdf.eq.3) then
C
C      y=yc surface
C
y = yc
x = xor y
z = yorz
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
cons = (-1.,-1.)
gbdy = py+cons*pe
return
end if
if (kbdf.eq.5) then
C
C      z=ze surface
C
z = ze
x = xor y
y = yorz
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
cons = (1.,-1.)
gbdy = pz + cons*pe
return
end if
end

subroutine exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
C
C      this subroutine is used to set an exact solution
for testing cud3sp
C      (i.e., setting the rhs, boundary conditions and
computing the exact
C      error)
C
implicit none
real x,y,z,exy,eyz
complex pe,px,py,pz,pxx,pyy,pzz
exy = exp(x+y)

```

```

eyz = exp(y+z)
pe = cmplx(exy,eyz)
px = cmplx(exy,0.)
pxx = px
py = pe
pyy = pe
pz = cmplx(0.,eyz)
pzz = pz
return
end

```

TCUH2

```
C      file tcuh2.f
C
C      * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *       University Corporation for Atmospheric
Research           *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK version 5.0.1
*
C      *
*
C      *               A Fortran Package of Multigrid
```



```
c      test program for the hybrid complex MUDPACK solver
cuh2
c
c ... required MUDPACK files
c
c      cuh2.f, cudcom.f
c
c
c
*****  

*c
c
c      sample program/test driver for cuh2
c
c
*****  

**  

c
*****  

**  

c
c      a sample program/test driver for cuh2 is below.
it can be
c      executed as an initial test.  the output is listed
for the
c      test case described.
c
c      test the driver below by solving the complex
linear elliptic pde
c
c      cmplx(1.+x*x,1.+y*y)*pxx +
c
c      cmplx(exp(-x),exp(-y))*(pyy - py) +
c
c      cmplx(y,x)*p(x,y) = r(x,y)
c
c      on a 46 by 57 grid superimposed on the unit
square.
c      Assume specified boundary conditions at xb=1.0,
c      yc = 1.0 and mixed boundary conditions
c
```

```

C           dp/dx - cmplx(y,y)*p(xa,y) = g(y) at x = xa
C           and
C
C           dp/dy + cmplx(x,x)*p(x,yd) = h(x) at y = yd.
C
C           the exact solution
C
C           p(x,y) = cmplx(x**5,y**5) + 1.0
C
C           is used for testing. one full multigrid cycle (no
initial guess)
C           with red/black gauss-seidel point relaxation and
the default multigrid
C           options is sufficient to reach discretization
level error.
C
C           choosing the grid size arguments
C
C           ixp = iparm(6) = 11, jpy = iparm(7) = 7
C
C           iex = iparm(8) = 3,   jey = iparm(9) = 4
C
C           fits the 46 X 57 grid exactly. This choice
results in the
C           grid coarsening:
C
C           45 X 57 > 23 X 29 > 12 X 15 > 12 X 8
C
C           The coarsest 12 X 8 grid has too many points for
effective error
C           reducting with relaxation only. cuh2 uses a
direct method whenever
C           the 12 X 8 grid is encountered which maintains
multigrid convergence
C           efficiency.
C
C           ****
C           output (32 bit floating point arithmetic)
C
C           ****
C
C           cuh2 test
C
C           integer input arguments

```

```

c      intl = 0 nxa = 2 nxb = 1 nyc = 1 nyd = 2
c      ixp = 11 jjyq = 7 iex = 3 jjey = 4
c      nx = 45 ny = 57 iguess = 0 maxcy = 1
c      method = 0 work space estimate = 34797
c
c      multigrid option arguments
c      kcycle = 2
c      iprer = 2
c      ipost = 1
c      interpol = 3
c
c      floating point input parameters
c      xa = 0.000 xb = 1.000 yc = 0.000 yd = 1.000
c      tolerance (error control) = 0.000E+00
c
c      discretization call to cuh2 intl = 0
c      ierror = 0 minimum work space = 30053
c
c      approximation call to cuh2
c      intl = 1 method = 0 iguess = 0
c      ierror = 0
c      maximum error = 0.803E-03
c
c
c ****
c ****
c      end of output
c
c ****
c ****
c
c      program tcuh2
c      implicit none
c      integer
c      iixp,jjyq,iiex,jjey,nnx,nny,llwrk,llwork,lldir
c      integer mmx,mmy
c
c      set grid sizes with parameter statements
c
c      parameter (iixp = 11, jjyq = 7 , iiex =3, jjey =
4)
c      parameter (mmx = iixp+1, mmy=jjyq+1)
c      parameter (nnx=iixp*2** (iiex-1)+1,
c      nny=jjyq*2** (jjey-1)+1)

```

```

C
C      set work space estimate (see cuh2.d)
C
C      parameter (llwrk=(40*nnx*nny+8* (nnx+nny+2)) /3)
C      parameter (lldir=(2* (iixp+1)* (2*jyq-1)+jyq+1) )
C      parameter (llwork = llwrk+lldir)
C
C      dimension solution,right hand side, and work
C      arrays
C
C      complex p(nnx,nny),r(nnx,nny),w(llwork)
C      integer iw(mmx,mmy)
C      put integer and floating point parameter names in
C      contiguous
C      storeage for labelling purposes
C
C      integer iprm(16),mgopt(4)
C      real fprm(6)
C      integer
C      intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny,
C      +
C      iguess,maxcy,method,nwork,lwrkqd,itero
C
C      common/itcud2/intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny
C      ,
C      +
C      iguess,maxcy,method,nwork,lwrkqd,itero
C      real xa,xb,yc,yd,tolmax,relmax
C      common/ftcud2/xa,xb,yc,yd,tolmax,relmax
C      equivalence(intl,iprm)
C      equivalence(xa,fprm)
C      integer i,j,ierror
C      complex pe,px,py,pxx,pyy,cxx,cyy,cx,cy,ce
C      real dlx,dly,x,y,errmax
C
C      declare coefficient and boundary condition input
C      subroutines external
C
C      external cof,bndc
C
C
C      set input integer arguments
C
C      intl = 0

```

```
C
C      set boundary condition flags
C
C      nxa = 2
C      nxb = 1
C      nyc = 1
C      nyd = 2
C
C      set grid sizes from parameter statements
C
C      ixp = iixp
C      jyq = jjyq
C      iex = iiex
C      jey = jjey
C      nx = nnx
C      ny = nny
C
C      set for one multigrid cycle
C
C      maxcy = 1
C
C      set work space length approximation from parameter
C      statement
C
C      nwork = llwork
C
C      set point relaxation
C
C      method = 0
C
C      flag no initial guess (this sets full multigrid
C      cycling)
C
C      iguess = 0
C
C      set end points of solution rectangle in (x,y)
C      space
C
C      xa = 0.0
C      xb = 1.0
C      yc = 0.0
C      yd = 1.0
C
C      set mesh increments
```

```

c
      dlx = (xb-xa)/float(nx-1)
      dly = (yd-yc)/float(ny-1)
c
c      set for no error control
c
      tolmax = 0.0
c
c      set right hand side in r
c      initialize p to zero
c
      do i=1,nx
      x = xa+float(i-1)*dlx
      do j=1,ny
          y = yc+float(j-1)*dly
          call cof(x,y,cxx,cyy,cx,cy,ce)
          call exact(x,y,pxx,pyy,px,py,pe)
          r(i,j) = cxx*pxx+cyy*pyy+cx*px+cy*py+ce*pe
          p(i,j) = (0.0,0.0)
      end do
      end do
c
c      set specified boundaries in p
c
      x = xb
      do j=1,ny
      y = yc+float(j-1)*dly
      call exact(x,y,pxx,pyy,px,py,pe)
      p(nx,j) = pe
      end do
      y = yc
      do i=1,nx
      x = xa+float(i-1)*dlx
      call exact(x,y,pxx,pyy,px,py,pe)
      p(i,1) = pe
      end do
c
c      set default multigrid options
c
      mgopt(1) = 2
      mgopt(2) = 2
      mgopt(3) = 1
      mgopt(4) = 3
c

```

```

c      print input parameters (except multigrid options
which are default)
c
      write(6,100)
100 format(//' cuh2 test ')
      write (*,101) (iprm(i),i=1,15)
101 format(/' integer input arguments ',
      +' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =
',i2,' nyd = ',i2,
      +' ixp = ',i2,' jyq = ',i2,' iex = ',i2,' jey =
',i2
      +' nx = ',i3,' ny = ',i3,' iguess = ',i2,' maxcy =
',i2,
      +' method = ',i2, ' work space estimate = ',i7)
      write (*,102) (mgopt(i),i=1,4)
102 format(/' multigrid option arguments ',
      +' kcycle = ',i2,
      +' iprer = ',i2,
      +' ipost = ',i2
      +' interpol = ',i2)
      write(*,103) xa,xb,yc,yd,tolmax
103 format(/' floating point input parameters ',
      +' xa = ',f6.3,' xb = ',f6.3,' yc = ',f6.3,' yd =
',f6.3,
      +' tolerance (error control) =    ',e10.3)
c
c      initialization call
c
      write(*,104) intl
104 format(/' discretization call to cuh2', ' intl =
', i2)
      call
      cuh2(iprm,fprm,w,iw,cof,bndc,r,p,mgopt,ierror)
c
c      print error parameter and minimum work space
requirement
c
      write (*,105) ierror,iprm(16)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)
c
c      attempt solution
c

```

```

        intl = 1
        write(*,106) intl,method,iguess
106 format(' approximation call to cuh2',
          +' intl = ',i2, ' method = ',i2,' iguess = ',i2)
          call
cuh2(iprm,fprm,w,iw,cof,bndc,r,p,mgopt,ierror)
        write (*,107) ierror
107 format(' ierror = ',i2)

c
c      compute and print exact maximum error
c
        errmax = 0.0
        do j=1,ny
y = yc+(j-1)*dly
        do i=1,nx
            x = xa+(i-1)*dlx
            call exact(x,y,pxx,pyy,px,py,pe)
            errmax = amax1(errmax,cabs((p(i,j)-pe)))
        end do
        end do
        write(*,108) errmax
108 format(' maximum error = ',e10.3)
        end

        subroutine cof(x,y,cxx,cyy,cx,cy,ce)
c
c      input pde coefficients at any grid point (x,y) in
the solution region
c      (xa.le.x.le.xb,yc.le.y.le.yd) to cud2
c
        implicit none
        real x,y
        complex cxx,cyy,cx,cy,ce
        cxx = cmplx(1.+x*x,1.+y*y)
        cyy = cmplx(exp(-x),exp(-y))
        cx = (0.,0.)
        cy = -cyy
        ce = -cmplx(y,x)
        return
        end

        subroutine bndc(kbdy,xory,alfa,gbdy)
c
c      input mixed derivative b.c. to cud2

```

```

c
implicit none
integer kbdy
real xory,x,y
complex alfa,gbdy
real xa,xb,yc,yd,tolmax,relmax
common/ftcud2/xa,xb,yc,yd,tolmax,relmax
complex pe,px,py,pxx,pyy
if (kbdy.eq.1) then
c
c      x=xa boundary (nxa must equal 2)
c      b.c. has the form px + alfxa(y)*pe = gbdxa(y)
c      alfa and gbdy corresponding to alfxa(y),gbdxa(y)
c      must be output
c
y = xory
x = xa
call exact(x,y,pxx,pyy,px,py,pe)
alfa = -cmplx(y,y)
gbdy = px + alfa*pe
return
end if
if (kbdy.eq.4) then
c
c      y = yd boundary (nyd must equal 2)
c      b.c. has the form py + alfyd(x)*pe = gbdyd(x)
c      alfa and gbdy corresponding to alfyd(x),gbdyd(x)
c      must be output
c
y = yd
x = xory
call exact(x,y,pxx,pyy,px,py,pe)
alfa = cmplx(x,x)
gbdy = py + alfa*pe
return
end if
end

subroutine exact(x,y,pxx,pyy,px,py,pe)
c
c      this subroutine is used to set an exact solution
for testing cud2
c
implicit none

```

```
real x,y
complex pxx,pyy,px,py,pe
pe = cmplx(x**5,y**5)+1.0
px = cmplx(5*x**4,0.)
py = cmplx(0.,5*y**4)
pxx = cmplx(20.*x**3,0.)
pyy = cmplx(0.,20.*y**3)
return
end
```

---

## TCUH24

```
C
C      file tcuh24.f
C
C      * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *               University Corporation for Atmospheric
Research           *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK version 5.0.1
*
C      *
*
C      *               A Fortran Package of Multigrid
*
C      *
```

```
*  
C      *          Subroutines and Example Programs  
*  
C      *  
*  
C      *      for Solving Elliptic Partial Differential  
Equations      *  
C      *  
*  
C      *          by  
*  
C      *  
*  
C      *          John Adams  
*  
C      *  
*  
C      *          of  
*  
C      *  
*  
C      *          the National Center for Atmospheric  
Research      *  
C      *  
*  
C      *          Boulder, Colorado (80307)  
U.S.A.      *  
C      *  
*  
C      *          which is sponsored by  
*  
C      *  
*  
C      *          the National Science Foundation  
*  
C      *  
*  
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C  
C ... purpose  
C  
C      test program for the hybrid complex MUDPACK solver  
cuh24
```

```

C
C ... required MUDPACK files
C
C      cuh24.f, cuh2.f, cudcom.f
C
C
C
***** ****
*
C
***** ****
*
C
C      sample program/test driver for cuh24
C
C
C
***** ****
**
C
***** ****
**
C
C      a sample program/test driver for cuh24 is below.
it can be
C      executed as an initial test.  the output is listed
for the
C      test case described.
C
C      test the driver below by solving the complex
linear elliptic pde
C
C      cmplx(1.+x*x,1.+y*y)*pxx +
C
C      cmplx(exp(-x),exp(-y))* (pyy - py) +
C
C      cmplx(y,x)*p(x,y) = r(x,y)
C
C      on a 46 by 57 grid superimposed on the unit
square.
C      Assume specified boundary conditions at xb=1.0,
C      yc = 1.0 and mixed boundary conditions
C
C      dp/dx - cmplx(y,y)*p(xa,y) = g(y) at x = xa
C      and

```

```

C
C           dp/dy + cmplx(x,x)*p(x,yd) = h(x) at y = yd.
C
C           the exact solution
C
C           p(x,y) = cmplx(x**5,y**5) + 1.0
C
C           is used for testing. Three multigrid cycles (no
initial guess)
C           with red/black gauss-seidel point relaxation and
the default multigrid
C           options are executed using cuh2 to reach
discretization level error.
C           Then cuh24 is called to improve the second-order
estimate to
C           fourth-order

C           choosing the grid size arguments
C
C           ixp = iparm(6) = 11, jpy = iparm(7) = 7
C
C           iex = iparm(8) = 3,   jey = iparm(9) = 4
C
C           fits the 46 X 57 grid exactly. This choice
results in the
C           grid coarsening:
C
C           45 X 57 > 23 X 29 > 12 X 15 > 12 X 8
C
C           The coarsest 12 X 8 grid has too many points for
effective error
C           reducting with relaxation only. cuh24 uses a
direct method whenever
C           the 12 X 8 grid is encountered which maintains
multigrid convergence
C           efficiency.

C ****
C           output (64 bit floating point arithmetic)
C
*****
C
C           cuh2 test
C
```

```

c      integer input arguments
c      intl = 0 nxa = 2 nxb = 1 nyc = 1 nyd = 2
c      ixp = 11 jyq = 7 iex = 3 jey = 4
c      nx = 45 ny = 57 iguess = 0 maxcy = 3
c      method = 0 work space estimate = 30053
c
c      multigrid option arguments
c      kcycle = 2
c      iprer = 2
c      ipost = 1
c      interpol = 3
c
c      floating point input parameters
c      xa = 0.000 xb = 1.000 yc = 0.000 yd = 1.000
c      tolerance (error control) = 0.000E+00
c
c      discretization call to cuh2 intl = 0
c      ierror = 0 minimum work space = 30053
c
c      approximation call to cuh2
c      intl = 1 method = 0 iguess = 0 maxcy = 3
c      ierror = 0
c      maximum error = 0.825E-03
c
c      cuh24 test ierror = 0
c      maximum error = 0.501E-05
c
c
c ****
c ****
c      end of output
c
c ****
c ****
c
c      program tcuh24
c      implicit none
c      integer iixp,jjyq,iiex,jjey,nnx,nny,llwork
c      integer mmx,mmy
c
c      set grid sizes with parameter statements
c
c      parameter (iixp = 11, jjyq = 7 , iiex =3, jjey =
4)

```

```

      parameter (mmx = iixp+1, mmy=jjyq+1)
      parameter (nnx=iixp*2** (iex-1)+1,
nny=jjyq*2** (jey-1)+1)
C
C      set exact minimal work space required (see
tcuh2.f)
C
      parameter (llwork = 30053)
C
C      dimension solution,right hand side, and work
arrays
C
      complex p(nnx,nny),r(nnx,nny),w(llwork)
      integer iw(mmx,mmy)
C
C      put integer and floating point parameter names in
contiguous
C      storeage for labelling purposes
C
      integer iprm(16),mgopt(4)
      real fprm(6)
      integer
intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero

common/itcud2/intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny
,
+
iguess,maxcy,method,nwork,lwrkqd,itero
      real xa,xb,yc,yd,tolmax,relmax
      common/ftcud2/xa,xb,yc,yd,tolmax,relmax
      equivalence(intl,iprm)
      equivalence(xa,fprm)
      integer i,j,ierror
      complex pe,px,py,pxx,pyy,cxx,cyy,cx,cy,ce
      real dlx,dly,x,y,errmax
C
C      declare coefficient and boundary condition input
subroutines external
C
      external cof,bndc
C

```

```
c      set input integer arguments
c
c      intl = 0
c
c      set boundary condition flags
c
c      nxa = 2
c      nxb = 1
c      nyc = 1
c      nyd = 2
c
c      set grid sizes from parameter statements
c
c      ixp = iixp
c      jyq = jjyq
c      iex = iiex
c      jey = jjey
c      nx = nnx
c      ny = nny
c
c      set three multigrid cycles
c
c      maxcy = 3
c
c      set work space length approximation from parameter
c      statement
c
c      nwork = llwork
c
c      set point relaxation
c
c      method = 0
c
c      flag no initial guess (this sets full multigrid
c      cycling)
c
c      iguess = 0
c
c      set end points of solution rectangle in (x,y)
c      space
c
c      xa = 0.0
c      xb = 1.0
c      yc = 0.0
```

```

yd = 1.0
c
c      set mesh increments
c
dix = (xb-xa)/float(nx-1)
dly = (yd-yc)/float(ny-1)
c
c      set for no error control
c
tolmax = 0.0
c
c      set right hand side in r
c      initialize p to zero
c
do i=1,nx
x = xa+float(i-1)*dix
do j=1,ny
y = yc+float(j-1)*dly
call cof(x,y,cxx,cyy,cx,cy,ce)
call exact(x,y,pxx,pyy,px,py,pe)
r(i,j) = cxx*pxx+cyy*pyy+cx*px+cy*py+ce*pe
p(i,j) = (0.0,0.0)
end do
end do
c
c      set specified boundaries in p
c
x = xb
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,pxx,pyy,px,py,pe)
p(nx,j) = pe
end do
y = yc
do i=1,nx
x = xa+float(i-1)*dix
call exact(x,y,pxx,pyy,px,py,pe)
p(i,1) = pe
end do
c
c      set default multigrid options
c
mgopt(1) = 2
mgopt(2) = 2

```

```

mgopt(3) = 1
mgopt(4) = 3
C
C      print input parameters (except multigrid options
which are default)
C
      write(6,100)
100 format(//' cuh2 test ')
      write (*,101) (iprm(i),i=1,15)
101 format(/' integer input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =
',i2,' nyd = ',i2,
+/' ixp = ',i2,' jyq = ',i2,' iex = ',i2,' jey =
',i2
+/' nx = ',i3,' ny = ',i3,' iguess = ',i2,' maxcy =
'i2,
+/' method = ',i2, ' work space estimate = ',i7)
      write (*,102) (mgopt(i),i=1,4)
102 format(/' multigrid option arguments ',
+/' kcycle = ',i2,
+/' iprer = ',i2,
+/' ipost = ',i2
+/' interpol = ',i2)
      write(*,103) xa,xb,yc,yd,tolmax
103 format(/' floating point input parameters ',
+/' xa = ',f6.3,' xb = ',f6.3,' yc = ',f6.3,' yd =
',f6.3,
+/' tolerance (error control) =     ',e10.3)
C
C      initialization call
C
      write(*,104) intl
104 format(/' discretization call to cuh2', ' intl =
', i2)
      call
cuh2(iprm,fprm,w,iw,cof,bndc,r,p,mgopt,ierror)
C
C      print error parameter and minimum work space
requirement
C
      write (*,105) ierror,iprm(16)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)

```

```

C
C      attempt solution
C
C      intl = 1
      write(*,106) intl,method,iguess,maxcy
106 format(/' approximation call to cuh2',
      +' intl = ',i2, ' method = ',i2,' iguess = ',i2,
maxcy = ',i2)
      call
      cuh2(iprm,fprm,w,iw,cof,bndc,r,p,mgopt,ierror)
      write (*,107) ierror
107 format(' ierror = ',i2)
C
C      compute and print exact maximum error
C
      errmax = 0.0
      do j=1,ny
      y = yc+(j-1)*dly
      do i=1,nx
      x = xa+(i-1)*dlx
      call exact(x,y,pxx,pyy,px,py,pe)
      errmax = amax1(errmax,cabs((p(i,j)-pe)))
      end do
      end do
      write(*,108) errmax
108 format(' maximum error = ',e10.3)
C
C      attempt fourth-order estimate
C
      call cuh24(w,iw,p,ierror)
      write(*,109) ierror
109 format(/' cuh24 test', ' ierror = ',i2)
      if (ierror.gt.0) call exit(0)
C
C      compute and print exact maximum error
C
      errmax = 0.0
      do j=1,ny
      y = yc+(j-1)*dly
      do i=1,nx
      x = xa+(i-1)*dlx
      call exact(x,y,pxx,pyy,px,py,pe)
      errmax = amax1(errmax,cabs((p(i,j)-pe)))
      end do

```

```

    end do
    write(*,108) errmax
    end

    subroutine cof(x,y,cxx,cyy,cx,cy,ce)
c
c      input pde coefficients at any grid point (x,y) in
the solution region
c      (xa.le.x.le.xb,yc.le.y.le.yd) to cud2
c
        implicit none
        real x,y
        complex cxx,cyy,cx,cy,ce
        cxx = cmplx(1.+x*x,1.+y*y)
        cyy = cmplx(exp(-x),exp(-y))
        cx = (0.,0.)
        cy = -cyy
        ce = -cmplx(y,x)
        return
        end

    subroutine bndc(kbdy,xory,alfa,gbdy)
c
c      input mixed derivative b.c. to cud2
c
        implicit none
        integer kbdy
        real xory,x,y
        complex alfa,gbdy
        real xa,xb,yc,yd,tolmax,relmax
        common/ftcud2/xa,xb,yc,yd,tolmax,relmax
        complex pe,px,py,pxx,pyy
        if (kbdy.eq.1) then
c
c      x=xa boundary (nxa must equal 2)
c      b.c. has the form px + alfxa(y)*pe = gbdxa(y)
c      alfa and gbdy corresponding to alfxa(y),gbdxa(y)
c      must be output
c
        y = xory
        x = xa
        call exact(x,y,pxx,pyy,px,py,pe)
        alfa = -cmplx(y,y)
        gbdy = px + alfa*pe

```

```

    return
    end if
    if (kbdy.eq.4) then
C
C      y = yd boundary (nyd must equal 2)
C      b.c. has the form py + alfyd(x)*pe = gbdy(x)
C      alfa and gbdy corresponding to alfyd(x),gbdy(x)
C      must be output
C
      y = yd
      x = xory
      call exact(x,y,pxx,pyy,px,py,pe)
      alfa = cmplx(x,x)
      gbdy = py + alfa*pe
      return
      end if
      end

      subroutine exact(x,y,pxx,pyy,px,py,pe)
C
C      this subroutine is used to set an exact solution
for testing cud2
C
      implicit none
      real x,y
      complex pxx,pyy,px,py,pe
      pe = cmplx(x**5,y**5)+1.0
      px = cmplx(5*x**4,0.)
      py = cmplx(0.,5*y**4)
      pxx = cmplx(20.*x**3,0.)
      pyy = cmplx(0.,20.*y**3)
      return
      end

```

---

## TCUH24CR

```

C
C      file tcuh24cr.f

```

```
C
C      * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *           copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *           all rights reserved
*
C      *
*
C      *           MUDPACK version 5.0.1
*
C      *
*
C      *           A Fortran Package of Multigrid
*
C      *
*
C      *           Subroutines and Example Programs
*
C      *
*
C      *           for Solving Elliptic Partial Differential
Equations        *
C      *
*
C      *           by
*
C      *
*
C      *           John Adams
*
C      *
*
C      *           of
*
C      *
```

```
*  
C      *          the National Center for Atmospheric  
Research           *  
C      *  
*  
C      *          Boulder, Colorado (80307)  
U.S.A.           *  
C      *  
*  
C      *          which is sponsored by  
*  
C      *  
*  
C      *          the National Science Foundation  
*  
C      *  
*  
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C  
C ... purpose  
C  
C      test program for the complex MUDPACK solver  
cuh24cr  
C  
C ... required MUDPACK files  
C  
C      cuh24cr.f, cuh2cr.f, cudcom.f  
C  
C  
*****  
*  
C  
*****  
*  
C  
C      sample program/test driver for cuh24cr  
C  
C  
*****  
**  
C  
*****  
**
```

```

c
c      a sample program/test driver for cuh24cr is below.
it can
c      be executed as an initial test.  the output is
listed for
c      the test case described.
c
c      test the driver below by solving the elliptic pde
c
c      cmplx(1.+y*y,1.-y*y)*d(dp/dx)/dx + cmplx(x*y,-
x*y)*d(dp/dx)/dy
c
c      cmplx(1.+x*x,1.-x*x)*d(dp/dy)/dy + cmplx(y,-
y)*dp/dx +
c
c      cmplx(x,-x)*dp/dy + cmplx(x+y,-x-y)*p(x,y) =
r(x,y)
c
c      on the region
c
c      0.25 < x < 1.0, 0.0 < y < 0.50
c
c      with specified boundary conditions at xa=0.25,
xb=1.0, yc=0.0
c      and mixed boundary conditions at yd=0.5 of the
form:
c
c      cmplx(-x,x)*dp/dx+cmplx(1.+x,1.-x)*dp/dy+cmplx(-
x,x)*p(x,yd)=gbdyd(x)
c
c      choose a 45 x 73 grid and use line relaxation in
the y direction.
c      forr testing purposes use the exact solution
c
c      p(x,y) = cmplx((x*y)**3,-(x*y)**3) + (1.,1.)
c
c      Choosing grid arguments
c
c      ixp = 11, iex = 3, jyq = 9, jey = 4
c
c      fits the required 45 X 73 grid exactly with the
coarsening
c
c      45 X 73 > 23 X 37 > 12 X 19 > 12 X 10

```

```

c
c      The 12 X 10 coarsest grid has too many points for
effective
c      error reduction with relaxation alone. cuh2cr
maintains
c      multigrid convergence efficiency by utilitzing a
direct method
c      (Gaussian elimination) whenever the coarsest grid
is encountered
c      within multigrid cycling. Two cycles with cuh2cr
are executed
c      and then cuh24cr is called for a fourth-order
estimate
c
c
c ****
c      output (64 bit floating point arithmetic)
c
c ****
c
c      cuh2cr test
c
c      integer input arguments
c      intl = 0 nxa = 1 nxb = 1 nyc = 1 nyd = 2
c      ixp = 11 jyq = 9 iex = 3 jey = 4
c      nx = 45 ny = 73 iguess = 0 maxcy = 2
c      method = 2 work space estimate = 69661
c
c      multigrid option arguments
c      kcycle = 2
c      iprer = 2
c      ipost = 1
c      interpol = 3
c
c      floating point input parameters
c      xa = 0.250 xb = 1.000 yc = 0.000 yd = 0.500
c      tolerance (error control) = 0.000E+00
c
c      discretization call to cuh2cr intl = 0
c      ierror = 0 minimum work space = 69661
c
c      approximation call to cuh2cr
c      intl = 1 method = 2 iguess = 0 maxcy = 2
c      ierror = 0

```

```

C      maximum error = 0.114E-04
C
C      cuh24cr test ierror = 0
C      maximum error = 0.739E-07
C
C
C*****
**          end of output
C
*****          ****
**          ****
C
      program tcuh24cr
      implicit none
      integer iixp,jjyq,iiex,jjey,nnx,nnv,llw,mmx,mmy
C
C      set grid sizes with parameter statements
C
      parameter (iixp =11 , jjyq = 9 , iiex =3, jjey =
4)
      parameter (mmx = iixp+1,mmy = jjyq+1)
      parameter (nnx=iixp*2** (iiex-1)+1,
nnv=jjyq*2** (jjey-1)+1)
C
C      set exact minimal required work space (see
tcuh2cr.f)
C
      parameter (llw = 69661)
C
C      dimension solution,right hand side, and work
arrays
C
      complex p(nnx,nnv),r(nnx,nnv),w(llw)
      integer iw(mmx,mmy)
C
C      put integer and floating point parameter names in
contiguous
C      storeage for labelling purposes
C
      integer iprm(16),mgopt(4)
      real fprm(6)
      integer
      intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny,

```

```

+
iguess,maxcy,method,nwork,lwrkqd,itero

common/itcuh2cr/intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,
ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero
    real xa,xb,yc,yd,tolmax,relmax
    common/ftcur2/xa,xb,yc,yd,tolmax,relmax
    equivalence(intl,iprm)
    equivalence(xa,fprm)
    integer i,j,ierror
    complex cxx,cxy,cyy,cx,cy,ce,pxx,pxy,pyy,px,py,pe
    real dlx,dly,x,y,errmax

C
C      declare coefficient and boundary condition input
subroutines external
C
        external cofcr,bndcr
C
C      set input integer arguments
C
        intl = 0
C
C      set boundary condition flags
C
        nxa = 1
        nxb = 1
        nyc = 1
        nyd = 2
C
C      set grid sizes from parameter statements
C
        ixp = iixp
        jyq = jjyq
        iex = iiex
        jey = jjey
        nx = nnx
        ny = nny
C
C      set two multigrid cycles
C
        maxcy = 2
C

```

```

c      set work space length approximation from parameter
statement
c
nwork = llw
c
c      set point relaxation
c
method = 2
c
c      flag no initial guess (this sets full multigrid
cycling)
c
iguess = 0
c
c      set end points of solution rectangle in (x,y)
space
c
xa = 0.25
xb = 1.0
yc = 0.0
yd = 0.50
c
c      set mesh increments
c
dlx = (xb-xa)/float(nx-1)
dly = (yd-yc)/float(ny-1)
c
c      set for no error control
c
tolmax = 0.0
c
c      set right hand side in r
c      initialize p to zero
c
do i=1,nx
x = xa+float(i-1)*dlx
do j=1,ny
y = yc+float(j-1)*dly
call cofcr(x,y,cxx,cxy,cyy,cx,cy,ce)
call exact(x,y,pxx,pxy,pyy,px,py,pe)
r(i,j) = cxx*pss+cxy*pss+cyy*pss+cx*pss+cy*pss+ce*pss
p(i,j) = (0.0,0.0)
end do
end do

```

```

c
c      set specified boundaries in p
c
x = xa
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,pxx,pxy,pyy,px,py,pe)
p(1,j) = pe
end do
x = xb
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,pxx,pxy,pyy,px,py,pe)
p(nx,j) = pe
end do
y = yc
do i=1,nx
x = xa+float(i-1)*dlx
call exact(x,y,pxx,pxy,pyy,px,py,pe)
p(i,1) = pe
end do
c
c      set default multigrid opitons
c
mgopt(1) = 2
mgopt(2) = 2
mgopt(3) = 1
mgopt(4) = 3
c
c      print input arguments
c
write(6,100)
100 format(//' cuh2cr test ')
      write (*,101) (iprm(i),i=1,15)
101 format(/' integer input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxn = ',i2,' nyc =
',i2,' nyd = ',i2,
+/' ixp = ',i2,' jyq = ',i2,' iex = ',i2,' jey =
',i2,
+/' nx = ',i3,' ny = ',i3,' iguess = ',i2,' maxcy =
',i2,
+/' method = ',i2, ' work space estimate = ',i7)
      write (*,102) (mgopt(i),i=1,4)
102 format(/' multigrid option arguments ',
```

```

+/' kcycle = ',i2,
+/' iprer = ',i2,
+/' ipost = ',i2
+/' interpol = ',i2)
    write(*,103) xa,xb,yc,yd,tolmax
103 format(/' floating point input parameters ',
+/' xa = ',f6.3,' xb = ',f6.3,' yc = ',f6.3,' yd =
',f6.3,
+/' tolerance (error control) =      ',e10.3)
C
C      initialization call
C
        write(*,104) intl
104 format(/' discretization call to cuh2cr', ' intl =
', i2)
        call
cuh2cr(iprm,fprm,w,iw,cofcr,bndcr,r,p,mgopt,ierror)
C
C      print error parameter and minimum work space
requirement
C
        write (*,105) ierror,iprm(16)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
        if (ierror.gt.0) call exit(0)
C
C      attempt solution
C
        intl = 1
        write(*,106) intl,method,iguess,maxcy
106 format(/' approximation call to cuh2cr',
+/' intl = ',i2, ' method = ',i2,' iguess = ',i2, '
maxcy = ',i2)
        call
cuh2cr(iprm,fprm,w,iw,cofcr,bndcr,r,p,mgopt,ierror)
        write (*,107) ierror
107 format(' ierror = ',i2)
C
C      compute and print exact maximum error
C
        errmax = 0.0
        do j=1,ny
y = yc+(j-1)*dly
        do i=1,nx

```

```

x = xa+(i-1)*dlx
call exact(x,y,pxx,pxy,pyy,px,py,pe)
errmax = amax1(errmax,cabs((p(i,j)-pe)))
end do
    end do
    write(*,108) errmax
108 format(' maximum error = ',e10.3)
c
c      attempt fourth-order estimate with difference
corrections
c
call cuh24cr(w,iw,cofcr,bndcr,p,ierror)
write (*,109) ierror
109 format(/ ' cuh24cr test ', ' ierror = ',i2)
if (ierror.gt.0) call exit(0)
errmax = 0.0
do j=1,ny
y = yc+(j-1)*dly
do i=1,nx
    x = xa+(i-1)*dlx
    call exact(x,y,pxx,pxy,pyy,px,py,pe)
    errmax = amax1(errmax,cabs((p(i,j)-pe)))
end do
    end do
    write(*,108) errmax
end

subroutine cofcr(x,y,cxx,cxy,cyy,cx,cy,ce)
c
c      input pde coefficients at any point (x,y) in the
solution region
c      (xa.le.x.le.xb,yc.le.y.le.yd) to cuh2cr
c
implicit none
complex cxx,cxy,cyy,cx,cy,ce
real x,y,xa,xb,yc,yd,tolmax,relmax
common/ftcur2/xa,xb,yc,yd,tolmax,relmax
cxx = cmplx(1.+y**2,1.-y**2)
cxy = cmplx(x*y,-x*y)
cyy = cmplx(1.+x**2,1.-x*x)
cx = cmplx(y,-y)
cy = cmplx(x,-x)
ce = -cmplx((x+y),-x-y)
return

```

```

    end

    subroutine bndcr (kbdy,xory,alfa,beta,gama,gbdy)
C
C      input mixed "oblique" derivative b.c. to cuh2cr
C
      implicit none
      integer kbdy
      real xory,x,y,xa,xb,yc,yd,tolmax,relmax
      complex alfa,beta,gama,gbdy,pe,px,py,pxx,pxy,pyy
      common/ftcur2/xa,xb,yc,yd,tolmax,relmax
      if (kbdy.eq.4) then
C
C      y=yd boundary (nyd must equal 2 if this code is to
be executed).
C      b.c. has the form
      alfyd(x)*px+betyd(x)*py+gamyd(x)*pe = gbdyd(x)
C      where x = yorx.  alfa,beta,gama,gbdy
corresponding to alfyd(x),
C      betyd(x),gamyd(x),gbdyd(y) must be output.
C
      y = yd
      x = xory
      alfa = -cmplx(x,-x)
      beta = cmplx(1.+x,1.-x)
      gama = -cmplx(x,-x)
      call exact(x,y,pxx,pxy,pyy,px,py,pe)
      gbdy = alfa*px + beta*py + gama*pe
      return
      end if
      end

      subroutine exact(x,y,pxx,pxy,pyy,px,py,pe)
C
C      this subroutine is used for setting an exact
solution in order
C      to test subroutine cuh2cr.
C
      implicit none
      real x,y,xy,xy2,xy3
      complex pxx,pxy,pyy,px,py,pe
      xy = x*y
      xy2 = xy*xy
      xy3 = xy2*xy

```

```
pe = cmplx(xy3,-xy3)
px = 3.*y*cmplx(xy2,-xy2)
py = 3.*x*cmplx(xy2,-xy2)
pxx = 6.*y*y*cmplx(xy,-xy)
pxy = 9.*xy*cmplx(xy,-xy)
pyy = 6.*x*x*cmplx(xy,-xy)
return
end
```

---

## TCUH2CR

```
C
C      file tcuh2cr.f
C
C      * * * * * * * * * * * * * * * * * * * * * *
* * * * * * *
C      *
*
C      *                               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *                               all rights reserved
*
C      *
*
C      *                               MUDPACK version 5.0.1
*
C      *
*
C      *           A Fortran Package of Multigrid
*
C      *
```

C \* Subroutines and Example Programs  
\*  
C \*  
\*  
C \* for Solving Elliptic Partial Differential  
Equations \*  
C \*  
\*  
C \* by  
\*  
C \*  
\*  
C \* John Adams  
\*  
C \*  
\*  
C \* of  
\*  
C \*  
\*  
C \* the National Center for Atmospheric  
Research \*  
C \*  
\*  
C \* Boulder, Colorado (80307)  
U.S.A. \*  
C \*  
\*  
C \* which is sponsored by  
\*  
C \*  
\*  
C \* the National Science Foundation  
\*  
C \*  
\*  
C \*  
\* \* \* \* \* \* \*  
C  
C ... purpose  
C  
C test program for the complex MUDPACK solver cuh2cr  
C  
C ... required MUDPACK files

```

C
C      cuh2cr.f, cudcom.f
C
C
C*****
C
C*****
C
C***** sample program/test driver for cuh2cr
C
C
C*****
C*****
C
C***** a sample program/test driver for cuh2cr is below.
it can
C      be executed as an initial test.  the output is
listed for
C      the test case described.
C
C      test the driver below by solving the elliptic pde
C
C      cmplx(1.+y*y,1.-y*y)*d(dp/dx)/dx + cmplx(x*y,-
x*y)*d(dp/dx)/dy
C
C      cmplx(1.+x*x,1.-x*x)*d(dp/dy)/dy + cmplx(y,-
y)*dp/dx +
C
C      cmplx(x,-x)*dp/dy + cmplx(x+y,-x-y)*p(x,y) =
r(x,y)
C
C      on the region
C
C      0.25 < x < 1.0, 0.0 < y < 0.50
C
C      with specified boundary conditions at xa=0.25,
xb=1.0, yc=0.0
C      and mixed boundary conditions at yd=0.5 of the
form:

```

```

C
C      cmplx(-x,x)*dp/dx+cmplx(1.+x,1.-x)*dp/dy+cmplx(-
x,x)*p(x,yd)=gbdyd(x)
C
C      choose a 45 x 73 grid and use line relaxation in
the y direction.
C      forr testing purposes use the exact solution
C
C      p(x,y) = cmplx((x*y)**3,-(x*y)**3) + (1.,1.)
C
C      Choosing grid arguments
C
C      ixp = 11, iex = 3, jyq = 9, jey = 4
C
C      fits the required 45 X 73 grid exactly with the
coarsening
C
C      45 X 73 > 23 X 37 > 12 X 19 > 12 X 10
C
C      The 12 X 10 coarsest grid has too many points for
effective
C      error reduction with relaxation alone. cuh2cr
maintains
C      multigrid convergence efficiency by utilitzing a
direct method
C      (Gaussian elimination) whenever the coarsest grid
is encountered
C      within multigrid cycling
C
C ****
C      output (32 bit floating point arithmetic)
C
C ****
C
C      cuh2cr test
C
C      integer input arguments
C      intl = 0 nxa = 1 nxb = 1 nyc = 1 nyd = 2
C      ixp = 11 jyq = 9 iex = 3 jey = 4
C      nx = 45 ny = 73 iguess = 0 maxcy = 1
C      method = 2 work space estimate = 72545
C
C      multigrid option arguments
C      kcycle = 2

```

```

c      iprer = 2
c      ipost = 1
c      interpol = 3
c
c      floating point input parameters
c      xa = 0.250 xb = 1.000 yc = 0.000 yd = 0.500
c      tolerance (error control) = 0.000E+00
c
c      discretization call to cuh2cr intl = 0
c      ierror = 0 minimum work space = 69661
c
c      approximation call to cuh2cr
c      intl = 1 method = 2 iguess = 0 maxcy = 1
c      ierror = 0
c      maximum error = 0.112E-04
c
c
c ****
**      end of output
c
c ****
**      program tcuh2cr
      implicit none
      integer iixp,jjyq,iiex,jjey,nnx,nny,llen,lldir,llw
      integer mmx,mmy
c
c      set grid sizes with parameter statements
c
      parameter (iixp =11 , jjyq = 9 , iiex =3, jjey =
4)
      parameter (mmx = iixp+1,mmy = jjyq+1)
      parameter (nnx=iixp*2**(iiex-1)+1,
nny=jjyq*2**(jjey-1)+1)
c
c      estimate work length approximation for method=0
c      (see cuh2cr.d)
c
      parameter
      (llen=4* ( (nnx+2) * (nny+2)+14*nnx*nny ) /3+ (nnx+2) * (nny+2) )
      parameter (lldir = (iixp+1)*(jjyq+1)*(2*iixp+3))
      parameter (llw = llen + lldir)

```

```

C
c      dimension solution, right hand side, and work
arrays
C
      complex p(nnx,nny),r(nnx,nny),w(llw)
      integer iw(mmx,mmy)
C
c      put integer and floating point parameter names in
contiguous
c      storeage for labelling purposes
C
      integer iprm(16),mgopt(4)
      real fprm(6)
      integer
intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero

common/itcuh2cr/intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,
ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero
      real xa,xb,yc,yd,tolmax,relmax
      common/ftcur2/xa,xb,yc,yd,tolmax,relmax
      equivalence(intl,iprm)
      equivalence(xa,fprm)
      integer i,j,ierror
      complex cxx,cxy,cyy,cx,cy,ce,pxx,pxy,pyy,px,py,pe
      real dlx,dly,x,y,errmax

C
c      declare coefficient and boundary condition input
subroutines external
C
      external cofcr,bndcr
C
c      set input integer arguments
C
      intl = 0
C
c      set boundary condition flags
C
      nxa = 1
      nxb = 1
      nyc = 1

```

```
nyd = 2
c
c      set grid sizes from parameter statements
c
ixp = iixp
jyq = jjyq
iex = iiex
jey = jjey
nx = nnx
ny = nny
c
c      set for one multigrid cycle
c
maxcy = 1
c
c      set work space length approximation from parameter
statement
c
nwork = llw
c
c      set point relaxation
c
method = 2
c
c      flag no initial guess (this sets full multigrid
cycling)
c
iguess = 0
c
c      set end points of solution rectangle in (x,y)
space
c
xa = 0.25
xb = 1.0
yc = 0.0
yd = 0.50
c
c      set mesh increments
c
dlx = (xb-xa)/float(nx-1)
dly = (yd-yc)/float(ny-1)
c
c      set for no error control
c
```

```

tolmax = 0.0
c
c      set right hand side in r
c      initialize p to zero
c
do i=1,nx
x = xa+float(i-1)*dlx
do j=1,ny
y = yc+float(j-1)*dly
call cofcr(x,y,cxx,cxy,cyy,cx,cy,ce)
call exact(x,y,pxx,pxy,pyy,px,py,pe)
r(i,j) = cxx*pxx+cxy*pxy+cyy*pwy+cx*px+cy*py+ce*pe
p(i,j) = (0.0,0.0)
end do
end do
c
c      set specified boundaries in p
c
x = xa
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,pxx,pxy,pwy,px,py,pe)
p(1,j) = pe
end do
x = xb
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,pxx,pxy,pwy,px,py,pe)
p(nx,j) = pe
end do
y = yc
do i=1,nx
x = xa+float(i-1)*dlx
call exact(x,y,pxx,pxy,pwy,px,py,pe)
p(i,1) = pe
end do
c
c      set default multigrid options
c
mgopt(1) = 2
mgopt(2) = 2
mgopt(3) = 1
mgopt(4) = 3
c

```

```

c      print input arguments
c
c      write(6,100)
100 format(//' cuh2cr test ')
      write (*,101) (iprm(i),i=1,15)
101 format(/' integer input arguments ',
      +' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =
',i2,' nyd = ',i2,
      +' ixp = ',i2,' jyq = ',i2,' iex = ',i2,' jey =
',i2
      +' nx = ',i3,' ny = ',i3,' iguess = ',i2,' maxcy =
',i2,
      +' method = ',i2, ' work space estimate = ',i7)
      write (*,102) (mgopt(i),i=1,4)
102 format(/' multigrid option arguments ',
      +' kcycle = ',i2,
      +' iprer = ',i2,
      +' ipost = ',i2
      +' interpol = ',i2)
      write(*,103) xa,xb,yc,yd,tolmax
103 format(/' floating point input parameters ',
      +' xa = ',f6.3,' xb = ',f6.3,' yc = ',f6.3,' yd =
',f6.3,
      +' tolerance (error control) =      ',e10.3)
c
c      initialization call
c
c      write(*,104) intl
104 format(/' discretization call to cuh2cr', ' intl =
', i2)
      call
cuh2cr(iprm,fprm,w,iw,cofcr,bndcr,r,p,mgopt,ierror)
c
c      print error parameter and minimum work space
requirement
c
      write (*,105) ierror,iprm(16)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)
c
c      attempt solution
c
      intl = 1

```

```

        write(*,106) intl,method,iguess,maxcy
106 format(' approximation call to cuh2cr',
          +' intl = ',i2, ' method = ',i2,' iguess = ',i2, '
maxcy = ',i2)
        call
cuh2cr(iprm,fprm,w,iw,cofcr,bndcr,r,p,mgopt,ierror)
        write (*,107) ierror
107 format(' ierror = ',i2)

c
c      compute and print exact maximum error
c
        errmax = 0.0
        do j=1,ny
y = yc+(j-1)*dly
        do i=1,nx
            x = xa+(i-1)*dlx
            call exact(x,y,pxx,pxy,pyy,px,py,pe)
            errmax = amax1(errmax,cabs((p(i,j)-pe)))
        end do
        end do
        write(*,108) errmax
108 format(' maximum error = ',e10.3)
        end

        subroutine cofcr(x,y,cxx,cxy,cyy,cx,cy,ce)
c
c      input pde coefficients at any point (x,y) in the
solution region
c      (xa.le.x.le.xb,yc.le.y.le.yd) to cuh2cr
c
        implicit none
        complex cxx,cxy,cyy,cx,cy,ce
        real x,y,xa,xb,yc,yd,tolmax,relmax
        common/ftcur2/xa,xb,yc,yd,tolmax,relmax
        cxx = cmplx(1.+y**2,1.-y**2)
        cxy = cmplx(x*y,-x*y)
        cyy = cmplx(1.+x**2,1.-x*x)
        cx = cmplx(y,-y)
        cy = cmplx(x,-x)
        ce = -cmplx((x+y),-x-y)
        return
        end

        subroutine bndcr(kbdy,xory,alfa,beta,gama,gbdy)

```

```

c
c      input mixed "oblique" derivative b.c. to cuh2cr
c
c      implicit none
c      integer kbdy
c      real xory,x,y,xa,xb,yc,yd,tolmax,relmax
c      complex alfa,beta,gama,gbdy,pe,px,py,pxx,pxy,pyy
c      common/ftcur2/xa,xb,yc,yd,tolmax,relmax
c      if (kbdy.eq.4) then
c
c      y=yd boundary (nyd must equal 2 if this code is to
c      be executed).
c      b.c. has the form
c      alfyd(x)*px+betyd(x)*py+gamyd(x)*pe = gbdy(x)
c      where x = yorx.    alfa,beta,gama,gbdy
c      corresponding to alfyd(x),
c      betyd(x),gamyd(x),gbdy(y) must be output.
c
c      y = yd
c      x = xory
c      alfa = -cmplx(x,-x)
c      beta = cmplx(1.+x,1.-x)
c      gama = -cmplx(x,-x)
c      call exact(x,y,pxx,pxy,pyy,px,py,pe)
c      gbdy = alfa*px + beta*py + gama*pe
c      return
c      end if
c      end
c
c      subroutine exact(x,y,pxx,pxy,pyy,px,py,pe)
c
c      this subroutine is used for setting an exact
c      solution in order
c      to test subroutine cuh2cr.
c
c      implicit none
c      real x,y,xy,xy2,xy3
c      complex pxx,pxy,pyy,px,py,pe
c      xy = x*y
c      xy2 = xy*xy
c      xy3 = xy2*xy
c      pe = cmplx(xy3,-xy3)
c      px = 3.*y*cmplx(xy2,-xy2)
c      py = 3.*x*cmplx(xy2,-xy2)

```

```
pxx = 6.*y*y*cmplx(xy,-xy)
pxy = 9.*xy*cmplx(xy,-xy)
pyy = 6.*x*x*cmplx(xy,-xy)
return
end
```

---

## TCUH3

```
C
C      file tcuh3.f
C
C      * * * * * * * * * * * * * * * * * * * *
* * * * * * *
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *               University Corporation for Atmospheric
Research           *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK   version 5.0.1
*
C      *
*
C      *               A Fortran Package of Multigrid
*
C      *
*
C      *               Subroutines and Example Programs
*
```

```
C      *
*
C      *      for Solving Elliptic Partial Differential
Equations      *
C      *
*
C      *                      by
*
C      *
*
C      *                      John Adams
*
C      *
*
C      *                      of
*
C      *
*
C      *      the National Center for Atmospheric
Research      *
C      *
*
C      *      Boulder, Colorado (80307)
U.S.A.      *
C      *
*
C      *                      which is sponsored by
*
C      *
*
C      *                      the National Science Foundation
*
C      *
*
C      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *      *
*      *      *      *      *      *      *
C
C ... purpose
C
C      test program for the MUDPACK solver cuh3
C
C ... required MUDPACK files
C
C      cuh3.f, cudcom.f, cud3ln.f, cud3pn.f
```

```

C
C
C
*****
*
C
*****
*
C
C      sample program/test driver for cuh3
C
C
*****
**
C
*****
**
C
C      a sample program/test driver for cuh3 is below. it
can be
c      executed as an initial test. the output is listed
for the
c      test case described.
C
c      test the driver below by solving the complex
nonseparable
c      linear elliptic pde
C
c      cmplx(y,z)*d(dp/dx)/dx +
cmplx(x,z)*d(dp/dy)/dy +
C
c      cmplx(x,y)*d(dp/dz)/dz -
cmplx(y+z,x+y)*pe(x,y,z)
C
c      = r(x,y,z)
C
c      on a 21 X 21 X 57 uniform grid superimposed on the
region
C
c      [0.5,1.0] x [0.5,1.0] x [0.0,1.0]
C
c      Assume specified boundary conditions at all x and
y boundaries

```

```

c      and periodic in the z direction.  Use line
relaxation in the
c      z direction.  The exact solution
c
c      pe(x,y,z) =
exp(x*y)*cmplx(cos(tpi*z),sin(tpi*z))
c
c      is used for testing (tpi = 8.0*atan(1.0)).
Choosing grid
c      size arguments
c
c      ixp = 5, jyq = 5, kzs = 7, iex = 3, jey = 3, kez
= 4
c
c      will give the gridcoarsening
c
c      21 X 21 X 57 > 11 X 11 X 29 > 6 X 6 X 15 > 6 X 6
X 8
c
c      The coarsest 6 by 6 by 8 (x,y,z) grid has too many
points
c      for effective error reduction with relaxation
alone.  cuh3
c      uses a direct method when this grid is encountered
within
c      multigrid cycling thus maintaining multigrid
convergence rates.
c
c ****
c      output (32 bit floating point arithmetic)
c
*****
c
c      cuh3 test
c
c      input arguments
c      intl = 0 nxa = 1 nxb = 1 nyc = 1 nyd = 1
c      nze = 0 nzf = 0
c      ixp = 5 jyq = 5 kzs = 7
c      iex = 3 jey = 3 kez = 4
c      nx = 21 ny = 21 nz = 57 iguess = 0 maxcy = 1
c      method = 3 work space length input = 499376
c      xa = 0.50 xb = 1.00
c      yc = 0.50 yd = 1.00

```

```

C      ze =  0.00 zf =  1.00
C      tolmax =  0.000E+00
C
C      multigrid options
C      kcycle =  2
C      iprer =  2
C      ipost =  1
C      interpol = 3
C
C      discretization call to cuh3 intl =  0
C      ierror =  0 minimum work space =  488841
C
C      approximation call to cuh3
C      intl =  1 method =  3 iguess =  0 maxcy =  1
C      ierror =  0
C      maximum error =  0.112E-02
C
C
C ****
C ****
C      end of output
C
C ****
C ****
C
C      program tcuh3
C      implicit none
C
C      set grid sizes with parameter statements
C
C      integer
iixp,jjyq,kkzr,iiex,jjey,kkez,llwork,nnx,nny,nnz
      integer mmx,mmy,mmz
      parameter(iixp=5,jjyq=5,kkzr=7)
      parameter (mmx=iixp+1,mmy=jjyq+1,mmz=kkzr+1)
      parameter (iiex=3,jjey=3,kkez=4)
      parameter (nnx = iixp*2** (iiex-1)+1)
      parameter (nny = jjyq*2** (jjey-1)+1)
      parameter (nnz = kkzr*2** (kkez-1)+1)
C
C      set work space length estimate for method=3 (see
C      cuh3.d).
C      this will probably overestimate required space
C

```

```

parameter (llwork = 16*(nnx+2)*(nny+2)*(nnz+2) )
C
C
C      dimension solution,right hand side, and work
arrays
C
      complex
phi(nnx,nny,nnz),rhs(nnx,nny,nnz),work(llwork)
      integer iw(mmx,mmy,mmz),iprm(23),mgopt(4)
      real fprm(8)
C
C      put integer and floating point arguments names in
contiguous
C      storeage labelling
C
      integer
intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex,jey,
+
kez,nx,ny,nz,iguess,maxcy,method,meth2,nwork,
+
lwrkqd,itero

common/itcud3/intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,i
ex,jey,
+
kez,nx,ny,nz,iguess,maxcy,method,meth2,nwork,
+
lwrkqd,itero

      real xa,xb,yc,yd,ze,zf,tolmax,relmax,tpi
      common/ftcud3/xa,xb,yc,yd,ze,zf,tolmax,relmax,tpi
      real dlx,dly,dlz,x,y,z,errm
      complex cxx,cyy,czz,cx,cy,cz,ce,
pxx,pyy,pzz,px,py,pz,pe
      integer i,j,k,ierror
      equivalence(intl,iprm)
      equivalence(xa,fprm)

C
C      declare coefficient and boundary condition input
subroutines external
C
      external cof,bndc
      tpi = 8.0*atan(1.0)
C
C      set for initial call
C

```

```
    intl = 0
c
c      set boundary condition flags
c
    nxa = 1
    nxb = 1
    nyc = 1
    nyd = 1
    nze = 0
    nzf = 0
c
c      set grid sizes from parameter statements
c
    ixp = iixp
    jyq = jjyq
    kzs = kkzs
    iex = iiex
    jey = jjey
    kez = kkez
    nx = nnx
    ny = nny
    nz = nnz
c
c      set for one multigrid cycles
c
    maxcy = 1
c
c      set work space length approximation from parameter
statement
c
    nwork = llwork
c
c      set method of relaxation--line in the z direction
c
    method = 3
c
c      meth2 only used in planar relaxation--but set
c
    meth2 = 0
c
c      set full multigrid cycling by flagging no initial
guess at the finest
c      grid level
c
```

```

    iguess = 0
c
c      set end points of solution region in (x,y,z) space
c
        xa = 0.5
        xb = 1.0
        yc = 0.5
        yd = 1.0
        ze = 0.0
        zf = 1.0
c
c      set default multigrid options
c
        mgopt(1) = 2
        mgopt(2) = 2
        mgopt(3) = 1
        mgopt(4) = 3
c
c      set mesh increments
c
        dlx = (xb-xa)/float(nx-1)
        dly = (yd-yc)/float(ny-1)
        dlz = (zf-ze)/float(nz-1)
c
c      set for no error control
c
        tolmax = 0.0
c
c      set right hand side in rhs and phi to zero
c
        do k=1,nz
        z = ze+(k-1)*dlz
        do j=1,ny
        y = yc+float(j-1)*dly
        do i=1,nx
        x = xa+float(i-1)*dlx
        call cof(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
        call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
        rhs(i,j,k) =
        cxx*pxx+cyy*pyy+czz*pzz+cx*px+cy*py+cz*pz+ce*pe
        phi(i,j,k) = (0.,0.)
        end do
        end do
        end do

```

```

c
c      set specified values at x and y boundaries in phi
c
x = xa
do k=1,nz
z = ze+(k-1)*dlz
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(1,j,k) = pe
end do
end do
x = xb
do k=1,nz
z = ze+(k-1)*dlz
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(nx,j,k) = pe
end do
end do
y = yc
do k=1,nz
z = ze+(k-1)*dlz
do i=1,nx
x = xa+float(i-1)*dlx
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(i,1,k) = pe
end do
end do
y = yd
do k=1,nz
z = ze+(k-1)*dlz
do i=1,nx
x = xa+float(i-1)*dlx
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(i,ny,k) = pe
end do
end do
write(6,50)
50 format(//' cuh3 test ')
c
c      print input arguments
c

```

```

write(6,100) intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex
,jey,kez,
+
nx,ny,nz,iguess,maxcy,method,nwork,xa,xb,yc,yd,ze,zf,
+
tolmax,mgopt(1),mgopt(2),mgopt(3),mgopt(4)
100 format(/' input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =
',i2,' nyd = 'i2,
+/' nze = ',i2, ' nzf = ',i2,
+/' ixp = ',i2,' jyq = 'i2,' kzr = 'i2,
+/' iex = ',i2, ' jey = 'i2, ' kez = 'i2,
+/' nx = 'i3,' ny = 'i3,' nz = 'i3, ' iguess =
'i2,' maxcy = 'i2,
+/' method = 'i2, ' work space length input = ',i7,
+/' xa = 'f5.2,' xb = 'f5.2,
+/' yc = 'f5.2,' yd = 'f5.2,
+/' ze = 'f5.2,' zf = 'f5.2,
+/' tolmax = ' ,e10.3
+/// multigrid options '
+/' kcycle = ',i2
+/' iprer = ',i2
+/' ipost = ',i2
+/' interpol = ',i2 )

C
C      discretize pde
C
      write(*,104) intl
104 format(/' discretization call to cuh3', ' intl =
', i2)
      call
cuh3(iprm,fprm,work,iw,cof,bndc,rhs,phi,mgopt,ierror)
      write (*,105) ierror,iprm(22)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)
C
C      approximate pde
C
      intl = 1
      write(*,106) intl,method,iguess,maxcy
106 format(/' approximation call to cuh3 ',
+/' intl = ',i2, ' method = ',i2,' iguess = ',i2, '

```

```

maxcy = ',i2)
      call
cuh3(iprm,fprm,work,iw,cof,bndc,rhs,phi,mgopt,ierror)
      write (*,107) ierror
107 format(' ierror = ',i2)
      if (ierror.gt.0) call exit(0)

C
C      compute and print maximum error
C
      call error(nx,ny,nz,phi,errm)
      write(*,108) errm
108 format(' maximum error = ',e10.3)
end

      subroutine error(nx,ny,nz,phi,errm)
C
C      compute the error in the estimate in phi
C
      implicit none
      integer nx,ny,nz
      complex phi(nx,ny,nz)
      real xa,xb,yc,yd,ze,zf,tolmax,relmax,errm,tpi
      common/ftcud3/xa,xb,yc,yd,ze,zf,tolmax,relmax,tpi
      real dlx,dly,dlz,x,y,z
      complex pxx,pyy,pzz,px,py,pz,pe
      integer i,j,k
      dlx = (xb-xa) / (nx-1)
      dly = (yd-yc) / (ny-1)
      dlz = (zf-ze) / (nz-1)
      errm = 0.0
      do k=1,nz
      z = ze+(k-1)*dlz
      do j=1,ny
      y = yc+float(j-1)*dly
      do i=1,nx
      x = xa+float(i-1)*dlx
      call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
      errm = amax1(errm,abs(phi(i,j,k)-pe))
      end do
      end do
      end do
      return
end

```

```

subroutine cof(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
c
c      input pde coefficients at grid point (x,y,z) in
the solution region
c      to subroutine cuh3
c
implicit none
complex cxx,cyy,czz,cx,cy,cz,ce
real x,y,z
cxx = cmplx(y,z)
cyy = cmplx(x,z)
czz = cmplx(x,y)
cx = (0.,0.)
cy = cx
cz = cz
ce = -cmplx(y+z,x+z)
return
end

subroutine bndc(kbdy,xory,yorz,alfa,gbdy)
c
c      dummy subroutine since no mixed derivative b.c.
c
implicit none
integer kbdy
real xory,yorz
complex alfa,gbdy
return
end

subroutine exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
c
c      this subroutine is used to set an exact solution
for testing cuh3
c
implicit none
real x,y,z
complex pxx,pyy,pzz,px,py,pz,pe
real xa,xb,yc,yd,ze,zf,tolmax,relmax,tpi
common/ftcud3/xa,xb,yc,yd,ze,zf,tolmax,relmax,tpi
pe = exp(x*y)*cmplx(cos(tpi*z),sin(tpi*z))
px = y*pe
py = x*pe
pz = tpi*exp(x*y)*cmplx(-sin(tpi*z),cos(tpi*z))

```

```
pxx = y*y*pe
pyy = x*x*pe
pzz = -tpi*tpi*pe
return
end
```

---

## TCUH34

```
C
C      file tcuh34.f
C
C      * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK version 5.0.1
*
C      *
*
C      *               A Fortran Package of Multigrid
*
C      *
*
C      *               Subroutines and Example Programs
*
C      *
```

```
*  
C      *      for Solving Elliptic Partial Differential  
Equations      *  
C      *  
*  
C      *          by  
*  
C      *  
*  
C      *          John Adams  
*  
C      *  
*  
C      *          of  
*  
C      *  
*  
C      *          the National Center for Atmospheric  
Research      *  
C      *  
*  
C      *          Boulder, Colorado (80307)  
U.S.A.      *  
C      *  
*  
C      *          which is sponsored by  
*  
C      *  
*  
C      *          the National Science Foundation  
*  
C      *  
*  
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C  
C ... purpose  
C  
C     test program for the MUDPACK solver cuh34  
C  
C ... required MUDPACK files  
C  
C     cuh34.f, cuh3.f, cudcom.f, cud3ln.f, cud3pn.f  
C
```

```

C
C
*****
*
C
*****
*
C
C      sample program/test driver for cuh34
C
C
*****
**
C
*****
**
C
C      a sample program/test driver for cuh34 is below.
it can be
c      executed as an initial test.  the output is listed
for the
c      test case described.
c
c      test the driver below by solving the complex
nonseparable
c      linear elliptic pde
c
c          cmplx(y,z)*d(dp/dx)/dx +
cmplx(x,z)*d(dp/dy)/dy +
c
c          cmplx(x,y)*d(dp/dz)/dz -
cmplx(y+z,x+y)*pe(x,y,z)
c
c          = r(x,y,z)
c
c      on a 21 X 21 X 57 uniform grid superimposed on the
region
c
c          [0.5,1.0] x [0.5,1.0] x [0.0,1.0]
c
c      Assume specified boundary conditions at all x and
y boundaries
c      and periodic in the z direction.  Use line

```

```

relaxation in the
c      z direction.  The exact solution
c
c      pe(x,y,z) =
exp(x*y)*cmplx(cos(tpi*z),sin(tpi*z))
c
c      is used for testing (tpi = 8.0*atan(1.0)).
Choosing grid
c      size arguments
c
c      ixp = 5, jyq = 5, kzs = 7, iex = 3, jey = 3, kez
= 4
c
c      will give the gridcoarsening
c
c      21 X 21 X 57 > 11 X 11 X 29 > 6 X 6 X 15 > 6 X 6
X 8
c
c      The coarsest 6 by 6 by 8 (x,y,z) grid has too many
points
c      for effective error reduction with relaxation
alone. cuh3
c      uses a direct method when this grid is encountered
within
c      multigrid cycling thus maintaining multigrid
convergence rates.
c      After two cycles are executed with cuh3, cuh34 is
called to
c      compute a fourth-order estimate
c
c ****
c      output (64 bit floating point arithmetic)
c
*****
c
c      cuh3 test
c
c      input arguments
c      intl = 0 nxa = 1 nxz = 1 nyc = 1 nyd = 1
c      nze = 0 nzf = 0
c      ixp = 5 jyq = 5 kzs = 7
c      iex = 3 jey = 3 kez = 4
c      nx = 21 ny = 21 nz = 57 iguess = 0 maxcy = 2
c      method = 3 work space length input = 488841

```

```

C      xa =  0.50 xb =  1.00
C      yc =  0.50 yd =  1.00
C      ze =  0.00 zf =  1.00
C      tolmax =  0.000E+00
C
C      multigrid options
C      kcycle =  2
C      iprer =  2
C      ipost =  1
C      interpol =  3
C
C      discretization call to cuh3 intl =  0
C      ierror =  0 minimum work space =  488841
C
C      approximation call to cuh3
C      intl =  1 method =  3 iguess =  0 maxcy =  2
C      ierror =  0
C      maximum error =  0.112E-02
C
C      cuh34 test ierror =  0
C      maximum error =  0.510E-05
C
C
*****  

****  

C      end of output
C
*****  

****  

C
C      program tcuh34
C      implicit none
C
C      set grid sizes with parameter statements
C
C      integer
iixp,jjyq,kkzr,iiex,jjey,kkez,llwork,nnx,nny,nnz
C      integer mmx,mmy,mmz
C      parameter(iixp=5,jjyq=5,kkzr=7)
C      parameter (mmx=iixp+1,mmy=jjyq+1,mmz+kkzr+1)
C      parameter(iiex=3,jjey=3,kkez=4)
C      parameter (nnx = iixp*2** (iiex-1)+1)
C      parameter (nny = jjyq*2** (jjey-1)+1)
C      parameter (nnz = kkzr*2** (kkez-1)+1)

```

```

C
C      set minimal work space required (see tcuh3.f)
C
C      parameter (llwork = 488841)
C
C
C      dimension solution,right hand side, and work
arrays
C
C      complex
phi(nnx,nny,nnz),rhs(nnx,nny,nnz),work(llwork)
      integer iw(mmx,mmy,mmz),iprm(23),mgopt(4)
      real fprm(8)
C
C      put integer and floating point arguments names in
contiguous
C      storeage labelling
C
C      integer
intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex,jey,
+
kez,nx,ny,nz,iguess,maxcy,method,meth2,nwork,
+
lwrkqd,itero

common/itcud3/intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,i
ex,jey,
+
kez,nx,ny,nz,iguess,maxcy,method,meth2,nwork,
+
lwrkqd,itero

      real xa,xb,yc,yd,ze,zf,tolmax,relmax,tpi
      common/ftcud3/xa,xb,yc,yd,ze,zf,tolmax,relmax,tpi
      real dlx,dly,dlz,x,y,z,errm
      complex cxx,cyy,czz,cx,cy,cz,ce,
pxx,pyy,pzz,px,py,pz,pe
      integer i,j,k,ierror
      equivalence(intl,iprm)
      equivalence(xa,fprm)
C
C      declare coefficient and boundary condition input
subroutines external
C
      external cof,bndc
      tpi = 8.0*atan(1.0)

```

```
c
c      set for initial call
c
c      intl = 0
c
c      set boundary condition flags
c
nxa = 1
nxb = 1
nyc = 1
nyd = 1
nze = 0
nzf = 0
c
c      set grid sizes from parameter statements
c
ixp = iixp
jyq = jjyq
kzr = kkzr
iex = iiex
jey = jjey
kez = kkez
nx = nnx
ny = nny
nz = nnz
c
c      set two multigrid cycles
c
maxcy = 2
c
c      set work space length approximation from parameter
statement
c
nwork = llwork
c
c      set method of relaxation--line in the z direction
c
method = 3
c
c      meth2 only used in planar relaxation--but set
c
meth2 = 0
c
c      set full multigrid cycling by flagging no initial
```

```

guess at the finest
c      grid level
c
c      iguess = 0
c
c      set end points of solution region in (x,y,z) space
c
c      xa = 0.5
c      xb = 1.0
c      yc = 0.5
c      yd = 1.0
c      ze = 0.0
c      zf = 1.0
c
c      set default multigrid options
c
c      mgopt(1) = 2
c      mgopt(2) = 2
c      mgopt(3) = 1
c      mgopt(4) = 3
c
c      set mesh increments
c
c      dlx = (xb-xa)/float(nx-1)
c      dly = (yd-yc)/float(ny-1)
c      dlz = (zf-ze)/float(nz-1)
c
c      set for no error control
c
c      tolmax = 0.0
c
c      set right hand side in rhs and phi to zero
c
c      do k=1,nz
c      z = ze+(k-1)*dlz
c      do j=1,ny
c          y = yc+float(j-1)*dly
c          do i=1,nx
c              x = xa+float(i-1)*dlx
c              call cof(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
c              call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
c              rhs(i,j,k) =
c              cxx*pxx+cyy*pyy+czz*pzz+cx*px+cy*py+cz*pz+ce*pe
c              phi(i,j,k) = (0.,0.)

```

```

    end do
end do
    end do
c
c      set specified values at x and y boundaries in phi
c
x = xa
do k=1,nz
z = ze+(k-1)*dlz
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(1,j,k) = pe
end do
    end do
x = xb
do k=1,nz
z = ze+(k-1)*dlz
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(nx,j,k) = pe
end do
    end do
y = yc
do k=1,nz
z = ze+(k-1)*dlz
do i=1,nx
x = xa+float(i-1)*dlx
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(i,1,k) = pe
end do
    end do
y = yd
do k=1,nz
z = ze+(k-1)*dlz
do i=1,nx
x = xa+float(i-1)*dlx
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(i,ny,k) = pe
end do
    end do
write(6,50)
50 format(//' cuh3 test ')

```

```

C
C      print input arguments
C

write(6,100) intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex
,jey,kez,
+
nx,ny,nz,iguess,maxcy,method,nwork,xa,xb,yc,yd,ze,zf,
+
tolmax,mgopt(1),mgopt(2),mgopt(3),mgopt(4)
100 format(/' input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =
',i2,' nyd = 'i2,
+/' nze = ',i2, ' nzf = ',i2,
+/' ixp = ',i2,' jyq = 'i2,' kzr = 'i2,
+/' iex = ',i2, ' jey = 'i2, ' kez = 'i2,
+/' nx = 'i3,' ny = 'i3,' nz = 'i3, ' iguess =
'i2,' maxcy = 'i2,
+/' method = 'i2, ' work space length input = ',i7,
+/' xa = 'f5.2,' xb = 'f5.2,
+/' yc = 'f5.2,' yd = 'f5.2,
+/' ze = 'f5.2,' zf = 'f5.2,
+/' tolmax = ' ,e10.3
+/' multigrid options '
+/' kcycle = ',i2
+/' iprer = ',i2
+/' ipost = ',i2
+/' interpol = ',i2 )

C
C      discretize pde
C

      write(*,104) intl
104 format(/' discretization call to cuh3', ' intl =
', i2)
      call
cuh3(iprm,fprm,work,iw,cof,bndc,rhs,phi,mgopt,ierror)
      write (*,105) ierror,iprm(22)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)
C
C      approximate pde
C

      intl = 1

```

```

        write(*,106) intl,method,iguess,maxcy
106 format(' approximation call to cuh3 ',
          +' intl = ',i2, ' method = ',i2,' iguess = ',i2,
          maxcy = ',i2)
          call
cuh3(iprm,fprm,work,iw,cof,bndc,rhs,phi,mgopt,ierror)
        write (*,107) ierror
107 format(' ierror = ',i2)
        if (ierror.gt.0) call exit(0)

C
C      compute and print maximum error
C
        call error(nx,ny,nz,phi,errm)
        write(*,108) errm
108 format(' maximum error = ',e10.3)

C
C      fourth-order computation
C
        call cuh34(work,iw,phi,ierror)
        write (*,109) ierror
109 format(' cuh34 test ', ' ierror = ',i2)
        if (ierror.gt.0) call exit(0)
        call error(nx,ny,nz,phi,errm)
        write(*,108) errm
        end

subroutine error(nx,ny,nz,phi,errm)
C
C      compute the error in the estimate in phi
C
        implicit none
        integer nx,ny,nz
        complex phi(nx,ny,nz)
        real xa,xb,yc,yd,ze,zf,tolmax,relmax,errm,tpi
        common/ftcud3/xa,xb,yc,yd,ze,zf,tolmax,relmax,tpi
        real dlx,dly,dlz,x,y,z
        complex pxx,pyy,pzz,px,py,pz,pe
        integer i,j,k
        dlx = (xb-xa) / (nx-1)
        dly = (yd-yc) / (ny-1)
        dlz = (zf-ze) / (nz-1)
        errm = 0.0
        do k=1,nz
          z = ze+(k-1)*dlz

```

```

do j=1,ny
    y = yc+float(j-1)*dly
    do i=1,nx
        x = xa+float(i-1)*dlx
        call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
        errm = amax1(errm,abs(phi(i,j,k)-pe))
    end do
end do
end do
return
end

subroutine cof(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
c
c      input pde coefficients at grid point (x,y,z) in
the solution region
c      to subroutine cuh3
c
implicit none
complex cxx,cyy,czz,cx,cy,cz,ce
real x,y,z
cxx = cmplx(y,z)
cyy = cmplx(x,z)
czz = cmplx(x,y)
cx = (0.,0.)
cy = cx
cz = cz
ce = -cmplx(y+z,x+z)
return
end

subroutine bndc(kbdy,xory,yorz,alfa,gbdy)
c
c      dummy subroutine since no mixed derivative b.c.
c
implicit none
integer kbdy
real xory,yorz
complex alfa,gbdy
return
end

subroutine exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
c

```

```

c      this subroutine is used to set an exact solution
for testing cuh3
c
implicit none
real x,y,z
complex pxx,pyy,pzz,px,py,pz,pe
real xa,xb,yc,yd,ze,zf,tolmax,relmax,tpi
common/ftcud3/xa,xb,yc,yd,ze,zf,tolmax,relmax,tpi
pe = exp(x*y)*cmplx(cos(tpi*z),sin(tpi*z))
px = y*pe
py = x*pe
pz = tpi*exp(x*y)*cmplx(-sin(tpi*z),cos(tpi*z))
pxx = y*y*pe
pyy = x*x*pe
pzz = -tpi*tpi*pe
return
end

```

TMUD2

```
C      file tmud2.f
C
C      * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *               University Corporation for Atmospheric
Research           *
C      *
*
C      *               all rights reserved
*
C      *
```

\*  
C \* MUDPACK version 5.0.1  
\*  
C \*  
\*  
C \* A Fortran Package of Multigrid  
\*  
C \*  
\*  
C \* Subroutines and Example Programs  
\*  
C \*  
\*  
C \* for Solving Elliptic Partial Differential  
Equations \*  
C \*  
\*  
C \* by  
\*  
C \*  
\*  
C \* John Adams  
\*  
C \*  
\*  
C \* of  
\*  
C \*  
\*  
C \* the National Center for Atmospheric  
Research \*  
C \*  
\*  
C \* Boulder, Colorado (80307)  
U.S.A. \*  
C \*  
\*  
C \* which is sponsored by  
\*  
C \*  
\*  
C \* the National Science Foundation  
\*  
C \*

```

*
C      * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C
C ... purpose
C
C      test program for the MUDPACK solver mud2
C
C ... required MUDPACK files
C
C      mud2.f, mudcom.f
C
C
C ****
*
C
C ****
*
C
C      sample program/test driver for mud2
C
C
C ****
**
C
C ****
**
C
C      a sample program/test driver for mud2 is below. it
can be
C      executed as an initial test. output is listed for
the test
C      case. test mud2 by solving the elliptic pde
C
C      (1.+y**2)*pxx + exp(-(x+y))* (pyy-py) -
(x+y)*pe = r(x,y)
C
C      on the unit square with specified boundary
conditions at
C      xb = 1.0, yc = 0.0 and mixed boundary conditions
C
C      dp/dx - y*p(0,y) = g(y)   (at x=0.)
C
C      and

```

```

C
C           dp/dy + x*p(x,1) = h(x)   (at y=1.) .
C
C       use line relaxation in the y direction and choose
C       a grid as close
C       to 50 by 100 as the grid size arguments allow. use
C       the exact
C       solution
C
C           pe(x,y) = x**5 + y**5 + 1.0
C
C       for testing. one full multigrid cycle (no initial
C       guess) with
C       the default multigrid options is executed and
C       reaches discretization
C       level error.
C
C
C
C ****
*****
C       output (32 bit floating point arithmetic)
C
C ****
*****
C ****
*****
C
C       mud2 test
C
C       integer input arguments
C       intl = 0 nxa = 2 nxb = 1 nyc = 1 nyd = 2
C       ixp = 3 jyq = 3 iex = 5 jey = 6
C       nx = 49 ny = 97 iguess = 0 maxcy = 1
C       method = 2 work space estimate = 83964
C
C       multigrid option arguments
C       kcycle = 2
C       iprer = 2
C       ipost = 1
C       interpol = 1
C
C       floating point input parameters
C       xa = 0.000 xb = 1.000 yc = 0.000 yd = 1.000
C       tolerance (error control) = 0.000E+00
C

```

```

c      discretization call to mud2  intl =  0
c      ierror =  0 minimum work space =    70048
c
c      approximation call to mud2
c      intl =  1 method =  2 iguess =  0
c      ierror =  0
c      maximum error   =   0.333E-03
c
c
c ****
c ****
c      end of output
c
c ****
c ****
c
c      program tmud2
c      implicit none
c      integer iixp,jjyq,iiex,jjey,nnx,nny,isx,jsy,llwork
c
c      set grid sizes with parameter statements
c
c      parameter (iixp = 3 , jjyq = 3 , iiex = 5, jjey =
6)
c          parameter (nnx=iixp*2** (iiex-1)+1,
nny=jjyq*2** (jjey-1)+1)
c
c      set work space length approximation for line-y
relaxation (see mud2.d)
c
c      parameter(isx=0,jsy=3)
c      parameter
c      (llwork=4* (nnx*nny* (10+isx+jsy)+8* (nnx+nny+2))/3)
c          real phi(nnx,nny),rhs(nnx,nny),work(llwork)
c
c      put integer and floating point argument names in
contiguous
c      storeage for labelling in vectors iprm,fprm
c
c      integer iprm(16),mgopt(4)
c      real fprm(6)
c      integer
c      intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny,
```

```

+
iguess,maxcy,method,nwork,lwrkqd,itero

common/itmud2/intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny
,
+
iguess,maxcy,method,nwork,lwrkqd,itero
    real xa,xb,yc,yd,tolmax,relmax
    common/ftmud2/xa,xb,yc,yd,tolmax,relmax
    equivalence(intl,iprm)
    equivalence(xa,fprm)
    integer i,j,ierror
    real
dlx,dly,x,y,cxx,cyy,cx,cy,ce,pxx,pyy,px,py,pe,errmax
c
c      declare coefficient and boundary condition input
subroutines external
c
        external cof,bndc
c
c
c      set input integer arguments
c
        intl = 0
c
c      set boundary condition flags
c
        nxa = 2
        nxb = 1
        nyc = 1
        nyd = 2
c
c      set grid sizes from parameter statements
c
        ixp = iixp
        jyq = jjyq
        iex = iiex
        jey = jjey
        nx = nnx
        ny = nny
c
c      set multigrid arguments (w(2,1) cycling with fully
weighted
c      residual restriction and cubic prolongation)

```

```
c
mgopt(1) = 2
mgopt(2) = 2
mgopt(3) = 1
mgopt(4) = 3
c
c      set for one cycle
c
maxcy = 1
c
c      set no initial guess forcing full multigrid
cycling
c
iguess = 0
c
c      set work space length approximation from parameter
statement
c
nwork = llwork
c
c      set line-y relaxation
c
method = 2
c
c      set end points of solution rectangle in (x,y)
space
c
xa = 0.0
xb = 1.0
yc = 0.0
yd = 1.0
c
c      set mesh increments
c
dlx = (xb-xa)/float(nx-1)
dly = (yd-yc)/float(ny-1)
c
c      set for no error control flag
c
tolmax = 0.0
c
c      set right hand side in rhs
c      initialize phi to zero
c
```

```

        do i=1,nx
        x = xa+float(i-1)*dlx
        do j=1,ny
          y = yc+float(j-1)*dly
          call cof(x,y,cxx,cyy,cx,cy,ce)
          call exact(x,y,pxx,pyy,px,py,pe)
          rhs(i,j) = cxx*pss+cyy*pss+cx*px+cy*py+ce*pe
          phi(i,j) = 0.0
        end do
        end do
c
c      set specified boundaries in phi
c
        x = xb
        do j=1,ny
          y = yc+float(j-1)*dly
          call exact(x,y,pxx,pyy,px,py,pe)
          phi(nx,j) = pe
        end do
        y = yc
        do i=1,nx
          x = xa+float(i-1)*dlx
          call exact(x,y,pxx,pyy,px,py,pe)
          phi(i,1) = pe
        end do
        write(*,100)
100 format(//' mud2 test ')
        write (*,101) (iprm(i),i=1,15)
101 format(/' integer input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =
',i2,' nyd = ',i2,
+/' ixp = ',i2,' jyq = ',i2,' iex = ',i2,' jey =
',i2
+/' nx = ',i3,' ny = ',i3,' iguess = ',i2,' maxcy =
',i2,
+/' method = ',i2, ' work space estimate = ',i7)
        write (*,102) (mgopt(i),i=1,4)
102 format(/' multigrid option arguments ',
+/' kcycle = ',i2,
+/' iprer = ',i2,
+/' ipost = ',i2
+/' interpol = ',i2)
        write(*,103) xa,xb,yc,yd,tolmax
103 format(/' floating point input parameters ',

```

```

      +' xa = ',f6.3,' xb = ',f6.3,' yc = ',f6.3,' yd =
      ',f6.3,
      +' tolerance (error control) =    ',e10.3)
C
C      intitialization call
C
      write(*,104) intl
 104 format(/' discretization call to mud2', ' intl =
      ', i2)
      call
mud2(iprm,fprm,work,cof,bndc,rhs,phi,mgopt,ierror)
C
C      print error parameter and minimum work space
requirement
C
      write (*,105) ierror,iprm(16)
 105 format(' ierror = ',i2, ' minimum work space =
      ',i7)
      if (ierror.gt.0) call exit(0)
C
C      attempt solution
C
      intl = 1
      write(*,106) intl,method,iguess
 106 format(/' approximation call to mud2',
      +' intl = ',i2, ' method = ',i2,' iguess = ',i2)
      call
mud2(iprm,fprm,work,cof,bndc,rhs,phi,mgopt,ierror)
      write (*,107) ierror
 107 format(' ierror = ',i2)
      if (ierror.gt.0) call exit(0)
C
C      compute and print maximum norm of error
C
      errmax = 0.0
      do j=1,ny
      y = yc+(j-1)*dly
      do i=1,nx
      x = xa+(i-1)*dlx
      call exact(x,y,pxx,pyy,px,py,pe)
      errmax = amax1(errmax,abs((phi(i,j)-pe)))
      end do
      end do
      write(*,108) errmax

```

```

108 format(' maximum error = ',e10.3)
      end

      subroutine cof(x,y,cxx,cyy,cx,cy,ce)
c
c      input pde coefficients at any grid point (x,y) in
the solution region
c
      implicit none
      real x,y,cxx,cyy,cx,cy,ce
      cxx = 1.+y*y
      cyy = exp(-(x+y))
      cx = 0.
      cy = -cyy
      ce = -(x+y)
      return
      end

      subroutine bndc(kbdy,xory,alfa,gbdy)
c
c      input mixed derivative b.c. to mud2
c
      implicit none
      integer kbdy
      real xory,alfa,gbdy,x,y,pe,px,py,pxx,pyy
      real xa,xb,yc,yd,tolmax,relmax
      common/ftmud2/xa,xb,yc,yd,tolmax,relmax
      if (kbdy.eq.1) then ! x=xa boundary
      y = xory
      x = xa
      call exact(x,y,pxx,pyy,px,py,pe)
      alfa = -y
      gbdy = px + alfa*pe
      return
      end if
      if (kbdy.eq.4) then ! y=yd boundary
      y = yd
      x = xory
      call exact(x,y,pxx,pyy,px,py,pe)
      alfa = x
      gbdy = py + alfa*pe
      return
      end if
      end

```

```
      subroutine exact(x,y,pxx,pyy,px,py,pe)
C
C      this subroutine is used to set an exact solution
for testing mud2
C
      implicit none
      real x,y,pxx,pyy,px,py,pe
      pe = x**5+y**5+1.
      px = 5.*x**4
      py = 5.*y**4
      pxx = 20.*x**3
      pyy = 20.*y**3
      return
      end
```

---

## TMUD24

```
C
C      file tmud24.f
C
C      * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *                               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *                               all rights reserved
*
C      *
*
C      *                               MUDPACK version 5.0.1
```

\*  
C \*  
\* C \* A Fortran Package of Multigrid  
\* C \*  
\* C \*  
\* C \* Subroutines and Example Programs  
\* C \*  
\* C \*  
\* C \* Equations \* for Solving Elliptic Partial Differential  
\* C \*  
\* C \* by  
\* C \*  
\* C \*  
\* C \* John Adams  
\* C \*  
\* C \*  
\* C \* of  
\* C \*  
\* C \*  
\* C \* Research \* the National Center for Atmospheric  
\* C \*  
\* C \* U.S.A. \* Boulder, Colorado (80307)  
\* C \*  
\* C \* which is sponsored by  
\* C \*  
\* C \*  
\* C \* Foundation \* the National Science Foundation  
\* C \*  
\* C \*

```

* * * * *
C
C ... purpose
C
C      test program for the mudpack solver mud24
C
C ... required files
C
C      mud24.f, mud2.f, mudcom.f
C
C
C
C ****
*
C
C ****
*
C
C      sample program/test driver for mud24
C
C
C ****
**
C
C ****
**
C
C      a sample program/test driver for mud24 is below.
it can be
c      executed as an initial test. output is listed for
the case
c      described.
C
C      test the driver below by solving the elliptic pde
C
C      (1.+y**2)*pxx + exp(-(x+y))* (pyy-py) -
(x+y)*pe = r(x,y)
C
C      on the unit square with specified boundary
conditions at
c      xb = 1.0, yc = 0.0 and mixed boundary conditions
C

```

```

C           dp/dx - y*p(xa,y) = g(y)   (at x=0)
C
C           and
C
C           dp/dy + x*p(x,yd) = h(x)   (at y=1).
C
C           use line relaxation in the y direction and choose
C           a grid as close
C           to 50 by 100 as the grid size parameters allow.
C           use the exact
C           solution
C
C           pe(x,y) = x**5 + y**5 + 1.0
C
C           for testing.
C
C           the default multigrid options with no initial
C           guess and two
C           cycles are used when first calling mud2 to yield a
C           second-
C           order estimate.  then mud24 is called to produce a
C           fourth-order
C           approximation.
C
C
C
C
*****
*****  

C           output (64 bit floating point arithmetic)
C
C
*****
*****  

*****  

C
C           mud2 test
C
C
C           integer input arguments
C           intl = 0 nxa = 2 nxb = 1 nyc = 1 nyd = 2
C           ixp = 3 jyq = 3 iex = 5 jey = 6
C           nx = 49 ny = 97 iguess = 0 maxcy = 2
C           method = 2 work space estimate = 70048
C
C
C           multigrid option arguments
C           kcycle = 2

```

```

c      iprer = 2
c      ipost = 1
c      interpol = 3
c
c      floating point input parameters
c      xa = 0.000 xb = 1.000 yc = 0.000 yd = 1.000
c      tolerance (error control) = 0.000E+00
c
c      discretization call to mud2 intl = 0
c      ierror = 0 minimum work space = 70048
c
c      approximation call to mud2
c      intl = 1 method = 2 iguess = 0
c      ierror = 0
c      maximum error = 0.342E-03
c
c      mud24 test ierror = 0
c      maximum error = 0.337E-05
c
c
c ****
c ****
c      end of output
c
c ****
c ****
c
c      program tmud24
c      implicit none
c      integer iixp,jjyq,iiex,jjey,nnx,nny,llwork
c
c      set grid sizes with parameter statements
c
c      parameter (iixp = 3 , jjyq = 3 , iiex = 5, jjey =
6)
c      parameter (nnx=iixp*2** (iiex-1)+1,
nny=jjyq*2** (jjey-1)+1)
c
c      set minimal required work space length (see
tmud2.f)
c
c      parameter (llwork=70048)
c      real phi(nnx,nny),rhs(nnx,nny),work(llwork)

```

```

c
c      put integer and floating point argument names in
contiguous
c      storeage for labelling in vectors iprm,fprm
c
integer iprm(16),mgopt(4)
real fprm(6)
integer
intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero

common/itmud2/intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny
,
+
iguess,maxcy,method,nwork,lwrkqd,itero
real xa,xb,yc,yd,tolmax,relmax
common/ftmud2/xa,xb,yc,yd,tolmax,relmax
equivalence(intl,iprm)
equivalence(xa,fprm)
integer i,j,ierror
real
dlx,dly,x,y,cxx,cyy,cx,cy,ce,pxx,pyy,px,py,pe,errmax
c
c      declare coefficient and boundary condition input
subroutines external
c
external cof,bndc
c
c      set input integer arguments
c
intl = 0
c
c      set boundary condition flags
c
nxa = 2
nxb = 1
nyc = 1
nyd = 2
c
c      set grid sizes from parameter statements
c
ixp = iixp

```

```

jyq = jjyq
iex = iiex
jey = jjey
nx = nnx
ny = nny
c
c      set multigrid arguments (w(2,1) cycling with fully
weighted
c      residual restriction and cubic prolongation)
c
mgopt(1) = 2
mgopt(2) = 2
mgopt(3) = 1
mgopt(4) = 3
c
c      set two mg cycles for second-order approximation
c
maxcy = 2
c
c      set no initial guess forcing full multigrid
cycling
c      on initial call
c
iguess = 0
c
c      set work space length approximation from parameter
statement
c
nwork = llwork
c
c      set line-y relaxation
c
method = 2
c
c      set end points of solution rectangle in (x,y)
space
c
xa = 0.0
xb = 1.0
yc = 0.0
yd = 1.0
c
c      set mesh increments
c

```

```

dlx = (xb-xa)/float(nx-1)
dly = (yd-yc)/float(ny-1)

C
C      set for no error control flag
C
C      tolmax = 0.0
C
C      set right hand side in rhs
C      initialize phi to zero
C
do i=1,nx
x = xa+float(i-1)*dlx
do j=1,ny
y = yc+float(j-1)*dly
call cof(x,y,cxx,cyy,cx,cy,ce)
call exact(x,y,pxx,pyy,px,py,pe)
rhs(i,j) = cxx*pxx+cyy*pyy+cx*px+cy*py+ce*pe
phi(i,j) = 0.0
end do
end do
C
C      set specified boundaries in phi
C
x = xb
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,pxx,pyy,px,py,pe)
phi(nx,j) = pe
end do
y = yc
do i=1,nx
x = xa+float(i-1)*dlx
call exact(x,y,pxx,pyy,px,py,pe)
phi(i,1) = pe
end do
write(*,100)
100 format(//' mud2 test ')
write (*,101) (iprm(i),i=1,15)
101 format(/' integer input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =
',i2,' nyd = ',i2,
+/' ixp = ',i2,' jyq = ',i2,' iex = ',i2,' jey =
',i2
+/' nx = ',i3,' ny = ',i3,' iguess = ',i2,' maxcy =

```

```

'i2,
+/' method = ',i2, ' work space estimate = ',i7)
    write (*,102) (mgopt(i),i=1,4)
102 format(/' multigrid option arguments ',
+/' kcycle = ',i2,
+/' iprer = ',i2,
+/' ipost = ',i2
+/' interpol = ',i2)
    write(*,103) xa,xb,yc,yd,tolmax
103 format(/' floating point input parameters ',
+/' xa = ',f6.3,' xb = ',f6.3,' yc = ',f6.3,' yd =
',f6.3,
+/' tolerance (error control) =     ',e10.3)
C
C      intitialization call
C
        write(*,104) intl
104 format(/' discretization call to mud2', ' intl =
', i2)
        call
mud2(iprm,fprm,work,cof,bndc,rhs,phi,mgopt,ierror)
C
C      print error parameter and minimum work space
requirement
C
        write (*,200) ierror,iprm(16)
200 format(' ierror = ',i2, ' minimum work space =
',i7)
        if (ierror.gt.0) call exit(0)
C
C      attempt solution
C
        intl = 1
        write(*,106) intl,method,iguess
106 format(/' approximation call to mud2',
+/' intl = ',i2, ' method = ',i2,' iguess = ',i2)
        call
mud2(iprm,fprm,work,cof,bndc,rhs,phi,mgopt,ierror)
        write (*,107) ierror
107 format(' ierror = ',i2)
        if (ierror.gt.0) call exit(0)
C
C      compute and print maximum norm of error
C

```

```

errmax = 0.0
do j=1,ny
y = yc+(j-1)*dly
do i=1,nx
x = xa+(i-1)*dlx
call exact(x,y,pxx,pyy,px,py,pe)
errmax = amax1(errmax,abs((phi(i,j)-pe)))
end do
end do
write(*,201) errmax
201 format(' maximum error = ',e10.3)

C
C      attempt to improve approximation to fourth order
C
call mud24(work,phi,ierror)
write (*,108) ierror
108 format(/' mud24 test ', ' ierror = ',i2)
if (ierror.gt.0) call exit(0)

C
C      compute and print maximum norm of error
C
errmax = 0.0
do j=1,ny
y = yc+(j-1)*dly
do i=1,nx
x = xa+(i-1)*dlx
call exact(x,y,pxx,pyy,px,py,pe)
errmax = amax1(errmax,abs((phi(i,j)-pe)))
end do
end do
write(*,201) errmax
end

subroutine cof(x,y,cxx,cyy,cx,cy,ce)
C
C      input pde coefficients at any grid point (x,y) in
the solution region
C
implicit none
real x,y,cxx,cyy,cx,cy,ce
cxx = 1.+y*y
cyy = exp(-(x+y))
cx = 0.
cy = -cyy

```

```

ce = -(x+y)
return
end

subroutine bndc(kbdy,xory,alfa,gbdy)
C
C      input mixed derivative b.c. to mud2
C
      implicit none
      integer kbdy
      real xory,alfa,gbdy,x,y,pe,px,py,pxx,pyy
      real xa,xb,yc,yd,tolmax,relmax
      common/ftmud2/xa,xb,yc,yd,tolmax,relmax
      if (kbdy.eq.1) then ! x=xa boundary
      y = xory
      x = xa
      call exact(x,y,pxx,pyy,px,py,pe)
      alfa = -y
      gbdy = px + alfa*pe
      return
      end if
      if (kbdy.eq.4) then ! y=yd boundary
      y = yd
      x = xory
      call exact(x,y,pxx,pyy,px,py,pe)
      alfa = x
      gbdy = py + alfa*pe
      return
      end if
      end

subroutine exact(x,y,pxx,pyy,px,py,pe)
C
C      this subroutine is used to set an exact solution
for testing mud2
C
      implicit none
      real x,y,pxx,pyy,px,py,pe
      pe = x**5+y**5+1.
      px = 5.*x**4
      py = 5.*y**4
      pxx = 20.*x**3
      pyy = 20.*y**3

```

```
    return  
    end
```

---

## TMUD24CR

```
C  
C      file tmud24cr.f  
C  
C      * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C      *  
*  
C      *          copyright (c) 2008 by UCAR  
*  
C      *  
*  
C      *          University Corporation for Atmospheric  
Research           *  
C      *  
*  
C      *          all rights reserved  
*  
C      *  
*  
C      *          MUDPACK version 5.0.1  
*  
C      *  
*  
C      *          A Fortran Package of Multigrid  
*  
C      *  
*  
C      *          Subroutines and Example Programs  
*  
C      *  
*  
C      *          for Solving Elliptic Partial Differential  
Equations       *
```



```

*
C
*****
*
C
C      sample program/test driver for mud24cr
C
C
*****
```

\*\*

```

C
*****
```

\*\*

```

C
C      a sample program/test driver for mud24cr is listed
below. it
c      can be executed as an initial test. the output is
listed for
c      test case described.
```

C

```

c      test mud24cr below by solving the nonseparable
elliptic pde
c      with cross derivative term
```

C

```

c      (1.+y**2)*pxx + (1.+x**2)*pyy + 2.*x*y*pxy +
c
c      y*px + x*py - (x*y)*pe = r(x,y)
```

C

```

c      on a grid as close to 50 by 64 as the mudpack size
constraints
c      allow. the solution region is the unit square.
assume a
c      mixed derivative boundary condition at y=1 of the
form
```

C

```

c      -x * dp/dx + (1+x) * dp/dy - x * pe =
gbdyd(x)
```

C

```

c      and specified (Dirchlet) boundary conditions
elsewhere. the
c      exact solution
```

C

```

c      p(x,y) = (x*y)**5
```

C

```
c      is used to set the right hand side, boundary
conditions, and
c      compute the error.
c
c      red/black gauss-seidel point relaxation is used
along with the
c      the default multigrid options. first mud2cr is
called to generate
c      a second-order approximation. then mud24cr is
called to improve
c      the estimate to fourth-order.
c
c
c ****
c      output (64 bit floating point arithmetic)
c
*****
```

c

c mud2cr test

c

c integer input arguments

c intl = 0 nxa = 1 nxb = 1 nyc = 1 nyd = 2

c ixp = 3 jyq = 2 iex = 5 jey = 6

c nx = 49 ny = 65 iguess = 0 maxcy = 3

c method = 0 work space estimate = 51496

c

c multigrid option arguments

c kcyle = 2

c iprer = 2

c ipost = 1

c interpol = 3

c

c floating point input parameters

c xa = 0.000 xb = 1.000 yc = 0.000 yd = 1.000

c tolerance (error control) = 0.000E+00

c

c discretization call to mud2cr intl = 0

c ierror = 0 minimum work space = 51496

c

c approximation call to mud2cr

c intl = 1 method = 0 iguess = 0

c ierror = 0

c maximum error = 0.626E-03

c

```

C      mud24cr test  ierror =  0
C      maximum error   =    0.584E-05
C
C
C ****
** C      end of output
C
C ****
** C
C
C      program tmud24cr
C      implicit none
C
C      set grid size params
C
C      integer iixp,jjyq,iiex,jjey,nnx,nny,llwork
C      parameter (iixp = 3 , jjyq = 2, iiex = 5, jjey = 6
C      )
C      parameter (nnx=iixp*2** (iiex-1)+1,
C      nny=jjyq*2** (jjey-1)+1)
C
C      set minimum required work space (see tmud2cr.f)
C
C      parameter (llwork=51496)
C      real phi(nnx,nny),rhs(nnx,nny),work(llwork)
C
C      put integer and floating point argument names in
C      contiguous
C      storeage for labelling in vectors iprm,fprm
C
C      integer iprm(16),mgopt(4)
C      real fprm(6)
C      integer
C      intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny,
C      +
C      iguess,maxcy,method,nwork,lwrkqd,itero
C
C      common/itmud2cr/intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,
C      ny,
C      +
C      iguess,maxcy,method,nwork,lwrkqd,itero
C      real xa,xb,yc,yd,tolmax,relmax

```

```
common/ftmud2cr/xa,xb,yc,yd,tolmax,relmax
 equivalence(intl,iprm)
 equivalence(xa,fprm)
 integer i,j,ierror
 real
 dlx,dly,x,y,cxx,cxy,cyy,cx,cy,ce,pxx,pxy,pyy,px,py,pe,er
 rmax
c
c      declare coefficient and boundary condition input
subroutines external
c
c      external cofcr,bndcr
c
c
c      set input integer arguments
c
c      intl = 0
c
c      set boundary condition flags
c
c      nxa = 1
c      nxb = 1
c      nyc = 1
c      nyd = 2
c
c      set grid sizes from parameter statements
c
c      ixp = iixp
c      jyq = jjyq
c      iex = iiex
c      jey = jjey
c      nx = nnx
c      ny = nny
c
c      set multigrid arguments (w(2,1) cycling with fully
weighted
c      residual restriction and cubic prolongation)
c
c      mgopt(1) = 2
c      mgopt(2) = 2
c      mgopt(3) = 1
c      mgopt(4) = 3
c
c      set three cycles to ensure second-order approx
```

```

c
      maxcy = 3
c
c      set no initial guess forcing full multigrid
cycling
c
      iguess = 0
c
c      set work space length approximation from parameter
statement
c
      nwork = llwork
c
c      set point relaxation
c
      method = 0
c
c      set end points of solution rectangle in (x,y)
space
c
      xa = 0.0
      xb = 1.0
      yc = 0.0
      yd = 1.0
c
c      set mesh increments
c
      dlx = (xb-xa)/float(nx-1)
      dly = (yd-yc)/float(ny-1)
c
c      set for no error control flag
c
      tolmax = 0.0
c
c      set right hand side in rhs
c      initialize phi to zero
c
      do i=1,nx
      x = xa+float(i-1)*dlx
      do j=1,ny
          y = yc+float(j-1)*dly
          call cofcr(x,y,cxx,cxy,cyy,cx,cy,ce)
          call exacr(x,y,pxx,pxy,pyy,px,py,pe)
          rhs(i,j) =

```

```

cpxx*pxx+cxy*pxy+cyy*pyy+cx*px+cy*py+ce*pe
    phi(i,j) = 0.0
end do
end do
C
C      set specified boundaries in phi at x=xa,xb and
y=yc
C
do j=1,ny
y = yc+float(j-1)*dly
call exacr(xa,y,pxx,pxy,pyy,px,py,pe)
phi(1,j) = pe
call exacr(xb,y,pxx,pxy,pyy,px,py,pe)
phi(nx,j) = pe
end do
do i=1,nx
x = xa+float(i-1)*dlx
call exacr(x,yc,pxx,pxy,pyy,px,py,pe)
phi(i,1) = pe
end do
write(*,100)
100 format(//' mud2cr test ')
write (*,101) (iprm(i),i=1,15)
101 format(/' integer input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =
',i2,' nyd = ',i2,
+/' ipx = ',i2,' jyq = ',i2,' iex = ',i2,' jey =
',i2
+/' nx = ',i3,' ny = ',i3,' iguess = ',i2,' maxcy =
',i2,
+/' method = ',i2, ' work space estimate = ',i7)
write (*,102) (mgopt(i),i=1,4)
102 format(/' multigrid option arguments ',
+/' kcycle = ',i2,
+/' iprer = ',i2,
+/' ipost = ',i2
+/' interpol = ',i2)
write(*,103) xa,xb,yc,yd,tolmax
103 format(/' floating point input parameters ',
+/' xa = ',f6.3,' xb = ',f6.3,' yc = ',f6.3,' yd =
',f6.3,
+/' tolerance (error control) =      ',e10.3)
C
C      initialization call

```

```

c
      write(*,104) intl
 104 format(/' discretization call to mud2cr', ' intl =
', i2)
      call
mud2cr(iprm,fprm,work,cofcr,bndcr,rhs,phi,mgopt,ierror)
      write (*,200) ierror,iprm(16)
 200 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)
c
c      attempt solution
c
      intl = 1
      write(*,106) intl,method,iguess
 106 format(/' approximation call to mud2cr',
+/ ' intl = ',i2, ' method = ',i2,' iguess = ',i2)
      call
mud2cr(iprm,fprm,work,cofcr,bndcr,rhs,phi,mgopt,ierror)
      write (*,107) ierror
 107 format(' ierror = ',i2)
      if (ierror.gt.0) call exit(0)
      if (ierror .le. 0) then
c
c      compute and print maximum norm of error
c
      errmax = 0.0
      do j=1,ny
y = yc+(j-1)*dly
      do i=1,nx
x = xa+(i-1)*dlx
      call exacr(x,y,pxx,pxy,pyy,px,py,pe)
      errmax = amax1(errmax,abs((phi(i,j)-pe)))
      end do
      end do
      write(*,201) errmax
 201 format(' maximum error = ',e10.3)
      end if
c
c      attempt fourth order approximation
c
      call mud24cr(work,cofcr,bndcr,phi,ierror)
      write (*,108) ierror
 108 format(/' mud24cr test', ' ierror = ',i2)

```

```

        if (ierror.gt.0) call exit(0)
C
C      compute and print maximum norm of error
C
        errmax = 0.0
        do j=1,ny
          y = yct+(j-1)*dly
          do i=1,nx
            x = xa+(i-1)*dlx
            call exacr(x,y,pxx,pxy,pyy,px,py,pe)
            errmax = amax1(errmax,abs((phi(i,j)-pe)))
          end do
        end do
        write(*,201) errmax
      end

      subroutine cofcr(x,y,cxx,cxy,cyy,cx,cy,ce)
C
C      input pde coefficients at any grid point (x,y) in
C      the solution region
C      (xa.le.x.le.xb,yc.le.y.le.yd) to mud2cr
C
      implicit none
      real x,y,cxx,cxy,cyy,cx,cy,ce
      cxx = 1.+y**2
      cxy = 2.*x*y
      cyy = 1.+x**2
      cx = y
      cy = x
      ce = -(x*y)
      return
    end

      subroutine bndcr(kbdy,xory,alfa,beta,gama,gbdy)
C
C      input mixed "oblique" derivative b.c. to mud2cr
C      at upper y boundary
C
      implicit none
      integer kbdy
      real xory,alfa,beta,gama,gbdy
      integer
      intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny,
      +

```

```

iguess,maxcy,method,nwork,lwrkqd,itero

common/itmud2cr/intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,
ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero
    real xa,xb,yc,yd,tolmax,relmax
    common/ftmud2cr/xa,xb,yc,yd,tolmax,relmax
    real x,y,pxx,pxy,pyy,px,py,pe
    if (kbdy.eq.4) then
c
c      y=yd boundary (nyd must equal 2 if this code is to
be executed).
c      b.c. has the form
alfydz(x)*px+beta*yd*x*py+gamyd(x)*pe = gbdyz(x)
c      where x = yd. alfa,beta,gama,gbdy
corresponding to alfydz(x),
c      betydz(x),gamyd(x),gbdyz(y) must be output.
c
y = yd
x = xory
alfa = -x
beta = 1.+x
gama = -x
call exacr(x,y,pxx,pxy,pyy,px,py,pe)
gbdy = alfa*px + beta*py + gama*pe
return
end if
end

subroutine exacr(x,y,pxx,pxy,pyy,px,py,pe)
c
c      this subroutine is used for setting an exact
solution
c      to test subroutine mud2cr.
c
implicit none
real x,y,pxx,pxy,pyy,px,py,pe
pe = (x*y)**5
px = 5.* (x*y)**4*y
py = 5.* (x*y)**4*x
pxx = 20.* (x*y)**3*y*y
pxy = 25.* (x*y)**4
pyy = 20.* (x*y)**3*x*x

```

```
    return  
    end
```

---

## TMUD24SP

```
C  
C      file tmud24sp.f  
C  
C      * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C      *  
*  
C      *          copyright (c) 2008 by UCAR  
*  
C      *  
*  
C      *          University Corporation for Atmospheric  
Research           *  
C      *  
*  
C      *          all rights reserved  
*  
C      *  
*  
C      *          MUDPACK version 5.0.1  
*  
C      *  
*  
C      *          A Fortran Package of Multigrid  
*  
C      *  
*  
C      *          Subroutines and Example Programs  
*  
C      *  
*  
C      *          for Solving Elliptic Partial Differential  
Equations       *
```



```

C
*****
*
C
C      sample program/test driver for mud24sp
C
C
*****
**
C
*****
**
C
C      a sample program/test driver for mud24sp is listed
below. it
c      can be executed as an initial test. the output is
listed for
c      the test case described.
C
C      test the driver below by solving the separable
elliptic pde
C
C      (1.+x**2)*pxx + exp(1.-y)*(pyy-py) - (x+y)*pe =
r(x,y)
C
C      on the unit square with specified boundary
conditions at
c      xb = 1.0, yc = 0.0 and mixed boundary conditions
C
C      dp/dx - pe(0.0,y) = ga(y)   (at x = 0.0)
C
C      dp/dy + pe(x,1.0) = gd(x)   (at y = 1.0)
C
C      use point relaxation and choose a grid as close to
60 x 50
c      as the grid size constraints allow. use the exact
solution
C
C      pe(x,y) = (x**3+y**3+1.0)/3
C
C      for testing. first mud2sp is called to yield a
second-order
c      approximation. then mud24sp is called to improve

```

```
the estimate.

C
C
*****
C      output (64 bit floating point arithmetic)
C
*****
C
C      mud2sp test
C
C      integer input arguments
C      intl =  0 nxa =  2 nxb =  1 nyc =  1 nyd =  2
C      ixp =  2 jyq =  3 iex =  6 jey =  5
C      nx =  65 ny =  49 iguess =  0 maxcy =  3
C      method =  0 work space estimate =  13264
C
C      multigrid option arguments
C      kcycle =  2
C      iprer =  2
C      ipost =  1
C      interpol =  3
C
C      floating point input parameters
C      xa =  0.000 xb =  1.000 yc =  0.000 yd =  1.000
C      tolerance (error control) =  0.000E+00
C
C      discretization call to mud2sp intl =  0
C      ierror =  0 minimum work space =  13264
C
C      approximation call to mud2sp
C      intl =  1 method =  0 iguess =  0
C      ierror =  0
C      maximum error =  0.496E-04
C
C      mud24sp test ierror =  0
C      maximum error =  0.123E-06
C
C
*****
C      end of output
C
```

```

*****
C
C
    program tmud24sp
        implicit none
        integer iixp,jjyq,iiex,jjey,nnx,nny,llwork
C
C      set grid sizes with parameter statements
C
        parameter (iixp = 2 , jjyq = 3 , iiex = 6, jjey =
5)
            parameter (nnx=iixp*2** (iiex-1)+1,
nny=jjyq*2** (jjey-1)+1)
C
C      set minimal required work space (see tmud2sp.f)
C
        parameter (llwork = 13264)
        real phi(nnx,nny),rhs(nnx,nny),work(llwork)
C
C      put integer and floating point argument names in
contiguous
C      storeage for labelling in vectors iprm,fprm
C
        integer iprm(16),mgopt(4)
        real fprm(6)
        integer
intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero

common/itmud2sp/intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,
ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero
        real xa,xb,yc,yd,tolmax,relmax
        common/ftmud2sp/xa,xb,yc,yd,tolmax,relmax
        equivalence(intl,iprm)
        equivalence(xa,fprm)
        integer i,j,ierror
        real
dlx,dly,x,y,cxx,cyy,cx,cy,ce,pxx,pyy,px,py,pe,errmax
        real cex,cey
C
```

```
c      declare coefficient and boundary condition input
subroutines external
c
      external cofx,cofy,bndsp
c
c      set input integer arguments
c
      intl = 0
c
c      set boundary condition flags
c
      nxa = 2
      nxb = 1
      nyc = 1
      nyd = 2
c
c      set grid sizes from parameter statements
c
      ixp = iixp
      jyq = jjyq
      iex = iiex
      jey = jjey
      nx = nnx
      ny = nny
c
c      set multigrid arguments (w(2,1) cycling with fully
weighted
c      residual restriction and cubic prolongation)
c
      mgopt(1) = 2
      mgopt(2) = 2
      mgopt(3) = 1
      mgopt(4) = 3
c
c      set for three cycles to ensure second-order
approximation
c      is computed
c
      maxcy = 3
c
c      set no initial guess forcing full multigrid
cycling
c
```

```

    iguess = 0
c
c      set work space length approximation from parameter
statement
c
nwork = llwork
c
c      set point relaxation
c
method = 0
c
c      set end points of solution rectangle in (x,y)
space
c
xa = 0.0
xb = 1.0
yc = 0.0
yd = 1.0
c
c      set mesh increments
c
dlx = (xb-xa)/float(nx-1)
dly = (yd-yc)/float(ny-1)
c
c      set for no error control flag
c
tolmax = 0.0
c
c      set right hand side in rhs
c      initialize phi to zero
c
do i=1,nx
x = xa+float(i-1)*dlx
call cofx(x,cxx,cx,cex)
do j=1,ny
y = yc+float(j-1)*dly
call cofy(y,cyy,cy,cey)
ce = cex+cey
call exact(x,y,pxx,pyy,px,py,pe)
rhs(i,j) = cxx*pss+cyy*pss+cx*px+cy*py+ce*pe
phi(i,j) = 0.0
end do
end do
c

```

```

c      set specified boundaries in phi
c
x = xb
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,pxx,pyy,px,py,pe)
phi(nx,j) = pe
end do
y = yc
do i=1,nx
x = xa+float(i-1)*dlx
call exact(x,y,pxx,pyy,px,py,pe)
phi(i,1) = pe
end do
write(*,100)
100 format(//' mud2sp test ')
write (*,101) (iprm(i),i=1,15)
101 format(/' integer input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =
',i2,' nyd = ',i2,
+/' ixp = ',i2,' jyq = ',i2,' iex = ',i2,' jey =
',i2
+/' nx = ',i3,' ny = ',i3,' iguess = ',i2,' maxcy =
',i2,
+/' method = ',i2, ' work space estimate = ',i7)
write (*,102) (mgopt(i),i=1,4)
102 format(/' multigrid option arguments ',
+/' kcycle = ',i2,
+/' iprer = ',i2,
+/' ipost = ',i2
+/' interpol = ',i2)
write(*,103) xa,xb,yc,yd,tolmax
103 format(/' floating point input parameters ',
+/' xa = ',f6.3,' xb = ',f6.3,' yc = ',f6.3,' yd =
',f6.3,
+/' tolerance (error control) =    ',e10.3)
c
c      intitialization call
c
write(*,104) intl
104 format(/' discretization call to mud2sp', ' intl =
', i2)
call
mud2sp(iprm,fprm,work,cofx,cofy,bndsp,rhs,phi,mgopt,ierr

```

```

or)
c
c      print error parameter and minimum work space
requirement
c
      write (*,200) ierror,iprm(16)
200 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)
c
c      attempt solution
c
      intl = 1
      write(*,106) intl,method,iguess
106 format(/' approximation call to mud2sp',
+/ ' intl = ',i2, ' method = ',i2,' iguess = ',i2)
      call
mud2sp(iprm,fprm,work,cofx,cofy,bndsp,rhs,phi,mgopt,ierr
or)
      write (*,107) ierror
107 format(' ierror = ',i2)
      if (ierror.gt.0) call exit(0)
      if (ierror .le. 0) then
c
c      compute and print maximum norm of error
c
      errmax = 0.0
      do j=1,ny
y = yc+(j-1)*dly
      do i=1,nx
x = xa+(i-1)*dlx
      call exact(x,y,pxx,pyy,px,py,pe)
      errmax = amax1(errmax,abs((phi(i,j)-pe)))
      end do
      end do
      write(*,201) errmax
201 format(' maximum error = ',e10.3)
      end if
c
c      attempt to improve approximation to fourth order
c
      call mud24sp(work,phi,ierror)
      write (*,108) ierror
108 format(/' mud24sp test ', ' ierror = ',i2)

```

```

    if (ierror.gt.0) call exit(0)
    if (ierror .le. 0) then
C
C      compute and print maximum norm of error
C
        errmax = 0.0
        do j=1,ny
          y = yc+(j-1)*dly
        do i=1,nx
          x = xa+(i-1)*dlx
          call exact(x,y,pxx,pyy,px,py,pe)
          errmax = amax1(errmax,abs((phi(i,j)-pe)))
        end do
        end do
        write(*,201) errmax
        end if
        end

        subroutine cofx(x,cxx,cx,cex)
C
C      input x dependent coefficients
C
        implicit none
        real x,cxx,cx,cex
        cxx = 1.0+x*x
        cx = 0.0
        cex = -x
        return
        end

        subroutine cofy(y,cyy,cy,cey)
C
C      input y dependent coefficients
C
        implicit none
        real y,cyy,cy,cey
        cyy = exp(1.0-y)
        cy = -cyy
        cey = -y
        return
        end

        subroutine bndsp(kbdy,xory,alfa,gbdy)
C

```

```

c      input mixed derivative b.c. to mud2sp
c
      implicit none
      integer kbdy
      real xory, alfa, gbdy, x, y, pe, px, py, pxx, pyy
      real xa, xb, yc, yd, tolmax, relmax
      common/ftmud2sp/xa, xb, yc, yd, tolmax, relmax
      if (kbdy.eq.1) then ! x=xa boundary
      y = xory
      x = xa
      call exact(x,y,pxx,pyy,px,py,pe)
      alfa = -1.0
      gbdy = px + alfa*pe
      return
      end if
      if (kbdy.eq.4) then ! y=yd boundary
      y = yd
      x = xory
      call exact(x,y,pxx,pyy,px,py,pe)
      alfa = 1.0
      gbdy = py + alfa*pe
      return
      end if
      end

      subroutine exact(x,y,pxx,pyy,px,py,pe)
c
c      set an exact solution for testing mud2sp
c
      implicit none
      real x,y,pxx,pyy,px,py,pe
      pe = (x**3+y**3+1.0)/3.0
      px = x*x
      py = y*y
      pxx = x+x
      pyy = y+y
      return
      end

```

## TMUD2CR

```
C
C      file tmud2cr.f
C
C      * * * * * * * * * * * * * * *
* * * * * * *
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK version 5.0.1
*
C      *
*
C      *               A Fortran Package of Multigrid
*
C      *
*
C      *               Subroutines and Example Programs
*
C      *
*
C      *               for Solving Elliptic Partial Differential
Equations          *
C      *
*
C      *               by
*
C      *
*
C      *               John Adams
*
```

```
C      *
*
C      *                                of
*
C      *
*
C      *            the National Center for Atmospheric
Research           *
C      *
*
C      *            Boulder, Colorado (80307)
U.S.A.          *
C      *
*
C      *            which is sponsored by
*
C      *
*
C      *            the National Science Foundation
*
C      *
*
C      *      * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C
C ... purpose
C
C      test program for the mudpack solver mud2cr
C
C ... required MUDPACK files
C
C      mud2cr.f, mudcom.f
C
C
*****
* *
C
*****
* *
C
C      sample program/test driver for mud2cr
C
C
*****
```

```

**
C
*****
**
C
C
C      a sample program/test driver for mud2cr is listed
below. it
C      can be executed as an initial test. the output is
listed
C      for the test case described.
C
C      test mud2cr below by solving the nonseparable
elliptic pde
C      with cross derivative term
C
C      (1.+y**2)*pxx + (1.+x**2)*pyy + 2.*x*y*pxy +
C
C      y*px + x*py - (x*y)*pe = r(x,y)
C
C      on a grid as close to 50 by 64 as the mudpack size
constraints
C      allow. the solution region is the unit square.
assume a
C      mixed derivative boundary condition at y=1 of the
form
C
C      -x * dp/dx + (1+x) * dp/dy - x * pe =
gbdyd(x) .
C
C      and specified (Dirchlet) boundary conditions
elsewhere. the
C      exact solution
C
C      p(x,y) = (x*y)**5
C
C      is used to set the right hand side, boundary
conditions, and
C      compute the error.
C
C      red/black gauss-seidel point relaxation is used
along with the
C      the default multigrid options. one full multigrid
cycle reaches

```

```
c      discretization level error for this problem.  
c  
c  
c ****  
c      output (32 bit floating point arithmetic)  
c  
*****  
c  
c  
c      mud2cr test  
c  
c      integer input arguments  
c      intl =  0 nxa =  1 nxb =  1 nyc =  1 nyd =  2  
c      ixp =  3 jyq =  2 iex =  5 jey =  6  
c      nx =  49 ny =  65 iguess =  0 maxcy =  1  
c      method =  0 work space estimate =  54686  
c  
c      multigrid option arguments  
c      kcycle =  2  
c      iprer =  2  
c      ipost =  1  
c      interpol =  3  
c  
c      floating point input parameters  
c      xa =  0.000 xb =  1.000 yc =  0.000 yd =  1.000  
c      tolerance (error control) =  0.000E+00  
c  
c      discretization call to mud2cr intl =  0  
c      ierror =  0 minimum work space =  51496  
c  
c      approximation call to mud2cr  
c      intl =  1 method =  0 iguess =  0  
c      ierror =  0  
c      maximum error =  0.623E-03  
c  
c  
c ****  
**  
c      end of output  
c  
*****  
**  
c  
program tmud2cr
```

```

    implicit none
c
c      set grid size params
c
integer iixp,jjyq,iiex,jjey,nnx,nny,llwork
parameter (iixp = 3 , jjyq = 2, iiex = 5, jjey = 6
)
      parameter (nnx=iixp*2** (iiex-1)+1,
nny=jjyq*2** (jjey-1)+1)
c
c      estimate work space for point relaxation (see
mud2cr.d)
c
      parameter (llwork=(7* (nnx+2) * (nny+2)+44*nnx*nny) /3
)
      real phi(nnx,nny),rhs(nnx,nny),work(llwork)
c
c      put integer and floating point argument names in
contiguous
c      storeage for labelling in vectors iprm,fprm
c
integer iprm(16),mgopt(4)
real fprm(6)
integer
intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero

common/itmud2cr/intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,
ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero
      real xa,xb,yc,yd,tolmax,relmax
      common/ftmud2cr/xa,xb,yc,yd,tolmax,relmax
      equivalence(intl,iprm)
      equivalence(xa,fprm)
      integer i,j,ierror
      real
dlx,dly,x,y,cxx,cxy,cyy,cx,cy,ce,pxx,pxy,pyy,px,py,pe,er
rmax
c
c      declare coefficient and boundary condition input
subroutines external
c

```

```
external cofcr,bndcr
c
c
c      set input integer arguments
c
c      intl = 0
c
c      set boundary condition flags
c
c      nxn = 1
c      nxb = 1
c      nyc = 1
c      nyd = 2
c
c      set grid sizes from parameter statements
c
c      ixp = iixp
c      jyq = jjyq
c      iex = iiex
c      jey = jjey
c      nx = nnx
c      ny = nny
c
c      set multigrid arguments (w(2,1) cycling with fully
c weighted
c      residual restriction and cubic prolongation)
c
c      mgopt(1) = 2
c      mgopt(2) = 2
c      mgopt(3) = 1
c      mgopt(4) = 3
c
c      set for one cycle
c
c      maxcy = 1
c
c      set no initial guess forcing full multigrid
c cycling
c
c      iguess = 0
c
c      set work space length approximation from parameter
c statement
c
```

```

nwork = llwork
c
c      set point relaxation
c
method = 0
c
c      set end points of solution rectangle in (x,y)
space
c
xa = 0.0
xb = 1.0
yc = 0.0
yd = 1.0
c
c      set mesh increments
c
dlx = (xb-xa)/float(nx-1)
dly = (yd-yc)/float(ny-1)
c
c      set for no error control flag
c
tolmax = 0.0
c
c      set right hand side in rhs
c      initialize phi to zero
c
do i=1,nx
x = xa+float(i-1)*dlx
do j=1,ny
y = yc+float(j-1)*dly
call cofcr(x,y,cxx,cxy,cyy,cx,cy,ce)
call exacr(x,y,pxx,pxy,pyy,px,py,pe)
rhs(i,j) =
cxx*pxx+cxy*pxy+cyy*pyy+cx*px+cy*py+ce*pe
phi(i,j) = 0.0
end do
end do
c
c      set specified boundaries in phi at x=xa,xb and
y=yc
c
do j=1,ny
y = yc+float(j-1)*dly
call exacr(xa,y,pxx,pxy,pyy,px,py,pe)

```

```

phi(1,j) = pe
call exacr(xb,y,pxx,pxy,pyy,px,py,pe)
phi(nx,j) = pe
end do
do i=1,nx
x = xa+float(i-1)*dlx
call exacr(x,yc,pxx,pxy,pyy,px,py,pe)
phi(i,1) = pe
end do
write(*,100)
100 format(//' mud2cr test ')
write (*,101) (iprm(i),i=1,15)
101 format(/' integer input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxn = ',i2,' nyc =
',i2,' nyd = ',i2,
+/' ixp = ',i2,' jyq = ',i2,' iex = ',i2,' jey =
',i2
+/' nx = ',i3,' ny = ',i3,' iguess = ',i2,' maxcy =
',i2,
+/' method = ',i2, ' work space estimate = ',i7)
write (*,102) (mgopt(i),i=1,4)
102 format(/' multigrid option arguments ',
+/' kcycle = ',i2,
+/' iprер = ',i2,
+/' ipost = ',i2
+/' interpol = ',i2)
write(*,103) xa,xb,yc,yd,tolmax
103 format(/' floating point input parameters ',
+/' xa = ',f6.3,' xb = ',f6.3,' yc = ',f6.3,' yd =
',f6.3,
+/' tolerance (error control) =    ',e10.3)
C
C      initialization call
C
      write(*,104) intl
104 format(/' discretization call to mud2cr', ' intl =
', i2)
      call
mud2cr(iprm,fprm,work,cofcr,bndcr,rhs,phi,mgopt,ierror)
      write (*,200) ierror,iprm(16)
200 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)
C

```

```

c      attempt solution
c
    intl = 1
    write(*,106) intl,method,iguess
106 format(' approximation call to mud2cr',
+/ ' intl = ',i2, ' method = ',i2,' iguess = ',i2)
    call
mud2cr(iprm,fprm,work,cofcr,bndcr,rhs,phi,mgopt,ierror)
    write (*,107) ierror
107 format(' ierror = ',i2)
    if (ierror.gt.0) call exit(0)
    if (ierror .le. 0) then
c
c      compute and print maximum norm of error
c
        errmax = 0.0
        do j=1,ny
y = yc+(j-1)*dly
        do i=1,nx
            x = xa+(i-1)*dlx
            call exacr(x,y,pxx,pxy,pyy,px,py,pe)
            errmax = amax1(errmax,abs((phi(i,j)-pe)))
        end do
        end do
        write(*,201) errmax
201 format(' maximum error = ',e10.3)
        end if
        end

        subroutine cofcr(x,y,cxx,cxy,cyy,cx,cy,ce)
c
c      input pde coefficients at any grid point (x,y) in
the solution region
c      (xa.le.x.le.xb,yc.le.y.le.yd) to mud2cr
c
        implicit none
        real x,y,cxx,cxy,cyy,cx,cy,ce
        cxx = 1.+y**2
        cxy = 2.*x*y
        cyy = 1.+x**2
        cx = y
        cy = x
        ce = -(x*y)
        return

```

```

    end

    subroutine bndcr(kbdy,xory,alfa,beta,gama,gbdy)
C
C      input mixed "oblique" derivative b.c. to mud2cr
C      at upper y boundary
C
        implicit none
        integer kbdy
        real xory,alfa,beta,gama,gbdy
        integer
        intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny,
        +
        iguess,maxcy,method,nwork,lwrkqd,itero

common/itmud2cr/intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,
ny,
        +
        iguess,maxcy,method,nwork,lwrkqd,itero
        real xa,xb,yc,yd,tolmax,relmax
        common/ftmud2cr/xa,xb,yc,yd,tolmax,relmax
        real x,y,pxx,pxy,pyy,px,py,pe
        if (kbdy.eq.4) then
C
C      y=yd boundary (nyd must equal 2 if this code is to
be executed).
C      b.c. has the form
alfyd(x)*px+betyd(x)*py+gamyd(x)*pe = gbdy(x)
C      where x = yorx.  alfa,beta,gama,gbdy
corresponding to alfyd(x),
C      betyd(x),gamyd(x),gbdy(y) must be output.
C
        y = yd
        x = xory
        alfa = -x
        beta = 1.+x
        gama = -x
        call exacr(x,y,pxx,pxy,pyy,px,py,pe)
        gbdy = alfa*px + beta*py + gama*pe
        return
        end if
        end

subroutine exacr(x,y,pxx,pxy,pyy,px,py,pe)

```

```

C
C      this subroutine is used for setting an exact
solution
C      to test subroutine mud2cr.
C
      implicit none
      real x,y,pxx,pxy,pyy,px,py,pe
      pe = (x*y)**5
      px = 5.* (x*y)**4*y
      py = 5.* (x*y)**4*x
      pxx = 20.* (x*y)**3*y*y
      pxy = 25.* (x*y)**4
      pyy = 20.* (x*y)**3*x*x
      return
      end

```

## TMUD2SA

```

C
C      file tmud2sa.f
C
C      * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *                               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *                               all rights reserved
*
C      *
*
C      *                               MUDPACK version 5.0.1

```



```
* * * * *
C
C ... purpose
C
C      test program for the MUDPACK solver mud2
C
C ... required MUDPACK files
C
C      mud2sa.f, mudcom.f
C
C
C
C ****
*
C
C ****
*
C
C      sample program/test driver for mud2sa
C
C
C ****
**
C
C ****
**
C
C
C      a sample program/test driver for mud2sa is below.
it can be
c      executed as an initial test. output is listed for
the test
c      case described.
C
C      test the nonseparable self-adjoint solver mud2sa
by approximating
c      the solution to the following elliptic pde in
divergence form on
c      the full surface of a sphere of radius one.
C
C      div(sigma(t,p)*grad(u(t,p))) -
lambda(t,p)*u(t,p) = f(t,p)
C
```

```

c      t and p are colatitude and longitude in radians.
choose a grid
c      as close to 2.5 degrees radians (73 x 145) as the
mudpack size
c      constraints allow. multiplying thru by sin(t) puts
the pde in the
c      following self-adjoint form suitable for mud2sa:
c
c          d(sin(t)*sigma*du/dt)/dt +
d(sigma/sin(t)*du/dp)/dp -
c
c          sin(t)*lambda*u(t,p) = sin(t)*f(t,p).
c
c
c      for testing use the coefficients and exact
solution:
c
c          sigma(t,p) = 1.5 + (sin(t)*cos(p))**2
c
c          lambda(t,p) = - sigma(t,p)
c
c          u(t,p) =
[sin(t)*(sin(t)*cos(t)*sin(p)*cos(p))]**2
c
c      (the exact solution is the restriction of the
solution u(x,y,z) =
c      (x*y*z)**2 in cartesian coordinates to the surface
of the sphere
c      in spherical coordinates). assume the solution
u(t,p) is specified
c      at the poles and is periodic in longitude.
c
c
*****
***output (32 bit floating point arithmetic)
c
*****
***mud2sa test
c
c      integer input parameters
c      intl = 0 nta = 1 ntb = 1 npc = 0 npd = 0

```

```

c      iitp = 5 jjpq = 5 iiet = 5 jjep = 6
c      nt = 81 np = 161 iguess = 0 maxcy = 1
c      method = 2 work space estimate = 260820

c
c      multigrid option parameters
c      kcycle = 2
c      iprer = 2
c      ipost = 1
c      interpol = 3
c
c      floating point input parameters
c      ta = 0.000 tb = 3.142 pc = 0.000 pd = 6.283
c      tolerance (error control) = 0.000E+00
c
c      discretization call to mud2sa  intl = 0
c      ierror = 0 minimum work space = 225200
c
c      approximation call to mud2sa
c      intl = 1 method = 2 iguess = 0 maxcy = 1
c      tolmax = 0.00
c      ierror = 0
c      maximum error = 0.655E-04
c
c
c ****
c ****
c
c      program tmud2sa
c      implicit none
c      integer iitp,jjpq,iiet,jjep,nnt,nnp,llwork
c
c      set grid size with parameter statements
c
c      parameter(iitp = 5, jjpq = 5, iiet = 5, jjep = 6)
c      parameter ( nnt = iitp*2** (iiet-1) + 1)
c      parameter ( nnp = jjpq*2** (jjep-1) + 1)
c
c      set work space for longitude relaxation only (see
c      mud2sa.d)
c

```

```

    parameter (llwork = 20*nnt*nnp)
c
c      dimension solution,right hand side, and work
arrays
c
      real u(nnt,nnp),r(nnt,nnp),w(llwork)
c
c      dimension input argument vectors and set up
continguous storage
c      for labelling entries
c
      integer iparm(17),mgopt(4)
      real fparm(6)
      integer
intl,nta,ntb,npd,npd,itp,jpq,iet,jep,nt,np,
+
iguess,maxcy,method,nwork,lwork,iter
      common / iprm /
intl,nta,ntb,npd,npd,itp,jpq,iet,jep,nt,np,
+
iguess,maxcy,method,nwork,lwork,iter
      real
ta,tb,pc,pd,tolmax,sinat,cosat,sinap,cosap,dlt,dlp
      common / fprm /
ta,tb,pc,pd,tolmax,sinat(81),cosat(81),
+           sinap(161),cosap(161),dlt,dlp
      integer i,j,ierror
      real pi,sint,cost,sinp,cosp,sigma,dsigdt,dsigdp
      real
ctt,cpp,ct,cp,ce,tmp,dt,dp,dtt,dpp,ue,ut,up,utt,upp
      real sigt,sigp,xlmb,errm,p,t
c
c      equivlance iparm,fparm with labelled commons
iprm,fprm
c
      equivalence (intl,iparm)
      equivalence (ta,fparm)
c
c      declare coefficient and boundary condition input
subroutines external
c
      external sigt,sigp,xlmb,bndc
c
c      set input integer parameters

```

```
c
c      initialization
c
c      intl = 0
c
c      set boundary condition flags: poles specified,
longitude periodic
c
c      nta = 1
c      ntb = 1
c      npc = 0
c      npd = 0
c
c      set grid sizes from parameter statements
c
c      itp = iitp
c      jpq = jjpq
c      iet = iiet
c      jep = jjep
c      nt = nnt
c      np = nnr
c
c      full multigrid cycling (no initial guess at finest
grid)
c
c      iguess = 0
c
c      set one multigrid cycle
c
c      maxcy = 1
c
c      set line relaxation in the longitude direction
c
c      method = 2
c
c      set work space estimate
c
c      nwork = llwork
c
c      set multigrid parameters (w(2,1) cycling with
fully weighted
c      residual restriction and cubic prolongation).
this can also
c      be set by inputting mgopt(1) = 0 to mud2sa.
```

```

c
mgopt(1) = 2
mgopt(2) = 2
mgopt(3) = 1
mgopt(4) = 3
c
c      set floating point input parameters
c
pi = 4.0*atan(1.0)
c
c      interval end points (in radians)
c
ta = 0.0
tb = pi
pc = 0.
pd = pi+pi
c
c      no error control
c
tolmax = 0.0
c
c      set mesh increments
c
dlt = (tb-ta)/float(nt-1)
dlp = (pd-pc)/float(np-1)
c
c      preset sin,cos vectors to save computation on grid
points
c
do i=1,nt
t = ta+(i-1)*dlt
sinat(i) = sin(t)
cosat(i) = cos(t)
end do
do j=1,np
p = pc+(j-1)*dlp
sinap(j) = sin(p)
cosap(j) = cos(p)
end do
c
c      initialize right hand side and solution array
except at poles
c
do i=2,nt-1

```

```

sint = sinat(i)
cost = cosat(i)
do j=1,np
    sinp = sinap(j)
    cosp = cosap(j)
    sigma = 1.5 + (sint*cosp)**2
    dsigdt = 2.0*cosp*cosp*sint*cost
    dsigdp = -2.0*sint*sint*cosp*sinp
c
c          compute expanded pde coefficients
c
    ctt = sint*sigma
    cpp = sigma/sint
    ct = sint*dsigdt + cost*sigma
    cp = dsigdp/sint
    ce = -sint*sigma
c
c          set intermediate variables for exact solution
c
    tmp = (sint*cosp*sint*sinp*cost)
    dt = (2.*sint*cost*cost-sint**3)*(cosp*sinp)
    dp = (cosp**2-sinp**2)*(sint**2*cost)
    dtt = (2.*cost**3-4.*cost*sint**2-
3.*sint**2*cost)*(cosp*sinp)
    dpp = (-4.*cosp*sinp)*(sint**2*cost)
c
c          set continuous solution and partial
derivatives
c
    ue = tmp*tmp
    ut = 2.*tmp*dt
    up = 2.*tmp*dp
    utt = 2.* (dt*dt+tmp*dtt)
    upp = 2.* (dp*dp+tmp*dpp)
c
c          set right hand side of continuous pde on grid
c
    r(i,j) = ctt*utt+cpp*upp+ct*ut+cp*up+ce*ue
c
c          initialize solution array to zero
c
    u(i,j) = 0.0
end do
end do

```

```

C
C      set u, r(unused) at poles
C
C      do j=1,np
C         u(1,j) = 0.0
C         r(1,j) = 0.0
C         u(nt,j) = 0.0
C         r(nt,j) = 0.0
C      end do
C
C      print input parameters
C
C      write(*,100)
100 format(//' mud2sa test' )

      write (*,101) (iparm(i),i=1,15)
101 format(/' integer input parameters ',
      +' intl = ',i2,' nta = ',i2,' ntb = ',i2,' npc =
',i2,' npd = ',i2,
      +' itp = ',i2,' jpq = ',i2,' iet = ',i2,' jep =
',i2
      +' nt = ',i3,' np = ',i3,' iguess = ',i2,' maxcy =
',i2,
      +' method = ',i2, ' work space estimate = ',i7)

      write (*,102) (mgopt(i),i=1,4)
102 format(/' multigrid option parameters ',
      +' kcycle = ',i2,
      +' iprер = ',i2,
      +' ipost = ',i2
      +' interpol = ',i2)

      write(*,103) (fparm(i),i=1,5)
103 format(/' floating point input parameters ',
      +' ta = 'f6.3,' tb = 'f6.3,' pc = 'f6.3,' pd =
',f6.3,
      +' tolerance (error control) =   ',e10.3)
C
C      discretization call to mud2sa
C
C      write(*,104) intl
104 format(/' discretization call to mud2sa ', ' intl
= ',i2)
      call

```

```

mud2sa(iparm,fparm,w,sigt,sigp,xlmb,bndc,r,u,mgopt,ierror
r)
      write (*,105) ierror,iparm(16)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)
C
C      aprroximation call to mud2sa
C
      intl = 1
      write(*,106) intl, method, iguess, maxcy, tolmax
106 format(/' approximation call to mud2sa ',
      +' intl = ',i2, ' method = ',i2 , ' iguess = ',i2,
' maxcy = ',i2
      +' tolmax = ',f5.2)
      call
mud2sa(iparm,fparm,w,sigt,sigp,xlmb,bndc,r,u,mgopt,ierror
r)
      write (*,107) ierror
107 format(' ierror = ',i2 )
      if (ierror.gt.0) call exit(0)
      if (ierror .le. 0) then
C
C      compute and print exact maximum error
C
      errm = 0.0
      do j=1,np
      sinp = sinap(j)
      cosp = cosap(j)
      do i=1,nt
          sint = sinat(i)
          cost = cosat(i)
          ue = (sint*cosp*sint*sinp*cost)**2
          errm = amax1(errm,abs((u(i,j)-ue)))
      end do
      end do
      write(*,108) errm
108 format(' maximum error   = ',e10.3 )
      end if
      end

      function sigt(t,p)
C
C      colatitude coefficient (this will be called off

```

```

grid by mud2sa)
c
    implicit none
    real t,p,sigt,sint,cosp
    sint = sin(t)
    cosp = cos(p)
    sigt = sint*(1.5+(sint*cosp)**2)
    return
    end

    function sigp(t,p)
c
c      longitude coefficient (this will be called off
grid by mud2sa)
c
    implicit none
    real t,p,sigp,sint,cosp
    sint = sin(t)
    cosp = cos(p)
c      avoid polar singularities and division by zero
    if (abs(sint) .gt. 1.e-7) then
        sigp = (1.5+(sint*cosp)**2)/sint
    else
        sigp = 1.0
    end if
    return
    end

    function xlmb(t,p)
c
c      zero order coefficient evaluated only on grid
points
c
    implicit none
    real t,p,xlmb
    integer i,j
    integer
    intl,nta,ntb,npc,npd,itp,jpq,iet,jep,nt,np,
    +
    iguess,maxcy,method,nwork,lwork,iter
    real
    ta,tb,pc,fd,tolmax,sinat,cosat,sinap,cosap,dlt,dlp
    common / iprm /
    intl,nta,ntb,npc,npd,itp,jpq,iet,jep,nt,np,

```

```

+
iguess,maxcy,method,nwork,lwork,iter
      common / fprm /
ta,tb,pc,pd,tolmax,sinat(81),cosat(81),
+           sinap(161),cosap(161),dlt,dlp
      i = int((t-ta)/dlt+0.5)+1
      j = int((p-pc)/dlp+0.5)+1
      xlmb = sinat(i)*(1.5 + (sinat(i)*cosap(j))**2)
      return
end

subroutine bndc(kbdy,torp,alfa,gbdy)
C
C      this subroutine must be provided as a dummy
argument even though
C      there are no mixed derivative b.c.
C
      implicit none
      integer kbdy
      real torp,alfa,gbdy
      return
end

```

## TMUD2SP

```

C
C      file tmud2sp.f
C
C      * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *                               copyright (c) 2008 by UCAR
*
C      *
*
C      *                               University Corporation for Atmospheric

```

Research \*  
C \*  
\*  
C \* all rights reserved  
\*  
C \*  
\*  
C \* MUDPACK version 5.0.1  
\*  
C \*  
\*  
C \* A Fortran Package of Multigrid  
\*  
C \*  
\*  
C \* Subroutines and Example Programs  
\*  
C \*  
\*  
C \* for Solving Elliptic Partial Differential  
Equations \*  
C \*  
\*  
C \* by  
\*  
C \*  
\*  
C \* John Adams  
\*  
C \*  
\*  
C \* of  
\*  
C \*  
\*  
C \*  
\*  
C \* the National Center for Atmospheric  
Research \*  
C \*  
\*  
C \* Boulder, Colorado (80307)  
U.S.A. \*  
C \*  
\*  
C \* which is sponsored by

```
*  
C      *  
*  
C      *          the National Science Foundation  
*  
C      *  
*  
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C  
C ... purpose  
C  
C      test program for the mudpack solver mud2sp  
C  
C ... required mudpack files  
C  
C      mud2sp.f, mudcom.f  
C  
C  
*****  
*  
C  
*****  
*  
C  
*****  
*  
C  
C      sample program/test driver for mud2sp  
C  
C  
*****  
**  
C  
*****  
**  
C  
C  
C      a sample program/test driver for mud2sp is listed  
below. it  
C      can be executed as an initial test. the output is  
listed for  
C      listed for the test case described.  
C  
C      test the driver below by solving the separable  
elliptic pde  
C
```

```

c      (1.+x**2)*pxx + exp(1.-y)*(pyy-py) - (x+y)*pe =
r(x,y)
c
c      on the unit square with specified boundary
conditions at
c      xb = 1.0, yc = 0.0 and mixed boundary conditions
c
c      dp/dx - pe(0.0,y) = ga(y)   (at x = 0.0)
c
c      dp/dy + pe(x,1.0) = gd(x)   (at y = 1.0)
c
c      use point relaxation and choose a grid as close to
60 x 50
c      as the grid size constraints allow.  use the exact
solution
c
c      pe(x,y) = (x**3+y**3+1.0)/3
c
c      for testing.
c
c
*****
***** output (32 bit floating point arithmetic)
c
*****
***** mud2sp test
c
c      integer input arguments
c      intl = 0 nxa = 2 nxb = 1 nyc = 1 nyd = 2
c      ixp = 2 jyq = 3 iex = 6 jey = 5
c      nx = 65 ny = 49 iguess = 0 maxcy = 1
c      method = 0 work space estimate = 17065
c
c      multigrid option arguments
c      kcycle = 2
c      iprer = 2
c      ipost = 1
c      interpol = 3
c
c      floating point input parameters
c      xa = 0.000 xb = 1.000 yc = 0.000 yd = 1.000

```

```

c      tolerance (error control) =    0.000E+00
c
c      discretization call to mud2sp intl =  0
c      ierror =  0 minimum work space =  13264
c
c      approximation call to mud2sp
c      intl =  1 method =  0 iguess =  0
c      ierror =  0
c      maximum error =  0.520E-04
c
c
c ****
c ****
c      end of output
c
c ****
c ****
c
c      program tmud2sp
c      implicit none
c      integer iixp,jjyq,iiex,jjey,nnx,nny,llwork
c
c      set grid sizes with parameter statements
c
c      parameter (iixp = 2 , jjyq = 3 , iiex = 6, jjey =
5)
c              parameter (nnx=iixp*2** (iiex-1)+1,
nny=jjyq*2** (jjey-1)+1)
c
c      set work space length approximation for point
relaxation
c      see (mud2sp.d)
c
c      parameter (llwork=5* (nnx*nny+2* (nnx+nny)))
c      real phi(nnx,nny),rhs(nnx,nny),work(llwork)
c
c      put integer and floating point argument names in
contiguous
c      storeage for labelling in vectors iprm,fprm
c
c      integer iprm(16),mgopt(4)
c      real fprm(6)
c      integer
c      intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny,

```

```

+
iguess,maxcy,method,nwork,lwrkqd,itero

common/itmud2sp/intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,
ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero
    real xa,xb,yc,yd,tolmax,relmax
    common/ftmud2sp/xa,xb,yc,yd,tolmax,relmax
    equivalence(intl,iprm)
    equivalence(xa,fprm)
    integer i,j,ierror
    real
dlx,dly,x,y,cxx,cyy,cx,cy,ce,pxx,pyy,px,py,pe,errmax
    real cex,cey
c
c      declare coefficient and boundary condition input
subroutines external
c
c      external cofx,cofy,bndsp
c
c
c      set input integer arguments
c
        intl = 0
c
c      set boundary condition flags
c
nxa = 2
nxb = 1
nyc = 1
nyd = 2
c
c      set grid sizes from parameter statements
c
        ixp = iixp
        jyq = jjyq
        iex = iiex
        jey = jjey
        nx = nnx
        ny = nny
c
c      set multigrid arguments (w(2,1) cycling with fully
weighted

```

```
c      residual restriction and cubic prolongation)
c
c      mgopt(1) = 2
c      mgopt(2) = 2
c      mgopt(3) = 1
c      mgopt(4) = 3
c
c      set for one cycle
c
c      maxcy = 1
c
c      set no initial guess forcing full multigrid
c      cycling
c
c      iguess = 0
c
c      set work space length approximation from parameter
c      statement
c
c      nwork = llwork
c
c      set point relaxation
c
c      method = 0
c
c      set end points of solution rectangle in (x,y)
c      space
c
c      xa = 0.0
c      xb = 1.0
c      yc = 0.0
c      yd = 1.0
c
c      set mesh increments
c
c      dlx = (xb-xa)/float(nx-1)
c      dly = (yd-yc)/float(ny-1)
c
c      set for no error control flag
c
c      tolmax = 0.0
c
c      set right hand side in rhs
c      initialize phi to zero
```

```

c
do i=1,nx
x = xa+float(i-1)*dlx
call cofx(x,cxx,cx,cex)
do j=1,ny
y = yc+float(j-1)*dly
call cofy(y,cyy,cy,cey)
ce = cex+cey
call exact(x,y,pxx,pyy,px,py,pe)
rhs(i,j) = cxx*pxx+cyy*pyy+cx*px+cy*py+ce*pe
phi(i,j) = 0.0
end do
end do
c
c      set specified boundaries in phi
c
x = xb
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,pxx,pyy,px,py,pe)
phi(nx,j) = pe
end do
y = yc
do i=1,nx
x = xa+float(i-1)*dlx
call exact(x,y,pxx,pyy,px,py,pe)
phi(i,1) = pe
end do
write(*,100)
100 format(//' mud2sp test ')
      write (*,101) (iprm(i),i=1,15)
101 format(/' integer input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =
',i2,' nyd = ',i2,
+/' ixp = ',i2,' jyq = ',i2,' iex = ',i2,' jey =
',i2
+/' nx = ',i3,' ny = ',i3,' iguess = ',i2,' maxcy =
',i2,
+/' method = ',i2, ' work space estimate = ',i7)
      write (*,102) (mgopt(i),i=1,4)
102 format(/' multigrid option arguments ',
+/' kcycle = ',i2,
+/' iprer = ',i2,
+/' ipost = ',i2

```

```

+/' intpol = ',i2)
      write(*,103) xa,xb,yc,yd,tolmax
103 format(/' floating point input parameters ',
+/' xa = ',f6.3,' xb = ',f6.3,' yc = ',f6.3,' yd =
',f6.3,
+/' tolerance (error control) =    ',e10.3)
C
C      initialization call
C
      write(*,104) intl
104 format(/' discretization call to mud2sp', ' intl =
', i2)
      call
mud2sp(iprm,fprm,work,cofx,cofy,bndsp,rhs,phi,mgopt,ierr
or)
C
C      print error parameter and minimum work space
requirement
C
      write (*,200) ierror,iprm(16)
200 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)
C
C      attempt solution
C
      intl = 1
      write(*,106) intl,method,iguess
106 format(/' approximation call to mud2sp',
+/' intl = ',i2, ' method = ',i2,' iguess = ',i2)
      call
mud2sp(iprm,fprm,work,cofx,cofy,bndsp,rhs,phi,mgopt,ierr
or)
      write (*,107) ierror
107 format(' ierror = ',i2)
      if (ierror.gt.0) call exit(0)
      if (ierror .le. 0) then
C
C      compute and print maximum norm of error
C
      errmax = 0.0
      do j=1,ny
y = yc+(j-1)*dly
      do i=1,nx

```

```

x = xa+(i-1)*dlx
call exact(x,y,pxx,pyy,px,py,pe)
errmax = amax1(errmax,abs((phi(i,j)-pe)))
end do
end do
write(*,201) errmax
201 format(' maximum error = ',e10.3)
end if
end

subroutine cofx(x,cxx,cx,cex)
c
c      input x dependent coefficients
c
implicit none
real x,cxx,cx,cex
cxx = 1.0+x*x
cx = 0.0
cex = -x
return
end

subroutine cofy(y,cyy,cy,cey)
c
c      input y dependent coefficients
c
implicit none
real y,cyy,cy,cey
cyy = exp(1.0-y)
cy = -cyy
cey = -y
return
end

subroutine bndsp(kbdy,xory,alfa,gbdy)
c
c      input mixed derivative b.c. to mud2sp
c
implicit none
integer kbdy
real xory,alfa,gbdy,x,y,pe,px,py,pxx,pyy
real xa,xb,yc,yd,tolmax,relmax
common/ftmud2sp/xa,xb,yc,yd,tolmax,relmax
if (kbdy.eq.1) then ! x=xa boundary

```

```

y = xory
x = xa
call exact(x,y,pxx,pyy,px,py,pe)
alfa = -1.0
gbdy = px + alfa*pe
return
end if
if (kbdf.eq.4) then ! y=yd boundary
y = yd
x = xory
call exact(x,y,pxx,pyy,px,py,pe)
alfa = 1.0
gbdy = py + alfa*pe
return
end if
end

subroutine exact(x,y,pxx,pyy,px,py,pe)
c
c      set an exact solution for testing mud2sp
c
implicit none
real x,y,pxx,pyy,px,py,pe
pe = (x**3+y**3+1.0)/3.0
px = x*x
py = y*y
pxx = x+x
pyy = y+y
return
end

```

TMUD3

```
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *               University Corporation for Atmospheric
Research          *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK version 5.0.1
*
C      *
*
C      *               A Fortran Package of Multigrid
*
C      *
*
C      *               Subroutines and Example Programs
*
C      *
*
C      *               for Solving Elliptic Partial Differential
Equations        *
C      *
*
C      *               by
*
C      *
*
C      *               John Adams
*
C      *
*
C      *               of
*
C      *
*
C      *               the National Center for Atmospheric
Research          *
```

```
C      *
*
C      *                      Boulder, Colorado (80307)
U.S.A.      *
C      *
*
C      *                      which is sponsored by
*
C      *
*
C      *                      the National Science Foundation
*
C      *
*
C      *      *      *      *      *      *      *      *      *      *      *
*      *      *      *      *      *      *      *      *      *      *      *
C
C ... purpose
C
C      test program for the MUDPACK solver mud3
C
C ... required MUDPACK files
C
C      mud3.f, mudcom.f, mud3ln.f, mud3pn.f
C
C
C
*****
*
C
*****
*
C
*****
**
C
*****
**
C
C      sample program/test driver for mud3
C
C
*****
**
C
*****
**
C
C      a sample program/test driver for mud3 is below. it
```

```

can be
c      executed as an initial test.  the output is listed
for the
c      test case described.
c
c      test the driver below by solving the nonseparable
3-d elliptic pde
c
c          pxx+ppy+pzz-y*z*px-x*z*py-x*y*pz-x*y*z*pe =
r(x,y,z)
c
c      on a 33 by 25 by 65 grid superimposed on the
(x,y,z) region
c
c          [0.5,1.0] X [1.0,2.0] X [0.25,0.75]
c
c      the solution is specified at x=0.5,1.0 and
y=1.0,2.0 and there are
c      mixed derivative conditions of the form
c
c          dp/dz - (x*y)*p = g(x,z) at z=0.25
c
c          dp/dz + (x*y)*p = h(x,y) at z=0.75
c
c      one full multigrid cycle using the default
multigrid options
c      and line relaxation in the z direction is
executed.  The
c      exact solution
c
c          pe(x,y,z) = exp(x*y*z)
c
c      is used to set the right hand side of the pde,
boundary conditions
c      and compute error.
c
c ****
c      output (32 bit floating point arithmetic)
c
c ****
c
c      mud3 test
c
c      input arguments

```

```

c      intl = 0 nxa = 1 nxb = 1 nyc = 1 nyd = 1
c      nze = 2 nzf = 2
c      ixp = 2 jjyq = 3 kkzr = 2
c      iex = 5 jjey = 4 kkez = 6
c      nx = 33 ny = 25 nz = 65 iguess = 0 maxcy = 1
c      method = 3 work space length input = 949725
c      xa = 0.50 xb = 1.00
c      yc = 1.00 yd = 2.00
c      ze = 0.25 zf = 0.75
c      tolmax = 0.000E+00
c
c      multigrid options
c      kcycle = 2
c      iprer = 2
c      ipost = 1
c      interpol = 3
c
c      discretization call to mud3 intl = 0
c      ierror = 0 minimum work space = 824224
c
c      approximation call to mud3
c      intl = 1 method = 3 iguess = 0 maxcy = 1
c      ierror = 0
c      maximum error = 0.117E-04
c
c
c ****
c ****
c      end of output
c
c ****
c ****
c
c      program tmud3
c      implicit none
c
c      set grid sizes with parameter statements
c
c      integer
iixp,jjyq,kkzr,iex,jjey,kkez,llwork,nnx,nny,nnz
parameter(iixp=2,jjyq=3,kkzr=2)
parameter(iex=5,jjey=4,kkez=6)
parameter (nnx = iixp*2** (iex-1)+1)
parameter (nny = jjyq*2** (jjey-1)+1)

```

```

      parameter (nnz = kkzr*2** (kkez-1)+1)
C
C      set work space length estimate for method=3 (see
mud3.d).
C      this will probably overestimate required space
C
      parameter (llwork = 15* (nnx+2) * (nny+2) * (nnz+2) )
C
C
C      dimension solution,right hand side, and work
arrays
C
      real
phi (nnx,nny,nnz),rhs (nnx,nny,nnz),work (llwork)
      integer iprm(23),mgopt(4)
      real fprm(8)
C
C      put integer and floating point arguments names in
contiguous
C      storeage labelling
C
      integer
intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex,jey,
      +
kez,nx,ny,nz,iguess,maxcy,method,meth2,nwork,
      +
lwrkqd,itero

common/itmud3/intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,i
ex,jey,
      +
kez,nx,ny,nz,iguess,maxcy,method,meth2,nwork,
      +
lwrkqd,itero

      real xa,xb,yc,yd,ze,zf,tolmax,relmax
      common/ftmud3/xa,xb,yc,yd,ze,zf,tolmax,relmax
      real
dlx,dly,dlz,x,y,z,cxx,cyy,czz,cx,cy,cz,ce,errm
      real pxx,pyy,pzz,px,py,pz,pe
      integer i,j,k,ierror
      equivalence(intl,iprm)
      equivalence(xa,fprm)
C
C      declare coefficient and boundary condition input
subroutines external

```

```
C
C      external cof,bndc
C
C      set for initial call
C
C      intl = 0
C
C      set boundary condition flags
C
C      nxa = 1
C      nxb = 1
C      nyc = 1
C      nyd = 1
C      nze = 2
C      nzf = 2
C
C      set grid sizes from parameter statements
C
C      ixp = iixp
C      jyq = jjyq
C      kzs = kkzs
C      iex = iiex
C      jey = jjey
C      kez = kkez
C      nx = nnx
C      ny = nny
C      nz = nnz
C
C      set for one multigrid cycles
C
C      maxcy = 1
C
C      set work space length approximation from parameter
C      statement
C
C      nwork = llwork
C
C      set method of relaxation--line in the z direction
C
C      method = 3
C
C      meth2 only used in planar relaxation--but set
C
C      meth2 = 0
```

```

c
c      set full multigrid cycling by flagging no initial
guess at the finest
c      grid level
c
c      iguess = 0
c
c      set end points of solution region in (x,y,z) space
c
xa = 0.5
xb = 1.0
yc = 1.0
yd = 2.0
ze = 0.25
zf = 0.75
c
c      set default multigrid options
c
mgopt(1) = 2
mgopt(2) = 2
mgopt(3) = 1
mgopt(4) = 3
c
c      set mesh increments
c
dlx = (xb-xa)/float(nx-1)
dly = (yd-yc)/float(ny-1)
dlz = (zf-ze)/float(nz-1)
c
c      set for no error control
c
tolmax = 0.0
c
c      set right hand side in rhs and phi to zero
c
do k=1,nz
z = ze+(k-1)*dlz
do j=1,ny
y = yc+float(j-1)*dly
do i=1,nx
x = xa+float(i-1)*dlx
call cof(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
rhs(i,j,k) =

```

```

cxxx*pxx+cyy*pyy+czz*pzz+cx*px+cy*py+cz*pz+ce*pe
    phi(i,j,k) = 0.0
    end do
end do
    end do
C
C      set specified values at x and y boundaries in phi
C
x = xa
do k=1,nz
z = ze+(k-1)*dlz
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(1,j,k) = pe
end do
    end do
x = xb
do k=1,nz
z = ze+(k-1)*dlz
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(nx,j,k) = pe
end do
    end do
y = yc
do k=1,nz
z = ze+(k-1)*dlz
do i=1,nx
x = xa+float(i-1)*dlx
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(i,1,k) = pe
end do
    end do
y = yd
do k=1,nz
z = ze+(k-1)*dlz
do i=1,nx
x = xa+float(i-1)*dlx
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(i,ny,k) = pe
end do
    end do

```

```

        write(6,50)
50 format(//' mud3 test ')
C
C      print input arguments
C

write(6,100)intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex
,jey,kez,
+
nx,ny,nz,iguess,maxcy,method,nwork,xa,xb,yc,yd,ze,zf,
+
tolmax,mgopt(1),mgopt(2),mgopt(3),mgopt(4)
100 format(/' input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =
',i2,' nyd = 'i2,
+/' nze = ',i2, ' nzf = ',i2,
+/' ixp = ',i2,' jyq = 'i2,' kzr = 'i2,
+/' iex = ',i2, ' jey = 'i2, ' kez = 'i2,
+/' nx = 'i3,' ny = 'i3,' nz = 'i3, ' iguess =
'i2,' maxcy = 'i2,
+/' method = 'i2, ' work space length input = ',i7,
+/' xa = 'f5.2,' xb = 'f5.2,
+/' yc = 'f5.2,' yd = 'f5.2,
+/' ze = 'f5.2,' zf = 'f5.2,
+/' tolmax = ' ,e10.3
+/' multigrid options '
+/' kcycle = ',i2
+/' iprer = ',i2
+/' ipost = ',i2
+/' interpol = ',i2 )

C
C      discretize pde
C

        write(*,104) intl
104 format(/' discretization call to mud3', ' intl =
', i2)
        call
mud3(iprm,fprm,work,cof,bndc,rhs,phi,mgopt,ierror)
        write (*,105) ierror,iprm(22)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
        if (ierror.gt.0) call exit(0)
C
C      approximate pde

```

```

c
    intl = 1
    write(*,106) intl,method,iguess,maxcy
106 format(/' approximation call to mud3 ',
     +' intl = ',i2, ' method = ',i2,' iguess = ',i2, '
maxcy = ',i2)
     call
mud3(iprm,fprm,work,cof,bndc,rhs,phi,mgopt,ierror)
     write (*,107) ierror
107 format(' ierror = ',i2)
     if (ierror.gt.0) call exit(0)

c
c      compute and print maximum error
c
     call error(nx,ny,nz,phi,errm)
     write(*,108) errm
108 format(' maximum error = ',e10.3)
end

subroutine error(nx,ny,nz,phi,errm)
c
c      compute the error in the estimate in phi
c
implicit none
integer nx,ny,nz
real phi(nx,ny,nz),errm
real xa,xb,yc,yd,ze,zf,tolmax,relmax
common/ftmud3/xa,xb,yc,yd,ze,zf,tolmax,relmax
real dlx,dly,dlz,x,y,z,pxx,pyy,pzz,px,py,pz,pe
integer i,j,k
dlx = (xb-xa) / (nx-1)
dly = (yd-yc) / (ny-1)
dlz = (zf-ze) / (nz-1)
errm = 0.0
do k=1,nz
  z = ze+(k-1)*dlz
  do j=1,ny
    y = yc+float(j-1)*dly
    do i=1,nx
      x = xa+float(i-1)*dlx
      call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
      errm = amax1(errm,abs(phi(i,j,k)-pe))
    end do
  end do
end do

```

```

    end do
    return
end

subroutine cof(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
c
c      input pde coefficients at grid point (x,y,z) in
the solution region
c      to subroutine mud3
c
implicit none
real x,y,z,cxx,cyy,czz,cx,cy,cz,ce
cxx = 1.0
cyy = 1.0
czz = 1.0
cx = -y*z
cy = -x*z
cz = -x*y
ce = -(x*y*z)
return
end

subroutine bndc(kbdy,xory,yorz,alfa,gbdy)
c
c      input mixed derivative conditions at z boundaries
to mud3
c
implicit none
integer kbdy
real xory,yorz,alfa,gbdy
real xa,xb,yc,yd,ze,zf,tolmax,relmax
common/ftmud3/xa,xb,yc,yd,ze,zf,tolmax,relmax
real x,y,z,pxx,pyy,pzz,px,py,pz,pe
if (kbdy.eq.5) then
c
c      z = ze (lower z boundary)
c
z = ze
x = xory
y = yorz
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
alfa = -(x*y)
gbdy = pz + alfa*pe
return

```



```
* * * * * *
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK version 5.0.1
*
C      *
*
C      *               A Fortran Package of Multigrid
*
C      *
*
C      *               Subroutines and Example Programs
*
C      *
*
C      *               for Solving Elliptic Partial Differential
Equations        *
C      *
*
C      *               by
*
C      *
*
C      *               John Adams
*
C      *
*
C      *               of
*
C      *
*
C      *               the National Center for Atmospheric
```

```
Research          *
C      *
*
C      *                      Boulder, Colorado (80307)
U.S.A.          *
C      *
*
C      *                      which is sponsored by
*
C      *
*
C      *                      the National Science Foundation
*
C      *
*
C      *      *      *      *      *      *      *      *      *      *      *
*      *      *      *      *      *      *      *      *      *      *      *
C
C ... purpose
C
C      test program for the MUDPACK solver mud34
C
C ... required MUDPACK files
C
C      mud34.f, mud3.f, mudcom.f, mud3ln.f, mud3pn.f
C
C
C
*****
* 
C
*****
* 
C
*****
* 
C
sample program/test driver for mud34
C
C
*****
** 
C
*****
* 
C
*****
* 
C
a sample program/test driver for mud34 is below.
```

```

it can be
c      executed as an initial test.  the output is listed
for the
c      test case described.
c
c      test the driver below by solving the nonseparable
3-d elliptic pde
c
c          pxx+ppy+pzz-y*z*px-x*z*py-x*y*pz-x*y*z*pe =
r(x,y,z)
c
c      on a 33 by 25 by 65 grid superimposed on the
(x,y,z) region
c
c          [0.5,1.0] X [1.0,2.0] X [0.25,0.75]
c
c      the solution is specified at x=0.5,1.0 and
y=1.0,2.0 and there are
c      mixed derivative conditions of the form
c
c          dp/dz - (x*y)*p = g(x,z) at z=0.25
c
c          dp/dz + (x*y)*p = h(x,y) at z=0.75
c
c      One full multigrid cycle using the default
multigrid options
c      and line relaxation in the z direction is first
executed (see
c      tmud3.f) with mud3.  Then two additional cycles
are executed
c      with iguess=1 to ensure that second-order
discretization level
c      error is reached (a requirement for fourth-order
mudpack solvers).
c      Finally mud34 is called to produce a fourth-order
estimate.
c      the exact solution
c
c          pe(x,y,z) = exp(x*y*z)
c
c      is used to set the right hand side of the pde,
boundary conditions
c      and compute error.
c

```

```
C
C ****
C      output (64 bit floating point arithmetic)
C
C ****
C
C      mud3 test
C
C
C      input arguments
C      intl = 0 nxa = 1 nxb = 1 nyc = 1 nyd = 1
C      nze = 2 nzf = 2
C      ixp = 2 jyq = 3 kxr = 2
C      iex = 5 jey = 4 kez = 6
C      nx = 33 ny = 25 nz = 65 iguess = 0 maxcy = 1
C      method = 3 work space length input = 824224
C      xa = 0.50 xb = 1.00
C      yc = 1.00 yd = 2.00
C      ze = 0.25 zf = 0.75
C      tolmax = 0.000E+00
C
C      multigrid options
C      kcycle = 2
C      iprer = 2
C      ipost = 1
C      interpol = 3
C
C      discretization call to mud3 intl = 0
C      ierror = 0 minimum work space = 824224
C
C      approximation call to mud3
C      intl = 1 method = 3 iguess = 0 maxcy = 1
C      ierror = 0
C      maximum error = 0.146E-04
C
C      approximation call to mud3
C      intl = 1 method = 3 iguess = 1 maxcy = 2
C      ierror = 0
C      maximum error = 0.146E-04
C
C      mud34 test ierror = 0
C      maximum error = 0.373E-06
C
C
C ****
```

```

*****
C      end of output
C
***** *****
*****
C
      program tmud34
      implicit none
C
C      set grid sizes with parameter statements
C
      integer
iixp,jjyq,kkzr,iiex,jjey,kkez,llwork,nnx,nny,nnz
      parameter(iixp=2,jjyq=3,kkzr=2)
      parameter(iiex=5,jjey=4,kkez=6)
      parameter (nnx = iixp*2** (iiex-1)+1)
      parameter (nny = jjyq*2** (jjey-1)+1)
      parameter (nnz = kkzr*2** (kkez-1)+1)
C
C      set exact minimal work space (see tmud3.f)
C
      parameter (llwork = 824224 )
C
C      dimension solution,right hand side, and work
arrays
C
      real
phi(nnx,nny,nnz),rhs(nnx,nny,nnz),work(llwork)
      integer iprm(23),mgopt(4)
      real fprm(8)
C
C      put integer and floating point arguments names in
contiguous
C      storeage for labelling purposes
C
      integer
intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex,jey,
+
kez,nx,ny,nz,iguess,maxcy,method,meth2,nwork,
+
lwrkqd,itero

common/itmud3/intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,i
ex,jey,
+

```

```

kez,nx,ny,nz,iguess,maxcy,method,meth2,nwork,
+                      lwrkqd,itero
      real xa,xb,yc,yd,ze,zf,tolmax,relmax
      common/ftmud3/xa,xb,yc,yd,ze,zf,tolmax,relmax
      real
dlx,dly,dlz,x,y,z,cxx,cyy,czz,cx,cy,cz,ce,errm
      real pxx,pyy,pzz,px,py,pz,pe
      integer i,j,k,ierror
      equivalence(intl,iprm)
      equivalence(xa,fprm)

C
C      declare coefficient and boundary condition input
subroutines external
C
      external cof,bndc
C
C      set for initial call
C
      intl = 0
C
C      set boundary condition flags
C
      nxa = 1
      nxb = 1
      nyc = 1
      nyd = 1
      nze = 2
      nzf = 2
C
C      set grid sizes from parameter statements
C
      ixp = iixp
      jyq = jjyq
      kzs = kkzs
      iex = iiex
      jey = jjey
      kez = kkez
      nx = nnx
      ny = nny
      nz = nnz
C
C      set for one multigrid cycles
C
      maxcy = 1

```

```
C
C      set work space length approximation from parameter
statement
C
nwork = llwork
C
C      set method of relaxation--line in the z direction
C
method = 3
C
C      meth2 only used in planar relaxation--but set
C
meth2 = 0
C
C      set full multigrid cycling by flagging no initial
guess at the finest
C      grid level
C
iguess = 0
C
C      set end points of solution region in (x,y,z) space
C
xa = 0.5
xb = 1.0
yc = 1.0
yd = 2.0
ze = 0.25
zf = 0.75
C
C      set default multigrid options
C
mgopt(1) = 2
mgopt(2) = 2
mgopt(3) = 1
mgopt(4) = 3
C
C      set mesh increments
C
dlx = (xb-xa)/float(nx-1)
dly = (yd-yc)/float(ny-1)
dlz = (zf-ze)/float(nz-1)
C
C      set for no error control
C
```

```

tolmax = 0.0
c
c      set right hand side in rhs and phi to zero
c
do k=1,nz
z = ze+(k-1)*dlz
do j=1,ny
y = yc+float(j-1)*dly
do i=1,nx
x = xa+float(i-1)*dlx
call cof(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
rhs(i,j,k) =
cxx*pxx+cyy*pyy+czz*pzz+cx*px+cy*py+cz*pz+ce*pe
phi(i,j,k) = 0.0
end do
end do
end do
c
c      set specified values at x and y boundaries in phi
c
x = xa
do k=1,nz
z = ze+(k-1)*dlz
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(1,j,k) = pe
end do
end do
x = xb
do k=1,nz
z = ze+(k-1)*dlz
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(nx,j,k) = pe
end do
end do
y = yc
do k=1,nz
z = ze+(k-1)*dlz
do i=1,nx
x = xa+float(i-1)*dlx

```

```

call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(i,1,k) = pe
end do
    end do
    y = yd
    do k=1,nz
z = zet+(k-1)*dlz
do i=1,nx
    x = xa+float(i-1)*dlx
    call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
    phi(i,ny,k) = pe
end do
    end do
    write(6,50)
50 format(//' mud3 test ')
C
C      print input arguments
C

write(6,100)intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex
,jey,kez,
+
nx,ny,nz,iguess,maxcy,method,nwork,xa,xb,yc,yd,ze,zf,
+
tolmax,mgopt(1),mgopt(2),mgopt(3),mgopt(4)
100 format(/' input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =
',i2,' nyd = 'i2,
+/' nze = ',i2, ' nzf = ',i2,
+/' ixp = ',i2,' jyq = 'i2,' kzr = 'i2,
+/' iex = ',i2, ' jey = 'i2, ' kez = 'i2,
+/' nx = 'i3,' ny = 'i3,' nz = 'i3, ' iguess =
'i2,' maxcy = 'i2,
+/' method = 'i2, ' work space length input = ',i7,
+/' xa = 'f5.2,' xb = 'f5.2,
+/' yc = 'f5.2,' yd = 'f5.2,
+/' ze = 'f5.2,' zf = 'f5.2,
+/' tolmax = ' ,e10.3
+/' multigrid options '
+/' kcyle = ',i2
+/' iprer = ',i2
+/' ipost = ',i2
+/' interpol = ',i2 )
C

```

```

c      discretize pde
c
c      write(*,104) intl
104 format(/' discretization call to mud3', ' intl =
', i2)
      call
mud3(iprm,fprm,work,cof,bndc,rhs,phi,mgopt,ierror)
      write (*,105) ierror,iprm(22)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)
c
c      approximate pde
c
      intl = 1
      write(*,106) intl,method,iguess,maxcy
106 format(/' approximation call to mud3 ',
+/' intl = ',i2, ' method = ',i2,' iguess = ',i2,
maxcy = ',i2)
      call
mud3(iprm,fprm,work,cof,bndc,rhs,phi,mgopt,ierror)
      write (*,107) ierror
107 format(' ierror = ',i2)
      if (ierror.gt.0) call exit(0)
      call error(nx,ny,nz,phi,errm)
      write(*,108) errm
108 format(' maximum error  =  ',e10.3)
c
c      execute two more cycles using this estimate to
ensure second-order
c
      iguess = 1
      maxcy = 2
      write(*,106) intl,method,iguess,maxcy
      call
mud3(iprm,fprm,work,cof,bndc,rhs,phi,mgopt,ierror)
      write (*,107) ierror
      if (ierror.gt.0) call exit(0)
      call error(nx,ny,nz,phi,errm)
      write(*,108) errm
c
c      attempt fourth-order approximation
c
      call mud34(work,phi,ierror)

```

```

        write (*,109) ierror
109 format(/' mud34 test ', ' ierror = ',i2)
        if (ierror.gt.0) call exit(0)
        call error(nx,ny,nz,phi,errm)
        write(*,108) errm
        end

        subroutine error(nx,ny,nz,phi,errm)
c
c      compute the maximum error in the estimate in phi
c
        implicit none
        integer nx,ny,nz
        real phi(nx,ny,nz),errm
        real xa,xb,yc,yd,ze,zf,tolmax,relmax
        common/ftmud3/xa,xb,yc,yd,ze,zf,tolmax,relmax
        real dlx,dly,dlz,x,y,z,pxx,pyy,pzz,px,py,pz,pe
        integer i,j,k
        dlx = (xb-xa) / (nx-1)
        dly = (yd-yc) / (ny-1)
        dlz = (zf-ze) / (nz-1)
        errm = 0.0
        do k=1,nz
        z = ze+(k-1)*dlz
        do j=1,ny
        y = yc+float(j-1)*dly
        do i=1,nx
        x = xa+float(i-1)*dlx
        call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
        errm = amax1(errm,abs(phi(i,j,k)-pe))
        end do
        end do
        end do
        return
        end

        subroutine cof(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
c
c      input pde coefficients at grid point (x,y,z) in
c      the solution region
c      to subroutine mud3
c
        implicit none
        real x,y,z,cxx,cyy,czz,cx,cy,cz,ce

```

```

cxxx = 1.0
cyyy = 1.0
czzz = 1.0
cx = -y*z
cy = -x*z
cz = -x*y
ce = -(x*y*z)
return
end

subroutine bndc(kbdy,xory,yorz,alfa,gbdy)
c
c      input mixed derivative conditions at z boundaries
to mud3
c
implicit none
integer kbdy
real xory,yorz,alfa,gbdy
real xa,xb,yc,yd,ze,zf,tolmax,relmax
common/ftmud3/xa,xb,yc,yd,ze,zf,tolmax,relmax
real x,y,z,pxx,pyy,pzz,px,py,pz,pe
if (kbdy.eq.5) then
c
c      z = ze (lower z boundary)
c
z = ze
x = xory
y = yorz
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
alfa = -(x*y)
gbdy = pz + alfa*pe
return
end if
if (kbdy.eq.6) then
c
c      z=zf (upper z boundary)
c
z = zf
x = xory
y = yorz
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
alfa = (x*y)
gbdy = pz + alfa*pe
return

```

```

    end if
    end

    subroutine exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
C
C      this subroutine sets an exact solution for testing
mud34
C
      implicit none
      real x,y,z,pe,px,py,pz,pxx,pyy,pzz
      pe = exp(x*y*z)
      px = y*z*pe
      py = x*z*pe
      pz = x*y*pe
      pxx = (y*z)*px
      pyy = (x*z)*py
      pzz = (x*y)*pz
      return
      end

```

## TMUD34SP

```

C
C      file tmud34sp.f
C
C      * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *                               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *                               all rights reserved

```

\*  
C \*  
\*  
C \* MUDPACK version 5.0.1  
\*  
C \*  
\*  
C \* A Fortran Package of Multigrid  
\*  
C \*  
\*  
C \* Subroutines and Example Programs  
\*  
C \*  
\*  
C \* for Solving Elliptic Partial Differential  
Equations \*  
C \*  
\*  
C \* by  
\*  
C \*  
\*  
C \* John Adams  
\*  
C \*  
\*  
C \* of  
\*  
C \*  
\*  
C \* the National Center for Atmospheric  
Research \*  
C \*  
\*  
C \* Boulder, Colorado (80307)  
U.S.A. \*  
C \*  
\*  
C \* which is sponsored by  
\*  
C \*  
\*  
C \* the National Science Foundation

```

*
C      *
*
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*
C      ... purpose
C
C      test program for the MUDPACK solver mud34sp
C
C      ... required MUDPACK files
C
C      mud34sp.f, mud3sp.f, mudcom.f
C
C
C
*****
*
C
*****
*
C
C      sample program/test driver for mud34sp
C
C
*****
**
C
*****
**
C
C      a sample program/test driver for mud3sp is below.
it can be
c      executed as an initial test. the output is listed
for the
c      test case described.
C
c      test the driver below by solving the separable
elliptic pde
C
c      d(x**2*dp/dx) /dx + d(y**2*dp/dy) /dy +
d(z**2*dp/dz) /dz
C
c      - (x+y+z) *pe (x, y, z) = r (x, y, z)

```

```

c
c
c      on the cube [0.5,1.0]**3 with specified boundary
conditions at x,y,z=1.0
c      and derivative boundary conditions of the form
c
c          dp/dx - pe(0.5,y,z) = f(0.5,y,z) at x = 0.5
c
c          dp/dy - pe(x,0.5,z) = g(x,0.5,z) at y = 0.5
c
c          dp/dz - pe(x,y,0.5) = h(x,y,0.5) at z = 0.5
c
c      generate the approximation on a  65 x 49 x 41 grid
using the exact
c      solution
c
c          pe(x,y,z) = (x*y*z)**4
c
c      for testing. First mud3sp is called to produce a
second-order estimate.
c      Then mud34sp is called for the fourth-order
approximation
c
c ****
c      output (64 bit floating point arithmetic)
c
*****mud3sp test
c
c      input arguments
c      intl = 0 nxa = 2 nxb = 1 nyc = 2 nyd = 1
c      nze = 2 nzf = 1
c      ixp = 2 jyq = 3 kzs = 5
c      iex = 6 jey = 5 kez = 4
c      nx = 65 ny = 49 nz = 41 iguess = 0 maxcy = 3
c      method = 0 work space length input = 514258
c      xa = 0.50 xb = 1.00
c      yc = 0.50 yd = 1.00
c      ze = 0.50 zf = 1.00
c      tolmax = 0.000E+00
c
c      multigrid options
c      kcyle = 2

```

```

C      iprer =  2
C      ipost =  1
C      interpol = 3
C
C      discretization call to mud3sp intl =  0
C      ierror =  0 minimum work space =  472557
C
C      approximation call to mud3sp
C      intl =  1 method =  0 iguess =  0 maxcy =  3
C      ierror =  0
C      maximum error =  0.741E-05
C
C      mud34sp test ierror =  0
C      maximum error =  0.207E-06
C
C
C ****
C ****
C      end of output
C
C ****
C ****
C
C      program tmud34sp
C
C      set grid sizes with parameter statements
C
C      implicit none
C
C      set grid sizes with parameter statements
C
C      integer
iixp,jjyq,kkzr,iiex,jjey,kkez,llwork,nnx,nny,nnz
      parameter(iixp=2,jjyq=3,kkzr=5)
      parameter(iiex=6,jjey=5,kkez=4)
      parameter (nnx = iixp*2** (iiex-1)+1)
      parameter (nny = jjyq*2** (jjey-1)+1)
      parameter (nnz = kkzr*2** (kkez-1)+1)
C
C      set work space length approximation (see mud3sp.d)
C
C      parameter(llwork = 7* (nnx+2) * (nny+2) * (nnz+2) /2 )
C

```

```

c
c      dimension solution, right hand side, and work
arrays
c
real
phi(nnx,nny,nnz),rhs(nnx,nny,nnz),work(llwork)
integer iprm(22),mgopt(4)
real fprm(8)
c
c      put integer and floating point arguments names in
contiguous
c      storeage for labelling
c
integer
intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex,jey,
+
kez,nx,ny,nz,iguess,maxcy,method,nwork,lwrkqd,itero
common/itmusp3/intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,i
ex,jey,
+
kez,nx,ny,nz,iguess,maxcy,method,nwork,lwrkqd,itero
real xa,xb,yc,yd,ze,zf,tolmax,relmax
common/ftmsp3/xa,xb,yc,yd,ze,zf,tolmax,relmax
real
dlx,dly,dlz,x,y,z,cxx,cyy,czz,cx,cy,cz,cex,cey,cez,ce,er
rm
real pxx,pyy,pzz,px,py,pz,pe
integer i,j,k,ierror
equivalence(intl,iprm)
equivalence(xa,fprm)
c
c      declare coefficient and boundary condition input
subroutines external
c
external cfx,cfy,cfz,bndc
c
c
c      set input integer parameters
c
intl = 0
c
c      set boundary condition flags
c

```

```

nxa = 2
nxb = 1
nyc = 2
nyd = 1
nze = 2
nzf = 1

c
c      set grid sizes from parameter statements
c
    ixp = iixp
    jyq = jjyq
    kzs = kkzs
    iex = iiex
    jey = jjey
    kez = kkez
    nx = nnx
    ny = nny
    nz = nnz

c
c      set work space length approximation from parameter
statement
c
    nwork = llwork

c
c      set method of relaxation--point gauss/seidel only
for mud3sp
c
    method = 0

c
c      set default multigrid options
c
    mgopt(1) = 2
    mgopt(2) = 2
    mgopt(3) = 1
    mgopt(4) = 3

c
c      set full multigrid cycling by flagging no initial
guess
c
    iguess = 0

c
c      set a limit of three w(2,1) cycles from the finest
level

```

```

C
maxcy = 3
C
C      set end points of solution cube in (x,y,z) space
C
xa = 0.5
xb = 1.0
yc = 0.5
yd = 1.0
ze = 0.5
zf = 1.0
C
C      set mesh increments
C
dlx = (xb-xa)/float(nx-1)
dly = (yd-yc)/float(ny-1)
dlz = (zf-ze)/float(nz-1)
C
C      set for no error control (this allows phi to be
equivlaenced with work)
C
tolmax = 0.0
C
C      set right hand side in rhs using exact solution
C      and initialize phi to zero
C
do k=1,nz
z = ze+(k-1)*dlz
call cfz(z,czz,cz,cez)
do j=1,ny
y = yc+float(j-1)*dly
call cfy(y,cyy,cy,cey)
do i=1,nx
x = xa+float(i-1)*dlx
call cfx(x,cxx,cx,cex)
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
ce = cex+cey+cez
rhs(i,j,k) =
cxx*pxx+cyy*pyy+czz*pzz+cx*px+cy*py+cz*pz+ce*pe
phi(i,j,k) = 0.0
end do
end do
end do
C

```

```

c      set specified values at upper boundaries
c
x = xb
do k=1,nz
z = ze+(k-1)*dlz
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(nx,j,k) = pe
end do
end do
y = yd
do k=1,nz
z = ze+(k-1)*dlz
do i=1,nx
x = xa+(i-1)*dlx
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(i,ny,k) = pe
end do
end do
z = zf
do j=1,ny
y = yc+(j-1)*dly
do i=1,nx
x = xa+float(i-1)*dlx
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(i,j,nz) = pe
end do
end do
write(6,50)
50 format(//' mud3sp test ')
c
c      print input arguments
c

write(6,100) intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex
,jey,kez,
+
nx,ny,nz,iguess,maxcy,method,nwork,xa,xb,yc,yd,ze,zf,
+
tolmax,mgopt(1),mgopt(2),mgopt(3),mgopt(4)
100 format(/' input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =
',i2,' nyd = ',i2,

```

```

+/' nze = ',i2, ' nzf = ',i2,
+/' ixp = ',i2,' jyq = 'i2,' kzs = 'i2,
+/' iex = ',i2, ' jey = 'i2, ' kez = 'i2,
+/' nx = 'i3,' ny = 'i3,' nz = 'i3, ' iguess =
'i2,' maxcy = 'i2,
+/' method = 'i2, ' work space length input = ',i7,
+/' xa = 'f5.2,' xb = 'f5.2,
+/' yc = 'f5.2,' yd = 'f5.2,
+/' ze = 'f5.2,' zf = 'f5.2,
+/' tolmax = ',e10.3
+/' multigrid options '
+/' kcyle = ',i2
+/' iprer = ',i2
+/' ipost = ',i2
+/' interpol = ',i2 )

C
C      discretize pde
C
      write(*,104) intl
104 format(' discretization call to mud3sp', ' intl =
', i2)
      call
mud3sp(iprm,fprm,work,cfx,cfy,cfz,bndc,rhs,phi,mgopt,ier
ror)
      write (*,105) ierror,iprm(21)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)
C
C      approximate pde
C
      intl = 1
      write(*,106) intl,method,iguess,maxcy
106 format(' approximation call to mud3sp ',
+/' intl = ',i2, ' method = ',i2,' iguess = ',i2, '
maxcy = ',i2)
      call
mud3sp(iprm,fprm,work,cfx,cfy,cfz,bndc,rhs,phi,mgopt,ier
ror)
      write (*,107) ierror
107 format(' ierror = ',i2)
      if (ierror.gt.0) call exit(0)
C
C      compute and print maximum error

```

```

C
    call error(nx,ny,nz,phi,errm)
    write(*,108) errm
108 format(' maximum error = ',e10.3)
C
C      compute fourth-order approximation
C
    call mud34sp(work,phi,ierror)
    write (*,109) ierror
109 format ('/ mud34sp test ', ' ierror = ',i2)
    if (ierror.gt.0) call exit(0)
    call error(nx,ny,nz,phi,errm)
    write(*,108) errm
    end

        subroutine error(nx,ny,nz,phi,errm)
C
C      compute the error in the estimate in phi
C
        implicit none
        integer nx,ny,nz
        real phi(nx,ny,nz),errm
        real xa,xb,yc,yd,ze,zf,tolmax,relmax
        common/ftmsp3/xa,xb,yc,yd,ze,zf,tolmax,relmax
        real dlx,dly,dlz,x,y,z,pxx,pyy,pzz,px,py,pz,pe
        integer i,j,k
        dlx = (xb-xa) / (nx-1)
        dly = (yd-yc) / (ny-1)
        dlz = (zf-ze) / (nz-1)
        errm = 0.0
        do k=1,nz
            z = ze+(k-1)*dlz
            do j=1,ny
                y = yc+float(j-1)*dly
                do i=1,nx
                    x = xa+float(i-1)*dlx
                    call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
                    errm = amax1(errm,abs(phi(i,j,k)-pe))
                end do
            end do
            end do
            return
        end
C

```

```

c      coefficient subroutines
c
      subroutine cfx(x,cxx,cx,cex)
      implicit none
      real x,cxx,cx,cex
      cxx = x*x
      cx = x+x
      cex = -x
      return
      end

      subroutine cfy(y,cyy,cy,cey)
      implicit none
      real y,cyy,cy,cey
      cyy = y*y
      cy = y+y
      cey = -y
      return
      end

      subroutine cfz(z,czz,cz,cez)
      implicit none
      real z,czz,cz,cez
      czz = z*z
      cz = z+z
      cez = -z
      return
      end

      subroutine bndc(kbdy,xory,yorz,cons,gbdy)
c
c      input derivative boundary conditions to mud3sp
c      at lower x,y,z boundaries
c
      implicit none
      integer kbdy
      real
      xory,yorz,cons,gbdy,x,y,z,pxx,pyy,pzz,px,py,pz,pe
      real xa,xb,yc,yd,ze,zf,tolmax,relmax
      common/ftmsp3/xa,xb,yc,yd,ze,zf,tolmax,relmax
      if (kbdf.eq.1) then
c
c      x=xa surface
c

```

```

x = xa
y = xory
z = yorz
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
cons = -1.0
gbdy = px+cons*pe
return
      end if
      if (kbdy.eq.3) then
C
C      y=yc surface
C
y = yc
x = xory
z = yorz
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
cons = -1.0
gbdy = py+cons*pe
return
      end if
      if (kbdy.eq.5) then
C
C      z=ze surface
C
z = ze
x = xory
y = yorz
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
cons = -1.0
gbdy = pz + cons*pe
return
      end if
      end

subroutine exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
C
C      this subroutine is used to set an exact solution
for testing mud3sp
C      (i.e., setting the rhs, boundary conditions and
computing the exact
C      error)
C
implicit none
real x,y,z,pxx,pyy,pzz,px,py,pz,pe

```

```

pe = (x*y*z)**4
px = 4.* (x*y*z)**3*y*z
py = 4.* (x*y*z)**3*x*z
pz = 4.* (x*y*z)**3*x*y
pxx = 12.* (x*y*z)**2*(y*z)**2
pyy = 12.* (x*y*z)**2*(x*z)**2
pzz = 12.* (x*y*z)**2*(x*y)**2
return
end

```

## TMUD3CR

```

C
C      file tmud3cr.f
C
C      * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *           copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *           all rights reserved
*
C      *
*
C      *           MUDPACK version 5.0.1
*
C      *
*
C      *           A Fortran Package of Multigrid
*
C      *

```

```
*  
C      *          Subroutines and Example Programs  
*  
C      *  
*  
C      *      for Solving Elliptic Partial Differential  
Equations      *  
C      *  
*  
C      *          by  
*  
C      *  
*  
C      *          John Adams  
*  
C      *  
*  
C      *          of  
*  
C      *  
*  
C      *          the National Center for Atmospheric  
Research      *  
C      *  
*  
C      *          Boulder, Colorado (80307)  
U.S.A.      *  
C      *  
*  
C      *          which is sponsored by  
*  
C      *  
*  
C      *          the National Science Foundation  
*  
C      *  
*  
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * *  
C  
C ... purpose  
C  
C     test program for the MUDPACK solver mud3cr  
C
```

```

c ... required MUDPACK files
c
c      mud3cr.f, mudcom.f
c
c
c
*****  

*  

c
*****  

*  

c
*****  

*  

c
c      sample program/test driver for mud3cr (see
mud3cr.d)
c
c
*****  

**  

c
*****  

**  

c
c      a sample program/test driver for mud3cr is below.
it can be
c      executed as an initial test.  The output from
executing
c      the code in this file is listed after the problem
description.
c
c      Problem Description:
c
c      test mud3cr by solving the nonseparable 3-d
elliptic pde
c      with cross derivative terms:
c
c          cxx*pxx + cyy*pyy + czz*pzz + cx*px + cy*py +
cz*pz + ce*pe +
c
c          cxy*pxy + cxz*pxz + cyz*pyz = r(x,y,z) .
c
c      on the region 0 < x < 1, 0 < y < (pi+pi), 0 < z <
1.
c      let s = sin(y).  the coefficients are given by:
c

```

```

c           cxx = 1.0+0.5*s*z, cyy = 1+x*z, czz =
1+0.5*s*x
c
c           cx = 0.0, cy = -x*z, cz = 0.0, ce = -(x+z)
c
c           cxy = sqrt(cxx*cyy), cxz = 0.0, cyz =
sqrt(cyy*czz)
c
c           assume the solution is periodic in y and is
specified at x=0 and z=0.
c           further assume mixed oblique derivative conditions
of the form
c
c           px + (z*(1-z))*py + sin(y)*cos(y)*pz -
p(1,y,z) = g(y,z)
c
c           at x=1 and mixed normal derivative conditions of
the form
c
c           pz + x*s*s*p(x,y,1) = h(x,y)
c
c           at z=1. for testing purposes use the exact
solution
c
c           p(x,y,z) = (x*cos(y)*z)**3
c
c           to set boundary conditions, the right hand side,
and compute
c           exact error. results from approximating this
problem on a
c           25 by 129 by 25 x-y-z grid using mud3cr are given
below. point
c           relaxation and an error tolerance of .001 are
used.
c
c ****
c           output (32 bit floating point arithmetic)
c
c ****
c
c           mud3cr test
c
c           input arguments
c           intl = 0

```

```
c      nxa = 1 nxb = 2
c      nyc = 0 nyd = 0
c      nze = 1 nzf = 2
c      ixp = 3 jyq = 2 kzs = 3
c      iex = 4 jey = 7 kez = 4
c      nx = 25 ny = 129 nz = 25
c      iguess = 0 maxcy = 2
c      method = 0 work space input = 1176431
c      xa = 0.00 xb = 1.00
c      yc = 0.00 yd = 6.28
c      ze = 0.00 zf = 1.00
c      tolmax = 0.000E+00
c
c      multigrid options
c      mgopt(1) = 0
c
c      new mud3cr arguments
c      icrs
c          1    0    1
c      tol = 0.0010
c      maxit = 10
c
c      initial call
c      intl = 0
c      iguess = 0
c      ierror = 0
c      minimum required work space length = 1176431
c
c      approximation call
c      intl = 1
c      iguess = 0
c      ierror = 0
c      number of outer iterations executed = 8
c      relative difference profile:
c      0.2694 0.0853 0.0357 0.0135 0.0075 0.0031
c      0.0016 0.0008
c      exact least squares error = 0.228E-03
c
c
c ****
c      end of output
c
*****
```

```

*****
C
      program tmd3cr
      implicit none
C
C      set grid sizes and predetermined minimal required
C      equired work
C      with parameter statements
C
      integer
iixp,jjyq,kkzr,iiex,jjey,kkez,nnx,nny,nnz,llengt,mmaxit
      parameter(iixp=3 ,jjyq=2,kkzr=3 )
      parameter(iiex=4,jjey=7,kkez=4)
      parameter(nnx = iixp*2** (iiex-1)+1)
      parameter(nny = jjyq*2** (jjey-1)+1)
      parameter(nnz = kkzr*2** (kkez-1)+1)
      parameter(llengt=1176431)
      parameter(mmaxit = 10)
      real rhs(nnx,nny,nnz),phi(nnx,nny,nnz)
      integer iparm(23),mgopt(4),icrs(3)
      integer i,j,k,ierror,maxit,iouter
      real work(llengt),fparm(8),rmax(mmaxit)
C
C      use labelled common to identify integer and
C      floating point arguments
C
      integer
intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex,jey,
+kkez,nx,ny,nz,iguess,maxcy,method,meth2,nwork,lwkmin,ite
ro
      common
/iprm/intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex,jey,
+kkez,nx,ny,nz,iguess,maxcy,method,meth2,nwork,lwkmin,ite
ro
      real xa,xb,yc,yd,ze,zf,tolmax,relmax
      common/fprm/xa,xb,yc,yd,ze,zf,tolmax,relmax
      real pi,dx,dy,dz,c,s,x,y,z,tol,err2
      real cxx,cyy,czz,cx,cy,cz,ce,cxy,cyz
      real pxx,pyy,pzz,px,py,pz,pxy,pxz,pyz,pe
      equivalence(iparm,intl)
      equivalence(fparm,xa)
      external cof,bd3cr,cxyf,cxzf,cyzf

```

```
pi = 4.*atan(1.)  
c  
c      set interval end points  
c  
      xa = 0.0  
      xb = 1.0  
      yc = 0.0  
      yd = pi+pi  
      ze = 0.0  
      zf = 1.0  
c  
c      set required no error control within multigrid  
cycling  
c  
      tolmax = 0.0  
c  
c      set integer input arguments  
c  
      intl = 0  
c  
c      set boundary condition flags  
c  
      nxa = 1  
      nxb = 2  
      nyc = 0  
      nyd = 0  
      nze = 1  
      nzf = 2  
c  
c      set grid size arguments from parameter statements  
c  
      ixp = iixp  
      jyq = jjyq  
      kzs = kkzs  
      iex = iiex  
      jey = jjey  
      kez = kkez  
      nx = nnx  
      ny = nny  
      nz = nnz  
c  
c      flag nonzero xy and yz and zero xz cross  
derivatives terms  
c
```

```

icrs(1) = 1
icrs(2) = 0
icrs(3) = 1
C
C      set two multigrid cycles per outer iteration
C
maxcy = 2
C
C      set point relaxation
C
method = 0
meth2 = 0
C
C      set work space length input
C
nwork = llengt
C
C      set uniform grid interval lengths in each
dimension
C
dx = (xb-xa) / (nx-1)
dy = (yd-yc) / (ny-1)
dz = (zf-ze) / (nz-1)
C
C      set right hand side and preset solution to zero
C      this also sets specified (Dirchlet) b.c. in phi
C
do j=1,ny
y = (j-1)*dy
s = sin(y)
do k=1,nz
z = (k-1)*dz
do i=1,nx
x = (i-1)*dx
call
exact(x,y,z,pe,px,py,pz,pxx,pyy,pzz,pxy,pxz,pyz)
call cof(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
call cxyf(x,y,z,cxy)
call cyzf(x,y,z,cyz)
rhs(i,j,k) =
cxx*pxx+cyy*pyy+czz*pzz+cxy*pxy+cyz*pyz+
+                               cx*px+cy*py+cz*pz+ce*pe
phi(i,j,k) = 0.0
end do

```

```

    end do
    end do
C
C      set default multigrid options
C
      mgopt(1) = 0
C
C      set error control tolerance and outer iteration
limit of 10
C
      tol = .001
      maxit = mmaxit
C
C      print input arguments shared with mud3
C
      write(*,50)
50 format(//' mud3cr test ' )

write(6,100) intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex
,jey,kez,
+
nx,ny,nz,iguess,maxcy,method,nwork,xa,xb,yc,yd,ze,zf,
+          tolmax,mgopt(1)
100 format(/' input arguments ',
+/' intl = ',i2,
+/' nxa = ',i2,' nxb = ',i2,
+/' nyc = ',i2,' nyd = ',i2,
+/' nze = ',i2,' nzf = ',i2,
+/' ixp = ',i2,' jyq = ',i2,' kzr = ',i2,
+/' iex = ',i2,' jey = ',i2,' kez = ',i2,
+/' nx = ',i3,' ny = ',i3,' nz = ',i3,
+/' iguess = ',i2,' maxcy = ',i2,
+/' method = ',i2,' work space input = ',i7,
+/' xa = ',f5.2,' xb = ',f5.2,
+/' yc = ',f5.2,' yd = ',f5.2,
+/' ze = ',f5.2,' zf = ',f5.2,
+/' tolmax = ',e10.3
+/' multigrid options '
+/' mgopt(1) = ',i2)

C
C      print new mud3cr arguments
C
      write(*,101) (icrs(i),i=1,3),tol,maxit
101 format(/' new mud3cr arguments ', / ' icrs '/ 3i5,

```

```

        +' tol = ',f6.4 /' maxit = ',i3)
C
C *** initialization call
C
        intl = 0
        iguess = 0
        call
mud3cr(iparm,fparm,work,cof,bd3cr,rhs,phi,mgopt,icrs,
        +cxyf,cxzf,cyzf,tol,maxit,iouter,rmax,ierror)
        write (6,200) intl,iguess,ierror,iparm(22)
200 format(/' initial call', /' intl = ',i2, /' iguess
= ',i2,
        +' ierror = ',i2, /' minimum required work space
length = ',i8)
        if (ierror.gt.0) call exit(0)
C
C *** noninitial call
C
        intl = 1
        iguess = 0
        call
mud3cr(iparm,fparm,work,cof,bd3cr,rhs,phi,mgopt,icrs,
        +cxyf,cxzf,cyzf,tol,maxit,iouter,rmax,ierror)
        write (6,201)
intl,iguess,ierror,iouter,(rmax(i),i=1,iouter)
201 format(/' approximation call ', /' intl = ',i2, /'
iguess = ',i2,
        +' ierror = ',i3,
        +' number of outer iterations executed = ',i3,
        +' relative difference profile:', /(10(f6.4,2x)))
C
C      compute and print exact least squares error after
iouter iterations
C
        err2 = 0.
        do j=1,ny
        y = (j-1)*dy
        c = cos(y)
        do k=1,nz
        z = (k-1)*dz
        do i=1,nx
        x = (i-1)*dx
        pe = (x*c*z)**3
        err2 = err2 + (pe - phi(i,j,k))**2

```

```

    end do
end do
    end do
    err2 = sqrt(err2/ (nx*ny*nz) )
    write(6,202) err2
202 format(' exact least squares error = ',e10.3)
    end

    subroutine
exact(x,y,z,pe,px,py,pz,pxx,pyy,pzz,pxy,pxz,pyz)
C
C      set exact solution and partial derivatives:
p(x,y,z) = (x*cos(y)*z)**3
C
        implicit none
        real x,y,z,pe,px,py,pz,pxx,pyy,pzz,pxy,pxz,pyz
        real s,c,x,xc,xz,cz,xcz
        c = cos(y)
        s = sin(y)
        xc = x*c
        xz = x*z
        cz = c*z
        xcz = xc*z
        pe = xcz**3
        px = 3.*x*x*cz**3
        pxx = 6.*x*cz**3
        py = -3.*c*c*s*xz**3
        pyy = -3.*xz**3*(c**3-2.*c*s**2)
        pz = 3.*z*z*xc**3
        pzz = 6.*z*xc**3
        pxy = -9.*x**2*c*c*s*z**3
        pxz = 9.*xz**2*c**3
        pyz = -9.*c*c*s*x**3*z**2
        return
    end

    subroutine cof(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
C
C      set noncross derivative pde coefficients
C
        implicit none
        real x,y,z,cxx,cyy,czz,cx,cy,cz,ce,s
        s = sin(y)
        cxx = 1.+0.5*s*z

```

```

cyy = 1.+x*z
czz = 1.+0.5*x*s
cx = 0.0
cy = -x*z
cz = 0.0
ce = -(x+z)
return
end

subroutine cxzf(x,y,z,cxy)
C
C      set x-y cross term coefficient at (x,y,z)
C
implicit none
real x,y,z,cxx,cyy,czz,cx,cy,cz,ce,cxy
call cof(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
cxy = sqrt(cxx*cyy)
return
end

subroutine cxzf(x,y,z,cxz)
C
C      this is a dummy subroutine since cxz=0.0 for all
(x,y,z)
C
return
end

subroutine cyzf(x,y,z,cyz)
C
C      set y-z cross term coefficient at (x,y,z)
C
implicit none
real x,y,z,cxx,cyy,czz,cx,cy,cz,ce,cyz
call cof(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
cyz = sqrt(cyy*czz)
return
end

subroutine bd3cr(kbdy,xory,yorz,a,b,c,g)
C
C      pass mixed derivative boundary conditions to
mud3cr
C

```

```

implicit none
integer kbdy
real xory,yorz,a,b,c,g,x,y,z
real pe,px,py,pz,pxx,pyy,pzz,pxy,pxz,pyz
if (kbdy.eq.2) then
C
C      upper x boundary (mixed oblique)
C
x = 1.0
y = xory
z = yorz
a = (z*(1.-z))
b = sin(y)*cos(y)
c = -1.0
call exact(x,y,z,pe,px,py,pz,pxx,pyy,pzz,pxy,pxz,pyz)
g = px + a*py + b*pz + c*pe
return
      else if (kbdy.eq.6) then
C
C      upper z boundary (mixed normal)
C
z = 1.0
x = xory
y = yorz
a = 0.0
b = 0.0
c = x*sin(y)**2
call exact(x,y,z,pe,px,py,pz,pxx,pyy,pzz,pxy,pxz,pyz)
g = pz + a*px + b*py + c*pe
return
      end if
end

```

---

## TMUD3SA

```

C
C      file tmud3sa.f
C

```

```
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * * *  
C      *  
*  
C      *          copyright (c) 2008 by UCAR  
*  
C      *  
*  
C      *          University Corporation for Atmospheric  
Research      *  
C      *  
*  
C      *          all rights reserved  
*  
C      *  
*  
C      *          MUDPACK version 5.0.1  
*  
C      *  
*  
C      *          A Fortran Package of Multigrid  
*  
C      *  
*  
C      *          Subroutines and Example Programs  
*  
C      *  
*  
C      *          for Solving Elliptic Partial Differential  
Equations      *  
C      *  
*  
C      *          by  
*  
C      *  
*  
C      *          John Adams  
*  
C      *  
*  
C      *          of  
*  
C      *  
*
```

```
C      *          the National Center for Atmospheric
Research           *
C      *
*
C      *          Boulder, Colorado (80307)
U.S.A.           *
C      *
*
C      *          which is sponsored by
*
C      *
*
C      *          the National Science Foundation
*
C      *
*
C      *          ****
* * * * *
C
C ... purpose
C
C      test program for the MUDPACK solver mud3sa
C
C ... required MUDPACK files
C
C      mud3sa.f, mudcom.f, mud3ln.f, mud3pn.f
C
C
C
*****
*
C
*****
*
C
*****
sample program/test driver for mud3sa
C
C
*****
**
C
*****
**
C
```

```

c
c
c      a sample program/test driver for mud3sa is below.
it can be
c      executed as an initial test.  the output is listed
for the
c      the test case described.
c
c      solve the following helmholtz equation (divergence
form) in
c      spherical coordinates.
c
c      div(grad(u(r,t,p)) - u(r,t,p) = f(r,t,p)
(original pde)
c
c      after multiplying the left and right hand sides by
r*r*sin(t) this
c      becomes
c
c      r*r*sin(t)*div(grad(u(r,t,p)) -
r*r*sin(t)*u(r,t,p)
c
c      = r*r*sin(t)*f(r,t,p)
c
c      or (expanded form)
c
c
c      d(r*r*sin(t)*du/dr)/dr + d(sin(t)*du/dt)/dt +
d(1/sin(t)*du/dp)/dp
c
c      - r*r*sin(t)*u(r,t,p) =
r*r*sin(t)*f(r,t,p).
c
c
c      this form is suitable for using mud3sa.  the
solution region is:
c
c      (pi = 4.*atan(1.0) radians)
c
c      0.5 < r < 1.0          r is the "radius"
coordinate
c
c      pi/4 < t < 3*pi/4    t is the colatitude
coordinate

```

```

c
c           0 < p < pi+pi      p is the longitude
coordinate
c
c       asssume the solution is specified at the r,t
boundaries and is periodic
c       in p. use line relaxation in the r direction and
choose a solution
c       grid as close to 50 x 30 x 60 as the mudpack size
constraints allow.
c       the exact solution
c
c           u(r,t,p) =
(r*sin(t)*sin(p)*r*sin(t)*cos(p)*r*cos(t))**2
c
c           = (x*y*z)**2      (in cartesian
coordinates)
c
c       is used for testing purposes. one full multigrid
cycle (no initial
c       guess) with the default multigrid options is
executed. error control
c       is not used.
c
c ****
c       output (32 bit floating point arithmetic
c
*****mud3sa test
c
c       input arguments
c       intl = 0 nra = 1 nrb = 1 ntc = 1 ntd = 1
c       npe = 0 npf = 0
c       irp = 3 jtq = 2 kpr = 2
c       ier = 5 jet = 5 kep = 6
c       nr = 49 nt = 33 np = 65 iguess = 0 maxcy = 1
c       method = 1 work space length input = 1598164
c       ra = 0.50 rb = 1.00
c       tc = 0.79 td = 2.36
c       pe = 0.00 pf = 6.28
c       tolmax = 0.000E+00
c
c       multigrid options

```

```

c      kcyle =  2
c      iprer =  2
c      ipost =  1
c      interpol = 3
c
c      discretization call to mud3sa intl =  0
c      ierror =  0 minimum work space = 1598164
c
c      approximation call to mud3sa
c      intl =  1 method =  1 iguess =  0 maxcy =  1
c      ierror =  0
c      maximum error = 0.369E-03
c
c
c ****
c ****
c      end of output
c
c ****
c ****
c
c      program tmud3sa
c      implicit none
c
c      set grid sizes with parameter statements
c
c      integer iirp,jjtq,kkpr,iier,jjet,kkep,nnr,nnt,nnp
c      integer irp,jtq,kpr,ier,jet,kep,rr,nt,np
c      parameter(iirp=3, jjtq=2, kkpr=2, iier=5, jjet=5,
c      kkep=6)
c          parameter (nnr = iirp*2** (iier-1)+1)
c          parameter (nnt = jjtq*2** (jjet-1)+1)
c          parameter (nnp = kkpr*2** (kkep-1)+1)
c
c      set predetermined minimum work space required
c
c      integer llwork
c      parameter (llwork = 1598164)
c
c      dimension solution,right hand side, and work
c      arrays
c
c      real
uso(nnr,nnt,nnp),rhs(nnr,nnt,nnp),work(llwork)

```

```

c
c      put integer and floating point parameter names in
contiguous
c      storeage for labelling in equivalenced vectors
iprm,fprm
c
      real fprm(8)
      integer iprm(23),mgopt(5)
      integer
intl,nra,nrb,ntc,ntd,npe,npf,irp,jtq,kpr,ier,jet,
      +
kep,nr,nt,np,iguess,maxcy,method,method2,nwork,
      +           lwrkqd,itero

common/itmud3/intl,nra,nrb,ntc,ntd,npe,npf,irp,jtq,kpr,i
er,jet,
      +
kep,nr,nt,np,iguess,maxcy,method,method2,nwork,
      +           lwrkqd,itero
      real ra,rb,tc,td,pe,pf,tolmax,relmax
      common/ftmud3/ra,rb,tc,td,pe,pf,tolmax,relmax
      real sinat,cosat,sinap,cosap,dlr,dlt,dlp
      integer i,j,k,ierror
      real urr,utt,upp,ur,ut,up,ue,r,r6,t,p,pi
      real st,ct,sp,cp,ep,et,ep,sint,cost,errm
c
c      vectors for saving sin,cos on lat-long points
c

common/sincos/sinat(33),cosat(33),sinap(65),cosap(65),dl
r,dlt,dlp
      equivalence(intl,iprm)
      equivalence(ra,fprm)
c
c      declare coefficient input functions external
c
      real lam
      external sigr,sigt,sigp,bndc,lam
c
c
c      set input integer arguments
c
      intl = 0
c

```

```
c      set boundary condition flags
c
nra = 1
nrb = 1
ntc = 1
ntd = 1
npe = 0
npf = 0
c
c      set grid sizes from arguments statements
c
irp = iirp
jtq = jjtq
kpr = kkpr
ier = iier
jet = jjet
kep = kkep
nr = nnr
nt = nnt
np = nnr
c
c      set one cycle limit
c
maxcy = 1
c
c      set work space length approximation from parameter
statement
c
nwork = llwork
c
c      set line relaxation in the r direction
c
method = 1
method2 = 0
c
c      flag no initial guess which forces full multigrid
cycling
c
iguess = 0
c
c      set ends of solution "cube" in (r,t,p) space
(actually the sphere)
c
pi = 4.*atan(1.)
```

```

    ra = 0.5
    rb = 1.0
    tc = 0.25*pi
    td = 0.75*pi
    pe = 0.0
    pf = pi+pi

C
C      set mesh increments
C
        dlr = (rb-ra)/float(nr-1)
        dlt = (td-tc)/float(nt-1)
        dlp = (pf-pe)/float(np-1)

C
C      preset sin,cos on lat-long grid to save
computation
C
        do k=1,np
        p = pe +(k-1)*dlp
        sinap(k) = sin(p)
        cosap(k) = cos(p)
        end do
        do j=1,nt
        t = tc+(j-1)*dlt
        sinat(j) = sin(t)
        cosat(j) = cos(t)
        end do

C
C      set for no error control
C
        tolmax = 0.0

C
C      set right hand side in rhs and initialize solution
to zero
C
        do j=1,nt
        t = tc+(j-1)*dlt
        sint = sinat(j)
        cost = cosat(j)
        do k=1,np
        p = pe+(k-1)*dlp
        do i=1,nr
        r = ra+float(i-1)*dlr
        call exact(r,t,p,urr,utt,upp,ur,ut,up,ue)
        rhs(i,j,k) =

```

```

r*sint* (r*urr+2.*ur)+sint*utt+cost*ut+upp/sint
      +                               - lam(r,t,p)*ue
      uso(i,j,k) = 0.0
end do
end do
end do
C
C      set specified values in uso at r and t boundaries
C
C      r = ra
      r6 = ra**6
      do k=1,np
      sp = sinap(k)
      cp = cosap(k)
      ep = (sp*cp)**2
      do j=1,nt
      st = sinat(j)
      ct = cosat(j)
      et = st*st*(st*ct)**2
      uso(1,j,k) = r6*et*ep
      end do
      end do
C
C      r = rb
      r6 = rb**6
      do k=1,np
      sp = sinap(k)
      cp = cosap(k)
      ep = (sp*cp)**2
      do j=1,nt
      st = sinat(j)
      ct = cosat(j)
      et = st*st*(st*ct)**2
      uso(nr,j,k) = r6*et*ep
      end do
      end do

C      t = tc
      st = sinat(1)
      ct = cosat(1)
      et = st*st*(st*ct)**2
      do k=1,np
      sp = sinap(k)
      cp = cosap(k)

```

```

ep = (cp*sp)**2
do i=1,nr
    r6 = (ra+float(i-1)*dlr)**6
    uso(i,1,k) = r6*et*ep
end do
    end do

c      t = td
st = sinat(nt)
ct = cosat(nt)
et = st*st*(st*ct)**2
do k=1,np
sp = sinap(k)
cp = cosap(k)
ep = (cp*sp)**2
do i=1,nr
    r6 = (ra+float(i-1)*dlr)**6
    uso(i,nt,k) = r6*et*ep
end do
    end do
c
c      set default multigrid options
c
mgopt(1) = 2
mgopt(2) = 2
mgopt(3) = 1
mgopt(4) = 3
c
c      write input arguments
c
      write(*,50)
50 format(//' mud3sa test')

write(*,100) intl,nra,nrb,ntc,ntd,npe,npf,irp,jtq,kpr,ier
,jet,kep,
+
nr,nt,np,iguess,maxcy,method,nwork,ra,rb,tc,td,pe,pf,
+
tolmax,mgopt(1),mgopt(2),mgopt(3),mgopt(4)
100 format(/' input arguments',
+/' intl = ',i2,' nra = ',i2,' nrb = ',i2,' ntc =
',i2,' ntd = ',i2,
+/' npe = ',i2,' npf = ',i2,
+/' irp = ',i2,' jtq = ',i2,' kpr = ',i2,

```

```

+/' ier = ',i2, ' jet = 'i2, ' kep = 'i2,
+/' nr = ',i3,' nt = ',i3,' np = ',i3, ' iguess =
'i2, ' maxcy = 'i2,
+/' method = 'i2, ' work space length input = ',i7,
+/' ra = 'f5.2,' rb = 'f5.2,
+/' tc = 'f5.2,' td = 'f5.2,
+/' pe = 'f5.2,' pf = 'f5.2,
+/' tolmax = ' ,e10.3
+/' multigrid options '
+/' kcycle = ',i2
+/' iprер = ',i2
+/' ipost = ',i2
+/' interpol = ',i2 )
      write(*,104) intl
104 format(/' discretization call to mud3sa', ' intl =
', i2)
      call
mud3sa(iprm,fprm,work,sigr,sigt,sigp,lam,bndc,rhs,uso,
       +
       mgopt,ierror)
      write (*,105) ierror,iprm(22)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)
C
C      attempt solution
C
      intl = 1
      write(*,106) intl,method,iguess,maxcy
106 format(/' approximation call to mud3sa ',
+/' intl = ',i2, ' method = ',i2,' iguess = ',i2, '
maxcy = ',i2)
      call
mud3sa(iprm,fprm,work,sigr,sigt,sigp,lam,bndc,rhs,uso,
       +
       mgopt,ierror)
      write (*,107) ierror
107 format(' ierror = ',i2)
      if (ierror.gt.0) call exit(0)
C
C      compute maximum error
C
      errm = 0.0
      do k=1,np
sp = sinap(k)
cp = cosap(k)

```

```

ep = (cp*sp)**2
do j=1,nt
    st = sinat(j)
    ct = cosat(j)
    et = st*st*(st*ct)**2
    do i=1,nr
        r6 = (ra+float(i-1)*dlr)**6
c      exact continuous solution
        ue = r6*et*ep
        errm = amax1(errm,abs(uso(i,j,k)-ue))
    end do
end do
end do
write(*,108) errm
108 format(' maximum error = ',e10.3)
end

function sigr(r,t,p)
c
c      coefficient for r derivative (mud3sa will call
sigr off grid)
c
    implicit none
    real sigr,r,t,p
    sigr = r*r*sin(t)
    return
end

function sigt(r,t,p)
c
c      coefficient for theta derivative (mud3sa will call
sigt off grid)
c
    implicit none
    real sigt,r,t,p
    sigt = sin(t)
    return
end

function sigp(r,t,p)
c
c      coefficient for phi derivative (mud3sa will call
sigp off grid)
c

```

```

implicit none
real sigp,r,t,p
sigp = 1.0/sin(t)
return
end

function lam(r,t,p)
c
c      input zero order coefficient in self adjoint pde
at (r,t,p) to mud3sa
c      (only grid values needed)
c
implicit none
real lam,r,t,p
integer j
integer
intl,nra,nrb,ntc,ntd,npe,npf,irp,jtq,kpr,ier,jet,
+
kep, nr, nt, np, iguess, maxcy, method, method2, nwork,
+           lwrkqd, itero

common/itmud3/intl,nra,nrb,ntc,ntd,npe,npf,irp,jtq,kpr,i
er,jet,
+
kep, nr, nt, np, iguess, maxcy, method, method2, nwork,
+           lwrkqd, itero
real ra,rb,tc,td,pe,pf,tolmax,relmax
real sinat,cosat,sinap,cosap,dlr,dlt,dlp
common/ftmud3/ra,rb,tc,td,pe,pf,tolmax,relmax

common/sincos/sinat(33),cosat(33),sinap(65),cosap(65),dl
r,dlt,dlp
real lam
j = int((t-tc)/dlt+0.5)+1
lam = r*r*sinat(j)
return
end

c
c
subroutine bndc(kbdy,rort,torp,alfa,gbdy)
c
c      a dummy routine since there is no mixed derivative
b.c. for this problem
c

```

```

    return
    end

    subroutine exact(r,t,p,urr,utt,upp,ur,ut,up,ue)
C
C      set exact solution taken from u(x,y,z) =
C      (x*y*z)**2 transformed
C      with spherical coordinates
C

common/itmud3/intl,nra,nrb,ntc,ntd,npe,npf,irp,jtq,kpr,i
er,jet,
+
kep, nr, nt, np, iguess, maxcy, method, method2, nwork,
+           lwrkqd, itero
common/ftmud3/ra,rb,tc,td,pe,pf,tolmax,relmax

common/sincos/sinat(33),cosat(33),sinap(65),cosap(65),dl
r,dlt,dlp
C
C      set subscripts for current lat-lon grid point
C
        j = int((t-tc)/dlt+0.5)+1
        k = int((p-pe)/dlp+0.5)+1
C
C      set sin, cos from pre-computed vectors
C
        st = sinat(j)
        ct = cosat(j)
        sp = sinap(k)
        cp = cosap(k)
C
C      set intermediate quantities
C
        r6 = r**6
        ep = (cp*sp)**2
        dep = 2.* (cp*sp)*(cp*cp-sp*sp)
        ddep = 2.* ((cp**2-sp**2)**2-4.* (sp*cp)**2)
        et = st*st*(st*ct)**2
        det = 2.* (2.* (st*ct)**3 -ct*st**5)
        ddet = 12.* (ct*st)**2*(ct**2-
        st**2)+2.*st**4*(st**2-4.*ct**2)
C
C      set exact values of continuous solution and its

```

```

partial derivatives
c
ue = r6*et*ep
ur = 6.*r**5*et*ep
urr = 30.*r**4*et*ep
ut = r6*det*ep
utt = r6*ddep*ep
up = r6*et*dep
upp = r6*et*ddep
return
end

```

TMUD3SP

```
C      file tmud3sp.f
C
C      * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * *
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *               University Corporation for Atmospheric
Research          *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK version 5.0.1
*
C      *
*
C      *               A Fortran Package of Multigrid
```



```

C      test program for the MUDPACK solver mud3sp
C
C ... required MUDPACK files
C
C      mud3sp.f, mudcom.f
C
C
C
***** ****
*
C
***** ****
*
C
C      sample program/test driver for mud3sp
C
C
***** ****
**
C
***** ****
**
C
C      a sample program/test driver for mud3sp is below.
it can be
c      executed as an initial test.  the output is listed
for the
c      test case described.
c
c      test the driver below by solving the separable
elliptic pde
c
c      d(x**2*dp/dx)/dx + d(y**2*dp/dy)/dy +
d(z**2*dp/dz)/dz
c
c      - (x+y+z) *pe(x,y,z) = r(x,y,z)
c
c
c      on the cube [0.5,1.0]**3 with specified boundary
conditions at x,y,z=1.0
c      and derivative boundary conditions of the form
c
c      dp/dx - pe(0.5,y,z) = f(0.5,y,z) at x = 0.5
c

```

```

c      dp/dy - pe(x,0.5,z) = g(x,0.5,z) at y = 0.5
c
c      dp/dz - pe(x,y,0.5) = h(x,y,0.5) at z = 0.5
c
c      generate the approximation on a  65 x 49 x 41 grid
c      using the exact
c      solution
c
c      pe(x,y,z) = (x*y*z)**4
c
c      for testing. notice the solver mud3 would require
c      far more work space
c      than mud3sp if used to solve this pde.
c
c ****
c      output (32 bit floating point arithmetic)
c
*****  

c
c      mud3sp test
c
c      input arguments
c      intl = 0 nxa = 2 nxb = 1 nyc = 2 nyd = 1
c      nze = 2 nzf = 1
c      ixp = 2 jyq = 3 kxr = 5
c      iex = 6 jey = 5 kez = 4
c      nx = 65 ny = 49 nz = 41 iguess = 0 maxcy = 3
c      method = 0 work space length input = 514258
c      xa = 0.50 xb = 1.00
c      yc = 0.50 yd = 1.00
c      ze = 0.50 zf = 1.00
c      tolmax = 0.000E+00
c
c      multigrid options
c      kcyle = 2
c      iprер = 2
c      ipost = 1
c      interpol = 1
c
c      discretization call to mud3sp intl = 0
c      ierror = 0 minimum work space = 472557
c
c      approximation call to mud3sp
c      intl = 1 method = 0 iguess = 0 maxcy = 3

```

```

c      ierror = 0
c      maximum error = 0.719E-05
c
c
c ****
c ****
c      end of output
c
c ****
c ****
c
c      program tmud3sp
c
c      set grid sizes with parameter statements
c
c      implicit none
c
c      set grid sizes with parameter statements
c
c      integer
iixp,jjyq,kkzr,iiex,jjey,kkez,llwork,nnx,nny,nnz
      parameter(iixp=2,jjyq=3,kkzr=5)
      parameter(iiex=6,jjey=5,kkez=4)
      parameter (nnx = iixp*2** (iiex-1)+1)
      parameter (nny = jjyq*2** (jjey-1)+1)
      parameter (nnz = kkzr*2** (kkez-1)+1)
c
c      set work space length approximation (see mud3sp.d)
c
c      parameter(llwork = 7* (nnx+2) * (nny+2) * (nnz+2) / 2 )
c
c
c      dimension solution,right hand side, and work
arrays
c
      real
phi(nnx,nny,nnz),rhs(nnx,nny,nnz),work(llwork)
      integer iprm(22),mgopt(4)
      real fprm(8)
c
c      put integer and floating point arguments names in
contiguous
c      storeage for labelling

```

```

c
      integer
intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex,jey,
      +
kez,nx,ny,nz,iguess,maxcy,method,nwork,lwrkqd,itero

common/itmsp3/intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,i
ex,jey,
      +
kez,nx,ny,nz,iguess,maxcy,method,nwork,lwrkqd,itero
      real xa,xb,yc,yd,ze,zf,tolmax,relmax
      common/ftmsp3/xa,xb,yc,yd,ze,zf,tolmax,relmax
      real
dlx,dly,dlz,x,y,z,cxx,cyy,czz,cx,cy,cz,cex,cey,cez,ce,er
rm
      real pxx,pyy,pzz,px,py,pz,pe
      integer i,j,k,ierror
      equivalence(intl,iprm)
      equivalence(xa,fprm)

c
c      declare coefficient and boundary condition input
subroutines external
c
      external cfx,cfy,cfz,bndc
c
c
c      set input integer parameters
c
      intl = 0
c
c      set boundary condition flags
c
      nxa = 2
      nxb = 1
      nyc = 2
      nyd = 1
      nze = 2
      nzf = 1

c
c      set grid sizes from parameter statements
c
      ixp = iixp
      jyq = jjyq

```

```
kzr = kkzr
iex = iiex
jey = jjey
kez = kkez
nx = nnx
ny = nny
nz = nnz

C
C      set work space length approximation from parameter
statement

C
nwork = llwork

C
C      set method of relaxation--point gauss/seidel only
for mud3sp

C
method = 0

C
C      set default multigrid options

C
mgopt(1) = 2
mgopt(2) = 2
mgopt(3) = 1
mgopt(4) = 3

C
C      set full multigrid cycling by flagging no initial
guess

C
iguess = 0

C
C      set a limit of three w(2,1) cycles from the finest
level

C
maxcy = 3

C
C      set end points of solution cube in (x,y,z) space

C
xa = 0.5
xb = 1.0
yc = 0.5
yd = 1.0
ze = 0.5
zf = 1.0
```

```

c      set mesh increments
c
dix = (xb-xa)/float(nx-1)
dly = (yd-yc)/float(ny-1)
dlz = (zf-ze)/float(nz-1)
c
c      set for no error control (this allows phi to be
equivlaenced with work)
c
tolmax = 0.0
c
c      set right hand side in rhs using exact solution
c      and initialize phi to zero
c
do k=1,nz
z = ze+(k-1)*dlz
call cfz(z,czz,cz,cez)
do j=1,ny
y = yc+float(j-1)*dly
call cfy(y,cyy,cy,cey)
do i=1,nx
x = xa+float(i-1)*dix
call cfx(x,cxx,cx,cex)
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
ce = cex+cey+cez
rhs(i,j,k) =
cxx*pzx+cyy*pyy+czz*pzz+cx*px+cy*py+cz*pz+ce*pe
phi(i,j,k) = 0.0
end do
end do
end do
c
c      set specified values at upper boundaries
c
x = xb
do k=1,nz
z = ze+(k-1)*dlz
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(nx,j,k) = pe
end do
end do
y = yd

```

```

      do k=1,nz
      z = ze+(k-1)*dlz
      do i=1,nx
         x = xa+(i-1)*dlx
         call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
         phi(i,ny,k) = pe
      end do
      end do
      z = zf
      do j=1,ny
         y = yc+(j-1)*dly
         do i=1,nx
            x = xa+float(i-1)*dlx
            call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
            phi(i,j,nz) = pe
         end do
         end do
         write(6,50)
50 format(//' mud3sp test ')
C
C      print input arguments
C

write(6,100)intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex
,jey,kez,
+
nx,ny,nz,iguess,maxcy,method,nwork,xa,xb,yc,yd,ze,zf,
+
tolmax,mgopt(1),mgopt(2),mgopt(3),mgopt(4)
100 format(/' input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =
',i2,' nyd = ',i2,
+/' nze = ',i2,' nzf = ',i2,
+/' ixp = ',i2,' jyq = ',i2,' kzr = ',i2,
+/' iex = ',i2,' jey = ',i2,' kez = ',i2,
+/' nx = ',i3,' ny = ',i3,' nz = ',i3,' iguess =
',i2,' maxcy = ',i2,
+/' method = ',i2,' work space length input = ',i7,
+/' xa = ',f5.2,' xb = ',f5.2,
+/' yc = ',f5.2,' yd = ',f5.2,
+/' ze = ',f5.2,' zf = ',f5.2,
+/' tolmax = ',e10.3
+/' multigrid options '
+/' kcyle = ',i2

```

```

+/' iprер = ',i2
+/' ipоst = ',i2
+/' intрol = ',i2 )
C
C      discretize pde
C
      write(*,104) intl
104 format(/' discretization call to mud3sp', ' intl =
', i2)
      call
mud3sp(iprm,fprm,work,cfx,cfy,cfz,bndc,rhs,phi,mgopt,ier
ror)
      write (*,105) ierror,iprm(21)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)
C
C      approximate pde
C
      intl = 1
      write(*,106) intl,method,iguess,maxcy
106 format(/' approximation call to mud3sp ',
+/' intl = ',i2, ' method = ',i2,' iguess = ',i2, '
maxcy = ',i2)
      call
mud3sp(iprm,fprm,work,cfx,cfy,cfz,bndc,rhs,phi,mgopt,ier
ror)
      write (*,107) ierror
107 format(' ierror = ',i2)
      if (ierror.gt.0) call exit(0)
C
C      compute and print maximum error
C
      call error(nx,ny,nz,phi,errm)
      write(*,108) errm
108 format(' maximum error = ',e10.3)
      end

      subroutine error(nx,ny,nz,phi,errm)
C
C      compute the error in the estimate in phi
C
      implicit none
      integer nx,ny,nz

```

```

real phi(nx,ny,nz),errm
real xa,xb,yc,yd,ze,zf,tolmax,relmax
common/ftmsp3/xa,xb,yc,yd,ze,zf,tolmax,relmax
real dlx,dly,dlz,x,y,z,pxx,pyy,pzz,px,py,pz,pe
integer i,j,k
dlx = (xb-xa) / (nx-1)
dly = (yd-yc) / (ny-1)
dlz = (zf-ze) / (nz-1)
errm = 0.0
do k=1,nz
  z = ze+(k-1)*dlz
  do j=1,ny
    y = yc+float(j-1)*dly
    do i=1,nx
      x = xa+float(i-1)*dlx
      call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
      errm = amax1(errm,abs(phi(i,j,k)-pe))
    end do
  end do
  end do
  return
end

C
C coefficient subroutines
C
subroutine cfx(x,cxx,cx,cex)
implicit none
real x,cxx,cx,cex
cxx = x*x
cx = x+x
cex = -x
return
end

subroutine cfy(y,cyy,cy,cey)
implicit none
real y,cyy,cy,cey
cyy = y*y
cy = y+y
cey = -y
return
end

subroutine cfz(z,czz,cz,cez)

```

```

implicit none
real z,czz,cz,cez
czz = z*z
cz = z+z
cez = -z
return
end

subroutine bndc(kbdy,xory,yorz,cons,gbdy)
C
C      input derivative boundary conditions to mud3sp
C      at lower x,y,z boundaries
C
implicit none
integer kbdf
real
xory,yorz,cons,gbdy,x,y,z,pxx,pyy,pzz,px,py,pz,pe
real xa,xb,yc,yd,ze,zf,tolmax,relmax
common/ftmsp3/xa,xb,yc,yd,ze,zf,tolmax,relmax
if (kbdf.eq.1) then
C
C      x=xa surface
C
x = xa
y = xory
z = yorz
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
cons = -1.0
gbdy = px+cons*pe
return
end if
if (kbdf.eq.3) then
C
C      y=yc surface
C
y = yc
x = xory
z = yorz
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
cons = -1.0
gbdy = py+cons*pe
return
end if
if (kbdf.eq.5) then

```

```

C
C      z=ze surface
C
C      z = ze
C      x = xory
C      y = yorz
C      call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
C      cons = -1.0
C      gbdy = pz + cons*pe
C      return
C          end if
C          end

      subroutine exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
C
C      this subroutine is used to set an exact solution
for testing mud3sp
C      (i.e., setting the rhs, boundary conditions and
computing the exact
C      error)
C
C      implicit none
      real x,y,z,pxx,pyy,pzz,px,py,pz,pe
      pe = (x*y*z)**4
      px = 4.* (x*y*z)**3*y*z
      py = 4.* (x*y*z)**3*x*z
      pz = 4.* (x*y*z)**3*x*y
      pxx = 12.* (x*y*z)**2*(y*z)**2
      pyy = 12.* (x*y*z)**2*(x*z)**2
      pzz = 12.* (x*y*z)**2*(x*y)**2
      return
      end

```

---

## TMUH2

```

C
C      file tmuh2.f
C

```

```
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
* * * * * * * * *  
C      *  
*  
C      *          copyright (c) 2008 by UCAR  
*  
C      *  
*  
C      *          University Corporation for Atmospheric  
Research      *  
C      *  
*  
C      *          all rights reserved  
*  
C      *  
*  
C      *          MUDPACK version 5.0.1  
*  
C      *  
*  
C      *          A Fortran Package of Multigrid  
*  
C      *  
*  
C      *          Subroutines and Example Programs  
*  
C      *  
*  
C      *          for Solving Elliptic Partial Differential  
Equations      *  
C      *  
*  
C      *          by  
*  
C      *  
*  
C      *          John Adams  
*  
C      *  
*  
C      *          of  
*  
C      *  
*
```

```
C      *          the National Center for Atmospheric
Research           *
C      *
*
C      *          Boulder, Colorado (80307)
U.S.A.           *
C      *
*
C      *          which is sponsored by
*
C      *
*
C      *          the National Science Foundation
*
C      *
*
C      *          ****
* * * * *
C
C ... purpose
C
C      test program for the MUDPACK solver muh2
C
C ... required MUDPACK files
C
C      muh2.f, mudcom.f
C
C
C
*****
* *
C
*****
* *
C
*****
* *
C
C      sample program/test driver for muh2
C
C
*****
** *
C
*****
** *
C
```

```

c
c      a sample program/test driver for muh2 is below. it
can be
c      executed as an initial test. Output is listed for
the test
c      case described. This example illustrates the
added grid
c      size flexibility provided by muh2.
c
c      test the hybrid multigrid/direct method solver
muh2 by approximating
c      the solution to the nonseparable elliptic pde in
divergence form on
c      the 2.5 degree grid on the full surface of a
sphere of radius one.
c
c      div(sigma(t,p)*grad(u(t,p))) -
lambda(t,p)*u(t,p) = f(t,p)
c
c      t and p are colatitude and longitude in radians.
expanding
c      the pde and multiplying thru by sin(t) puts it in
the following
c      form suitable for muh2:
c
c      ctt * utt + cpp*upp + ct*ut + cp *up + ce * u
= r(t,p)
c
c      the coefficients are given by
c
c      ctt(t,p) = sin(t)*sigma(t,p)
c
c      cpp(t,p) = (1/sin(t))*sigma(t,p)
c
c      ct(t,p) = sin(t)*d(sigma)/dt +
cos(t)*sigma(t,p)
c
c      cp(t,p) = (1/sin(t))*d(sigma)/dp
c
c      ce(t,p) = - sin(t)*lambda(t,p)
c
c      r(t,p) = sin(t)* f(t,p)
c
c      for testing use the coefficients and exact

```

```

solution:
c
c      sigma(t,p) = 1.5 + (sin(t)*cos(p))**2
c
c      lambda(t,p) = - sigma(t,p)
c
c      u(t,p) =
[ sin(t)*(sin(t)*cos(t)*sin(p)*cos(p)) ]**2
c
c      (the exact solution is the restriction of the
solution u(x,y,z) =
c      (x*y*z)**2 in cartesian coordinates to the surface
of the sphere
c      in spherical coordinates). assume the solution
u(t,p) is specified
c      at the poles and is periodic in longitude.
choosing the grid size
c      parameters:
c
c      itp = iparm(6) = 9, jpq = iparm(7) = 9
c
c      iet = iparm(8) = 4, jep = iparm(9) = 5
c
c      fits the required five degree 73 by 145 grid
exactly. the 10 x 10
c      coarsest grid is too large for effective error
reduction with multgrid
c      iteration using relaxation only (see tmud2sa.f).
the subgrid sizes
c      in the coarsening muh2 uses are:
c
c      73 X 145 > 37 X 73 > 19 X 37 > 10 X 19 > 10 X
10
c
c      The solvers mud2 or mud2sa would not be efficient
for this coarsening.
c      Guassian elimination is used whenever the coarsest
10 X 10 subgrid
c      is encountered within multigrid iteration and line
relaxation in
c      the phi direction is used at the higher resolution
grids. the
c      default multigrid options are used. one full
multigrid cycle with

```

```
c      no initial guess is executed.  discretization
level error is reached.

c
c
c
*****
*****
c      output (32 bit floating point arithmetic)
c
*****
***

c
c      muh2 test
c
c      integer input parameters
c      intl = 0 nta = 1 ntb = 1 npc = 0 npd = 0
c      itp = 9 jpq = 9 iet = 4 jep = 5
c      nt = 73 np = 145 iguess = 0 maxcy = 1
c      method = 2 work space estimate = 214396
c
c      multigrid option parameters
c      kcycle = 2
c      iprer = 2
c      ipost = 1
c      interpol = 3
c
c      floating point input parameters
c      ta = 0.000 tb = 3.142 pc = 0.000 pd = 6.283
c      tolerance (error control) = 0.000E+00
c
c      discretization call to muh2 intl = 0
c      ierror = 0 minimum work space = 186661
c
c      approximation call to muh2
c      intl = 1 method = 2 iguess = 0 maxcy = 1
c      tolmax = 0.00
c      ierror = 0
c      maximum error = 0.795E-04
c
c
*****
*****
c
c      program tmuh2
```

```

    implicit none
    integer
    iitp,jjpq,iiet,jjep,nnt,nnp,llwrk,lldir,llwork
    integer iitpl,jjpql
c
c      set grid size with parameter statements
c
      parameter (iitp = 9, jjpq = 9, iiet = 4, jjep =
5)
      parameter (iitpl = iitp+1, jjpq1 = jjpq+1)
      parameter ( nnt = iitp*2** (iiet-1) + 1)
      parameter ( nnp = jjpq*2** (jjep-1) + 1)
c
c      set work space estimate (see muh2.d)
c
      parameter (llwrk=4* (15*nnt*nnp+8* (nnt+nnp+2)) /3 )
      parameter (lldir=(2* (iitp+1)* (2*jjpq-1)+jjpq+1))
      parameter (llwork = llwrk+lldir)
c
c      dimension solution, right hand side, and work
arrays
c
      real u(nnt,nnp),r(nnt,nnp),w(llwork)
      integer iw(iitpl,jjpql)
c
c      dimension input argument vectors and set up
continguous storage
c      for labelling entries
c
      integer iparm(17),mgopt(4)
      real fparm(6)
      integer
      intl,nta,ntb,npc,npd,itp,jpq,iet,jep,nt,np,
      +
      iguess,maxcy,method,nwork,lwork,iter
      common / iprm /
      intl,nta,ntb,npc,npd,itp,jpq,iet,jep,nt,np,
      +
      iguess,maxcy,method,nwork,lwork,iter
      real
      ta,tb,pc,pd,tolmax,sinat,cosat,sinap,cosap,dlt,dlp
      common / fprm /
      ta,tb,pc,pd,tolmax,sinat(73),cosat(73),
      +                      sinap(145),cosap(145),dlt,dlp

```

```
integer i,j,ierror
real pi,sint,cost,sinp,cosp
real
ctt,cpp,ct,cp,ce,tmp,dt,dp,dtt,dpp,ue,ut,up,utt,upp
    real errm,p,t
c
c      equivlance iparm,fparm with labelled commons
iprm,fprm
c
        equivalence (intl,iparm)
        equivalence (ta,fparm)
c
c      declare coefficient and boundary condition input
subroutines external
c
        external cof,bndc
c
c      set input integer parameters
c
c      initialization
c
        intl = 0
c
c      set boundary condition flags: poles specified,
longitude periodic
c
        nta = 1
        ntb = 1
        npc = 0
        npd = 0
c
c      set grid sizes from parameter statements
c
        itp = iitp
        jpq = jjpq
        iet = iiet
        jep = jjep
        nt = nnt
        np = nnnp
c
c      full multigrid cycling (no initial guess at finest
grid)
c
        iguess = 0
```

```

C
C      set one multigrid cycle
C
C      maxcy = 1
C
C      set line relaxation in the longitude direction
C
C      method = 2
C
C      set work space estimate
C
C      nwork = llwork
C
C      set multigrid parameters (w(2,1) cycling with
fully weighted
C      residual restriction and cubic prolongation).
this can also
C      be set by inputting mgopt(1) = 0 to muh2
C
C      mgopt(1) = 2
C      mgopt(2) = 2
C      mgopt(3) = 1
C      mgopt(4) = 3
C
C      set floating point input parameters
C
C      pi = 4.0*atan(1.0)
C
C      interval end points (in radians)
C
C      ta = 0.0
C      tb = pi
C      pc = 0.
C      pd = pi+pi
C
C      no error control
C
C      tolmax = 0.0
C
C      set mesh increments
C
C      dlt = (tb-ta)/float(nt-1)
C      dlp = (pd-pc)/float(np-1)
C

```

```

c      preset sin,cos vectors to save computation on grid
points
c
do i=1,nt
t = ta+(i-1)*dlt
sinat(i) = sin(t)
cosat(i) = cos(t)
end do
do j=1,np
p = pc+(j-1)*dlp
sinap(j) = sin(p)
cosap(j) = cos(p)
end do
c
c      initialize right hand side and solution array
except at poles
c
do i=2,nt-1
t = ta+(i-1)*dlt
sint = sinat(i)
cost = cosat(i)
do j=1,np
p = pc+(j-1)*dlp
call cof(t,p,ctt,cpp,ct,cp,ce)
c
c      set intermediate variables for exact solution
c
sinp = sinap(j)
cosp = cosap(j)
tmp = (sint*cosp*sint*sinp*cost)
dt = (2.*sint*cost*cost-sint**3)*(cosp*sinp)
dp = (cosp**2-sinp**2)*(sint**2*cost)
dtt = (2.*cost**3-4.*cost*sint**2-
3.*sint**2*cost)*(cosp*sinp)
dpp = (-4.*cosp*sinp)*(sint**2*cost)
c
c      set continuous solution and partial
derivatives
c
ue = tmp*tmp
ut = 2.*tmp*dt
up = 2.*tmp*dp
utt = 2.* (dt*dt+tmp*dtt)
upp = 2.* (dp*dp+tmp*dpp)

```

```

c
c      set right hand side of continuous pde on grid
c
c      r(i,j) = ctt*utt+cpp*upp+ct*ut+cp*up+ce*ue
c
c      initialize solution array to zero
c
c      u(i,j) = 0.0
c      end do
c      end do
c
c      set u, r(unused) at poles
c
c      do j=1,np
c      u(1,j) = 0.0
c      r(1,j) = 0.0
c      u(nt,j) = 0.0
c      r(nt,j) = 0.0
c      end do
c
c      print input parameters
c
c      write(*,100)
100 format(//' muh2 test' )

      write (*,101) (iparm(i),i=1,15)
101 format(/' integer input parameters ',
      +' intl = ',i2,' nta = ',i2,' ntb = ',i2,' npc =
',i2,' npd = 'i2,
      +' itp = ',i2,' jpq = 'i2,' iet = ',i2,' jep =
',i2
      +' nt = 'i3,' np = 'i3,' iguess = 'i2,' maxcy =
'i2,
      +' method = 'i2, ' work space estimate = ',i7)

      write (*,102) (mgopt(i),i=1,4)
102 format(/' multigrid option parameters ',
      +' kcycle = ',i2,
      +' iprer = ',i2,
      +' ipost = ',i2
      +' interpol = ',i2)

      write(*,103) (fparm(i),i=1,5)
103 format(/' floating point input parameters ',

```

```

      +/- ta = 'f6.3,' tb = 'f6.3,' pc = 'f6.3,' pd =
'f6.3,
      +/- tolerance (error control) =   ' ,e10.3)
C
C      discretization call to muh2
C
      write(*,104) intl
104 format(/' discretization call to muh2 ', ' intl =
',i2)
      call
muh2(iparm,fparm,w,iw,cof,bndc,r,u,mgopt,ierror)
      write (*,105) ierror,iparm(16)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)
C
C      aprroximation call to muh2
C
      intl = 1
      write(*,106) intl, method, iguess, maxcy, tolmax
106 format(/' approximation call to muh2 ',
      +/- intl = ',i2, ' method = ',i2 , ' iguess = ',i2,
'maxcy = ',i2
      +/- tolmax = ',f5.2)
      call
muh2(iparm,fparm,w,iw,cof,bndc,r,u,mgopt,ierror)
      write (*,107) ierror
107 format(' ierror = ',i2 )
      if (ierror.gt.0) call exit(0)
      if (ierror .le. 0) then
C
C      compute and print exact maximum error
C
      errm = 0.0
      do j=1,np
      sinp = sinap(j)
      cosp = cosap(j)
      do i=1,nt
      sint = sinat(i)
      cost = cosat(i)
      ue = (sint*cosp*sint*sinp*cost)**2
      errm = amax1(errm,abs((u(i,j)-ue)))
      end do
      end do

```

```

        write(*,108) errm
108 format(' maximum error = ',e10.3 )
      end if
      end

      subroutine cof(t,p,ctt/cpp,ct,cp,ce)
C
C      coefficient subroutine
C
      implicit none
      real t,p,ctt/cpp,ct,cp,ce
      integer
      intl,nta,ntb,npd,itp,jpq,iet,jep,nt,np,
      +
      iguess,maxcy,method,nwork,lwork,iter
      real
      ta,tb,pc,pd,tolmax,sinat,cosat,sinap,cosap,dlt,dlp
      common / iprm /
      intl,nta,ntb,npd,itp,jpq,iet,jep,nt,np,
      +
      iguess,maxcy,method,nwork,lwork,iter
      common / fprm /
      ta,tb,pc,pd,tolmax,sinat(73),cosat(73),
      +
      sinap(145),cosap(145),dlt,dlp
      integer i,j
      real sinp,cosp,sint,cost,sigma,dsigdt,dsigdp
C
C      set subscripts for current grid point (t,p)
C
      i = int((t-ta)/dlt+0.5)+1
      j = int((p-pc)/dlp+0.5)+1
C
C      avoid poles where solution is specified and
coefficients are not used
C
      if (i.gt. 1 .and. i.lt. nt) then
C
C      set sin,cos at (t,p) from precomputed vectors
C
      sinp = sinap(j)
      cosp = cosap(j)
      sint = sinat(i)
      cost = cosat(i)
C

```

```

c      set sigma and its t,p derivatives
c
c      sigma = 1.5 + (sint*cosp)**2
c      dsigdt = 2.0*cosp*cosp*sint*cost
c      dsigdp = -2.0*sint*sint*cosp*sinp
c
c      set coefficients
c
c      ctt = sint*sigma
c      cpp = sigma/sint
c      ct = sint*dsigdt + cost*sigma
c      cp = dsigdp/sint
c      ce = -sint*sigma
c      return
c      else
c
c      set unused coefs at poles arbitrarily
c
c      ctt = 1.0
c      cpp = 1.0
c      ct = 0.0
c      cp = 0.0
c      ce = 0.0
c      return
c      end if
c      end

subroutine bndc(kbdy,torp,alfa,gbdy)
c
c      this subroutine must be provided as a dummy
c      argument even though
c      there are no mixed derivative b.c.
c
c      return
c      end

subroutine exact(t,p,utt,upp,ut,up,ue)
c
c      the exact solution used is the restriction of
c      u(x,y,z) = (x*y*z)**2
c      in cartesian coordinates to the surface of the
c      sphere of radius one
c      using the standard spherical coordinate transforms
c

```

```

    common / fprm /
ta,tb,pc,pd,tolmax,sinat(73),cosat(73),
+           sinap(145),cosap(145),dlt,dlp
C
C      set subscripts for current grid point (t,p)
C
i = int((t-ta)/dlt+0.5)+1
j = int((p-pc)/dlp+0.5)+1
C
C      set sin,cos from precomputed vectors
C
sinp = sinap(j)
cosp = cosap(j)
sint = sinat(i)
cost = cosat(i)
C
C      set intermediate variables
C
tmp = (sint*cosp*sint*sinp*cost)
dt = (2.*sint*cost*cost-sint**3)*(cosp*sinp)
dp = (cosp**2-sinp**2)*(sint**2*cost)
dtt = (2.*cost**3-4.*cost*sint**2-
3.*sint**2*cost)*(cosp*sinp)
dpp = (-4.*cosp*sinp)*(sint**2*cost)
C
C      set solution and partial derivatives
C
ue = tmp*tmp
ut = 2.*tmp*dt
up = 2.*tmp*dp
utt = 2.* (dt*dt+tmp*dtt)
upp = 2.* (dp*dp+tmp*dpp)
return
end

```

```
C
C      file tmuh24.f
C
C      * * * * *
* * * * *
C      *
*
C      *           copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *           all rights reserved
*
C      *
*
C      *           MUDPACK version 5.0.1
*
C      *
*
C      *           A Fortran Package of Multigrid
*
C      *
*
C      *           Subroutines and Example Programs
*
C      *
*
C      *           for Solving Elliptic Partial Differential
Equations          *
C      *
*
C      *           by
*
C      *
*
C      *           John Adams
*
C      *
```

C \* of  
\*  
C \*  
\*  
C \* the National Center for Atmospheric  
Research \*  
C \*  
\*  
C \* Boulder, Colorado (80307)  
U.S.A. \*  
C \*  
\*  
C \* which is sponsored by  
\*  
C \*  
\*  
C \* the National Science Foundation  
\*  
C \*  
\*  
C \*  
\* \* \* \* \* \* \* \*  
C  
C ... purpose  
C  
C test program for the MUDPACK solver muh24  
C  
C ... required MUDPACK files  
C  
C muh24.f, muh2.f, mudcom.f  
C  
C  
C \*\*\*\*\*  
\*  
C  
\*\*\*\*\*  
\*  
C  
\*\*\*\*\*  
\*  
C  
C sample program/test driver for muh24  
C  
C  
\*\*\*\*\*  
\*\*

```

C
*****
** 
C
C
c      a sample program/test driver for muh24 is below.
it can be
c      executed as an initial test.  Output is listed for
the test
c      case described.  This example illustrates the
added grid
c      size flexibility provided by muh2 and muh24.
c
c      test the hybrid multigrid/direct method solver
muh24 by approximating
c      the solution to the nonseparable elliptic pde in
divergence form on
c      the 2.5 degree grid on the full surface of a
sphere of radius one.
c
c          div(sigma(t,p)*grad(u(t,p))) -
lambda(t,p)*u(t,p) = f(t,p)
c
c      t and p are colatitude and longitude in radians.
expanding
c      the pde and multiplying thru by sin(t) puts it in
the following
c      form suitable for muh2:
c
c          ctt * utt + cpp*upp + ct*ut + cp *up + ce * u
= r(t,p)
c
c      the coefficients are given by
c
c          ctt(t,p) = sin(t)*sigma(t,p)
c
c          cpp(t,p) = (1/sin(t))*sigma(t,p)
c
c          ct(t,p)  = sin(t)*d(sigma)/dt +
cos(t)*sigma(t,p)
c
c          cp(t,p)  = (1/sin(t))*d(sigma)/dp
c
c          ce(t,p)  = - sin(t)*lambda(t,p)

```

```

C
C           r(t,p)    = sin(t)* f(t,p)
C
C       for testing use the coefficients and exact
solution:
C
C           sigma(t,p) = 1.5 + (sin(t)*cos(p) )**2
C
C           lambda(t,p) = - sigma(t,p)
C
C           u(t,p) =
[ sin(t)*(sin(t)*cos(t)*sin(p)*cos(p) ) ]**2
C
C       (the exact solution is the restriction of the
solution u(x,y,z) =
C       (x*y*z)**2 in cartesian coordinates to the surface
of the sphere
C       in spherical coordinates). assume the solution
u(t,p) is specified
C       at the poles and is periodic in longitude.
choosing the grid size
C       parameters:
C
C           itp = iparm(6) = 9, jpq = iparm(7) = 9
C
C           iet = iparm(8) = 4,   jep = iparm(9) = 5
C
C       fits the required five degree 73 by 145 grid
exactly. the 10 x 10
C       coarsest grid is too large for effective error
reduction with multgrid
C       iteration using relaxation only. the subgrid
sizes in the coarsening
C       muh2 and muh24 uses are:
C
C           73 X 145 > 37 X 73 > 19 X 37 > 10 X 19 > 10 X
10
C
C       Guassian elimination is used whenever the coarsest
10 X 10 subgrid
C       is encountered within multigrid iteration and line
relaxation in
C       the phi direction is used at the higher resolution
grids. the

```

```
c      default multigrid options are used. one full
multigrid cycle with
c      no initial guess is executed with muh2.  then, to
ensure a second-
c      order approximation is reached, two more cycles
are executed calling
c      muh2 with iguess = 1.  Due to fortuitous error
cancellations, the
c      additional cycles actually increase the error
measure with the
c      continuous solution but do give a better
approximation to the
c      solution of the second-order finite difference
equations.  Finally
c      muh24 is called to produce a fourth-order
estimate.

c
c
*****
***  

c      output (64 bit floating point arithmetic)
c
*****
***  

c
c      muh2 test
c
c      integer input parameters
c      intl =  0 nta =  1 ntb =  1 npc =  0 npd =  0
c      itp =  9 jpg =  9 iet =  4 jep =  5
c      nt =  73 np = 145 iguess =  0 maxcy =  1
c      method = 2 work space estimate = 214396
c
c      multigrid option parameters
c      kcyle = 2
c      iprer = 2
c      ipost = 1
c      interpol = 3
c
c      floating point input parameters
c      ta =  0.000 tb =  3.142 pc =  0.000 pd =  6.283
c      tolerance (error control) =  0.000E+00
c
c      discretization call to muh2  intl =  0
```

```

c      ierror = 0 minimum work space = 186661
c
c      approximation call to muh2
c      intl = 1 method = 2 iguess = 0 maxcy = 1
c      tolmax = 0.00
c      ierror = 0
c      maximum error = 0.795E-04
c
c      approximation call to muh2
c      intl = 1 method = 2 iguess = 1 maxcy = 2
c      tolmax = 0.00
c      ierror = 0
c      maximum error = 0.839E-04
c
c      muh24 test ierror = 0
c      maximum error = 0.209E-06
c
c
c ****
***
```

c

```

program tmuh24
implicit none
integer
iitp,jjpq,iiet,jjep,nnt,nnp,llwrk,lldir,llwork
    integer iitp1,jjpql
c
c      set grid size with parameter statements
c
parameter (iitp = 9, jjpq = 9, iiet = 4, jjep =
5)
parameter (iitp1 = iitp+1, jjpql = jjpq+1)
parameter ( nnt = iitp*2** (iiet-1) + 1)
parameter ( nnp = jjpq*2** (jjep-1) + 1)
c
c      set work space estimate (see muh2.d)
c
parameter (llwrk=4*(15*nnt*nnp+8*(nnt+nnp+2))/3 )
parameter (lldir=(2*(iitp+1)*(2*jjpq-1)+jjpq+1))
parameter (llwork = llwrk+lldir)
c
c      dimension solution, right hand side, and work
arrays
c
```

```

      real u(nnt,nnp),r(nnt,nnp),w(llwork)
      integer iw(iitpl,jjpql)

C
C      dimension input argument vectors and set up
continguous storage
C      for labelling entries
C
      integer iparm(17),mgopt(4)
      real fparm(6)
      integer
      intl,nta,ntb,npd,npd,itp,jpq,iet,jep,nt,np,
      +
      iguess,maxcy,method,nwork,lwork,iter
      common / iprm /
      intl,nta,ntb,npd,npd,itp,jpq,iet,jep,nt,np,
      +
      iguess,maxcy,method,nwork,lwork,iter
      real
      ta,tb,pc,pd,tolmax,sinat,cosat,sinap,cosap,dlt,dlp
      common / fprm /
      ta,tb,pc,pd,tolmax,sinat(73),cosat(73),
      +
      sinap(145),cosap(145),dlt,dlp
      integer i,j,ierror
      real pi,sint,cost,sinp,cosp
      real
      ctt,cpp,ct,cp,ce,tmp,dt,dp,dtt,dpp,ue,ut,up,utt,upp
      real errm,p,t
C
C      equivlance iparm,fparm with labelled commons
iprm,fprm
C
      equivalence (intl,iparm)
      equivalence (ta,fparm)
C
C      declare coefficient and boundary condition input
subroutines external
C
      external cof,bndc
C
C      set input integer parameters
C
C      initialization
C
      intl = 0

```

```
c
c      set boundary condition flags: poles specified,
longitude periodic
c
    nta = 1
    ntb = 1
    npc = 0
    npd = 0
c
c      set grid sizes from parameter statements
c
    itp = iitp
    jpq = jjpq
    iet = iiet
    jep = jjep
    nt = nnt
    np = nnr
c
c      full multigrid cycling (no initial guess at finest
grid)
c
    iguess = 0
c
c      set one multigrid cycle
c
    maxcy = 1
c
c      set line relaxation in the longitude direction
c
    method = 2
c
c      set work space estimate
c
    nwork = llwork
c
c      set multigrid parameters (w(2,1) cycling with
fully weighted
c      residual restriction and cubic prolongation).
this can also
c      be set by inputting mgopt(1) = 0 to muh2
c
    mgopt(1) = 2
    mgopt(2) = 2
    mgopt(3) = 1
```

```

mgopt(4) = 3
c
c      set floating point input parameters
c
pi = 4.0*atan(1.0)
c
c      interval end points (in radians)
c
ta = 0.0
tb = pi
pc = 0.
pd = pi+pi
c
c      no error control
c
tolmax = 0.0
c
c      set mesh increments
c
dlt = (tb-ta)/float(nt-1)
dlp = (pd-pc)/float(np-1)
c
c      preset sin,cos vectors to save computation on grid
points
c
do i=1,nt
t = ta+(i-1)*dlt
sinat(i) = sin(t)
cosat(i) = cos(t)
end do
do j=1,np
p = pc+(j-1)*dlp
sinap(j) = sin(p)
cosap(j) = cos(p)
end do
c
c      initialize right hand side and solution array
except at poles
c
do i=2,nt-1
t = ta+(i-1)*dlt
sint = sinat(i)
cost = cosat(i)
do j=1,np

```

```

    p = pc+(j-1)*dlp
    call cof(t,p,ctt,cpp,ct,cp,ce)
c
c      set intermediate variables for exact solution
c
    sinp = sinap(j)
    cosp = cosap(j)
    tmp = (sint*cosp*sint*sinp*cost)
    dt = (2.*sint*cost*cost-sint**3)*(cosp*sinp)
    dp = (cosp**2-sinp**2)*(sint**2*cost)
    dtt = (2.*cost**3-4.*cost*sint**2-
3.*sint**2*cost)*(cosp*sinp)
    dpp = (-4.*cosp*sinp)*(sint**2*cost)
c
c      set continuous solution and partial
derivatives
c
    ue = tmp*tmp
    ut = 2.*tmp*dt
    up = 2.*tmp*dp
    utt = 2.* (dt*dt+tmp*dtt)
    upp = 2.* (dp*dp+tmp*dpp)
c
c      set right hand side of continuous pde on grid
c
    r(i,j) = ctt*utt+cpx*upp+ct*ut+cp*up+ce*ue
c
c      initialize solution array to zero
c
    u(i,j) = 0.0
end do
end do
c
c      set u, r(unused) at poles
c
    do j=1,np
    u(1,j) = 0.0
    r(1,j) = 0.0
    u(nt,j) = 0.0
    r(nt,j) = 0.0
    end do
c
c      print input parameters
c

```

```

        write(*,100)
100 format(//' muh2 test' )

        write (*,101) (iparm(i),i=1,15)
101 format(' integer input parameters ',
+' intl = ',i2,' nta = ',i2,' ntb = ',i2,' npc =
',i2,' npd = ',i2,
+' itp = ',i2,' jpq = ',i2,' iet = ',i2,' jep =
',i2
+' nt = ',i3,' np = ',i3,' iguess = ',i2,' maxcy =
',i2,
+' method = ',i2, ' work space estimate = ',i7)

        write (*,102) (mgopt(i),i=1,4)
102 format(' multigrid option parameters ',
+' kcycle = ',i2,
+' iprer = ',i2,
+' ipost = ',i2
+' interpol = ',i2)

        write(*,103) (fparm(i),i=1,5)
103 format(' floating point input parameters ',
+' ta = 'f6.3,' tb = 'f6.3,' pc = 'f6.3,' pd =
'f6.3,
+' tolerance (error control) =  ',e10.3)
C
C      discretization call to muh2
C
        write(*,104) intl
104 format(' discretization call to muh2 ', ' intl =
',i2)
        call
muh2(iparm,fparm,w,iw,cof,bndc,r,u,mgopt,ierror)
        write (*,105) ierror,iparm(16)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
        if (ierror.gt.0) call exit(0)
C
C      aprroximation call to muh2
C
        intl = 1
        write(*,106) intl, method, iguess, maxcy, tolmax
106 format(' approximation call to muh2 ',
+' intl = ',i2, ' method = ',i2 , ' iguess = ',i2,

```

```

' maxcy = ',i2
  +' tolmax = ',f5.2)
  call
muh2(iparm,fparm,w,iw,cof,bndc,r,u,mgopt,ierror)
  write (*,107) ierror
107 format(' ierror = ',i2 )
  if (ierror.gt.0) call exit(0)
  if (ierror .le. 0) then
C
C      compute and print exact maximum error
C
      errm = 0.0
      do j=1,np
        sinp = sinap(j)
        cosp = cosap(j)
        do i=1,nt
          sint = sinat(i)
          cost = cosat(i)
          ue = (sint*cosp*sint*sinp*cost)**2
          errm = amax1(errm,abs((u(i,j)-ue)))
        end do
      end do
      write(*,108) errm
108 format(' maximum error   = ',e10.3 )
      end if
C
C      execute two more cycles with iguess=1 to ensure
second order
C
      maxcy = 2
      iguess = 1
      write(*,106) intl, method, iguess, maxcy, tolmax
      call
muh2(iparm,fparm,w,iw,cof,bndc,r,u,mgopt,ierror)
  write (*,107) ierror
  if (ierror.gt.0) call exit(0)
  if (ierror .le. 0) then
C
C      compute and print exact maximum error
C
      errm = 0.0
      do j=1,np
        sinp = sinap(j)
        cosp = cosap(j)

```

```

      do i=1,nt
        sint = sinat(i)
        cost = cosat(i)
        ue = (sint*cosp*sint*sinp*cost)**2
        errm = amax1(errm,abs((u(i,j)-ue)))
      end do
    end do
    write(*,108) errm
    end if

C
C      attempt to improve approximation to fourth order
C
      call muh24(w,iw,u,ierror)
      write (*,109) ierror
109 format(/' muh24 test ', ' ierror = ',i2)
      if (ierror.gt.0) call exit(0)
      if (ierror .le. 0) then

C
C      compute and print exact maximum error
C
      errm = 0.0
      do j=1,np
        sinp = sinap(j)
        cosp = cosap(j)
        do i=1,nt
          sint = sinat(i)
          cost = cosat(i)
          ue = (sint*cosp*sint*sinp*cost)**2
          errm = amax1(errm,abs((u(i,j)-ue)))
        end do
      end do
      write(*,108) errm
      end if
    end

      subroutine cof(t,p,ctt,cpp,ct,cp,ce)
C
C      coefficient subroutine
C
      implicit none
      real t,p,ctt,cpp,ct,cp,ce
      integer
      intl,nta,ntb,npc,npd,itp,jpq,iet,jep,nt,np,
      +

```

```

iguess,maxcy,method,nwork,lwork,iter
      real
ta,tb,pc,pd,tolmax,sinat,cosat,sinap,cosap,dlt,dlp
      common / iprm /
intl,nta,ntb,npd,itp,jpq,iet,jep,nt,np,
      +
iguess,maxcy,method,nwork,lwork,iter
      common / fprm /
ta,tb,pc,pd,tolmax,sinat(73),cosat(73),
      +           sinap(145),cosap(145),dlt,dlp
      integer i,j
      real sinp,cosp,sint,cost,sigma,dsigdt,dsigdp
C
C      set subscripts for current grid point (t,p)
C
      i = int((t-ta)/dlt+0.5)+1
      j = int((p-pc)/dlp+0.5)+1
C
C      avoid poles where solution is specified and
coefficients are not used
C
      if (i.gt. 1 .and. i.lt. nt) then
C
C      set sin,cos at (t,p) from precomputed vectors
C
      sinp = sinap(j)
      cosp = cosap(j)
      sint = sinat(i)
      cost = cosat(i)
C
C      set sigma and its t,p derivatives
C
      sigma = 1.5 + (sint*cosp)**2
      dsigdt = 2.0*cosp*cosp*sint*cost
      dsigdp = -2.0*sint*sint*cosp*sinp
C
C      set coefficients
C
      ctt = sint*sigma
      cpp = sigma/sint
      ct = sint*dsigdt + cost*sigma
      cp = dsigdp/sint
      ce = -sint*sigma
      return

```

```

    else
c
c      set unused coeffs at poles arbitrarily
c
      ctt = 1.0
      cpp = 1.0
      ct = 0.0
      cp = 0.0
      ce = 0.0
      return
      end if
      end

      subroutine bndc(kbdy,torp,alfa,gbdy)
c
c      this subroutine must be provided as a dummy
argument even though
c      there are no mixed derivative b.c.
c
      return
      end

      subroutine exact(t,p,utt,upp,ut,up,ue)
c
c      the exact solution used is the restriction of
u(x,y,z) = (x*y*z)**2
c      in cartesian coordinates to the surface of the
sphere of radius one
c      using the standard spherical coordinate transforms
c
      common / fprm /
ta,tb,pc,pd,tolmax,sinat(73),cosat(73),
      +           sinap(145),cosap(145),dlt,dlp
c
c      set subscripts for current grid point (t,p)
c
      i = int((t-ta)/dlt+0.5)+1
      j = int((p-pc)/dlp+0.5)+1
c
c      set sin,cos from precomputed vectors
c
      sinp = sinap(j)
      cosp = cosap(j)
      sint = sinat(i)

```

```

cost = cosat(i)
C
C      set intermediate variables
C
tmp = (sint*cosp*sint*sinp*cost)
dt = (2.*sint*cost*cost-sint**3)*(cosp*sinp)
dp = (cosp**2-sinp**2)*(sint**2*cost)
dtt = (2.*cost**3-4.*cost*sint**2-
3.*sint**2*cost)*(cosp*sinp)
dpp = (-4.*cosp*sinp)*(sint**2*cost)
C
C      set solution and partial derivatives
C
ue = tmp*tmp
ut = 2.*tmp*dt
up = 2.*tmp*dp
utt = 2.* (dt*dt+tmp*dtt)
upp = 2.* (dp*dp+tmp*dpp)
return
end

```

## TMUH24CR

```

C
C      file tmuh24cr.f
C
C      * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
*
C      *                               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
```

C \* all rights reserved  
\*  
C \*  
\*  
C \* MUDPACK version 5.0.1  
\*  
C \*  
\*  
C \* A Fortran Package of Multigrid  
\*  
C \*  
\*  
C \* Subroutines and Example Programs  
\*  
C \*  
\*  
C \* Equations \* for Solving Elliptic Partial Differential  
C \*  
\*  
C \* by  
\*  
C \*  
\*  
C \* John Adams  
\*  
C \*  
\*  
C \* of  
\*  
C \*  
\*  
C \* the National Center for Atmospheric  
Research \*  
C \*  
\*  
C \* Boulder, Colorado (80307)  
U.S.A. \*  
C \*  
\*  
C \* which is sponsored by  
\*  
C \*  
\*

```
C      *
      the National Science Foundation
*
C      *
*
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * *
C
C ... purpose
C
C      test program for the mudpack solver muh24cr
C
C ... required MUDPACK files
C
C      muh24cr.f, muh2cr.f, mudcom.f
C
C
*****
* *
C
*****
* *
C
C      sample program/test driver for muh24cr
C
C
*****
** *
C
*****
** *
C
C
C      a sample program/test driver for muh24cr is listed
below. it
c      can be executed as an initial test. the output is
listed
c      for the test case described.
C
C      test muh24cr below by solving the nonseparable
elliptic pde
c      with cross derivative term
C
C      (1.+y**2)*pxx + (1.+x**2)*pyy + 2.*x*y*pxy +
C
```

```

c           y*px + x*py - (x*y)*pe = r(x,y)
c
c       on a grid as close to 60 by 70 as muh24cr size
constraints
c       allow.  the solution region is the unit square.
assume a
c       mixed derivative boundary condition at y=1 of the
form
c
c           -x * dp/dx + (1+x) * dp/dy - x * pe =
gbdyd(x).
c
c       and specified (Dirchlet) boundary conditions
elsewhere.  the
c       exact solution
c
c           p(x,y) = (x*y)**5
c
c       is used to set the right hand side, boundary
conditions, and
c       compute the error.
c
c       red/black gauss-seidel point relaxation is used
along with the
c       the default multigrid options.  Three multigrid
cycles are
c       executed with muh2cr to assure second-order
discretization
c       level error for this problem.  muh24cr is then
called for
c       a fourth-order estimate.  Choosing grid parameters
c
c           ixp = 15, jyq=9, iex=3,jey=4
c
c       yields a 61 by 73 grid.  The grid coarsening is
c
c           61 X 73 > 31 X 37 > 16 X 19 > 16 X 10
c
c       The coarsest 16 X 10 grid has too much resolution
for effective
c       error reduction with relaxation only.  muh24cr
uses a direct
c       method whenever the 16 X 10 grid is encountered
which maintains

```

```
c      multigrid convergence efficiency
c
c ****
c      output (64 bit floating point arithmetic)
c
c ****
c
c      muh2cr test
c
c      integer input arguments
c      intl = 0 nxa = 1 nxb = 1 nyc = 1 nyd = 2
c      ixp = 15 jyq = 9 iex = 3 jey = 5
c      nx = 61 ny = 145 iguess = 0 maxcy = 1
c      method = 0 work space estimate = 149055
c
c      multigrid option arguments
c      kcycle = 2
c      iprer = 2
c      ipost = 1
c      interpol = 3
c
c      floating point input parameters
c      xa = 0.000 xb = 1.000 yc = 0.000 yd = 1.000
c      tolerance (error control) = 0.000E+00
c
c      discretization call to muh2cr intl = 0
c      ierror = 0 minimum work space = 149055
c
c      approximation call to muh2cr
c      intl = 1 method = 0 iguess = 0
c      ierror = 0
c      maximum error = 0.414E-03
c
c      muh24cr test ierror = 0
c      maximum error = 0.202E-05
c
c
c
c ****
**
```

end of output

```
*****
**
```

```

c
program tmuh24cr
implicit none
c
c      set grid size params
c
integer iixp,jjyq,iiex,jjey,nnx,nny,llwork
integer iixp1,jjyq1
parameter (iixp = 15 , jjyq = 9, iiex = 3, jjey =
5 )
parameter(iixp1 = iixp+1, jjyq1 = jjyq+1)
parameter (nnx=iixp*2**(iiex-1)+1,
nny=jjyq*2**(jjey-1)+1)
c
c      set exact minimum work space required (see
tmuh2cr.f)
c
parameter (llwork = 149055)
real phi(nnx,nny),rhs(nnx,nny),work(llwork)
integer iwork(iixp1,jjyq1)
c
c      put integer and floating point argument names in
contiguous
c      storeage for labelling in vectors iprm,fprm
c
integer iprm(16),mgopt(4)
real fprm(6)
integer
intl,nxa,nxb,nyc,nyd,ixp,jyq,iiex,jjey,nx,ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero

common/itmuh2cr/intl,nxa,nxb,nyc,nyd,ixp,jyq,iiex,jjey,nx,
ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero
real xa,xb,yc,yd,tolmax,relmax
common/ftmuh2cr/xa,xb,yc,yd,tolmax,relmax
equivalence(intl,iprm)
equivalence(xa,fprm)
integer i,j,ierror
real
dlx,dly,x,y,cxx,cxy,cyy,cx,cy,ce,pxx,pxy,pyy,px,py,pe,er
rmax

```

```
C
c      declare coefficient and boundary condition input
subroutines external
C
      external cofcr,bndcr
C
C      set input integer arguments
C
      intl = 0
C
C      set boundary condition flags
C
      nxn = 1
      nxm = 1
      nyc = 1
      nyd = 2
C
C      set grid sizes from parameter statements
C
      ixp = iixp
      jyq = jjyq
      iex = iiex
      jey = jjey
      nx = nnx
      ny = nny
C
C      set multigrid arguments (w(2,1) cycling with fully
weighted
C      residual restriction and cubic prolongation)
C
      mgopt(1) = 2
      mgopt(2) = 2
      mgopt(3) = 1
      mgopt(4) = 3
C
C      set three cycles
C
      maxcy = 3
C
C      set no initial guess forcing full multigrid
cycling
C
      iguess = 0
```

```

C
c      set work space length approximation from parameter
statement
C
nwork = llwork
C
c      set point relaxation
C
method = 0
C
c      set end points of solution rectangle in (x,y)
space
C
xa = 0.0
xb = 1.0
yc = 0.0
yd = 1.0
C
c      set mesh increments
C
dlx = (xb-xa)/float(nx-1)
dly = (yd-yc)/float(ny-1)
C
c      set for no error control flag
C
tolmax = 0.0
C
c      set right hand side in rhs
c      initialize phi to zero
C
do i=1,nx
x = xa+float(i-1)*dlx
do j=1,ny
y = yc+float(j-1)*dly
call cofcr(x,y,cxx,cxy,cyy,cx,cy,ce)
call exacr(x,y,pxx,pxy,pyy,px,py,pe)
rhs(i,j) =
cxx*pxx+cxy*pxy+cyy*pyy+cx*px+cy*py+ce*pe
phi(i,j) = 0.0
end do
end do
C
c      set specified boundaries in phi at x=xa,xb and
y=yc

```

```

c
    do j=1,ny
        y = yc+float(j-1)*dly
        call exacr(xa,y,pxx,pxy,pyy,px,py,pe)
        phi(1,j) = pe
        call exacr(xb,y,pxx,pxy,pyy,px,py,pe)
        phi(nx,j) = pe
    end do
    do i=1,nx
        x = xa+float(i-1)*dlx
        call exacr(x,yc,pxx,pxy,pyy,px,py,pe)
        phi(i,1) = pe
    end do
    write(*,100)
100 format(//' muh2cr test ')
    write (*,101) (iprm(i),i=1,15)
101 format(/' integer input arguments ',
    +' intl = ',i2,' nxa = ',i2,' nxn = ',i2,' nyc =
',i2,' nyd = ',i2,
    +' ixp = ',i2,' jyq = ',i2,' iex = ',i2,' jey =
',i2
    +' nx = ',i3,' ny = ',i3,' iguess = ',i2,' maxcy =
',i2,
    +' method = ',i2, ' work space estimate = ',i7)
    write (*,102) (mgopt(i),i=1,4)
102 format(/' multigrid option arguments ',
    +' kcycle = ',i2,
    +' iprer = ',i2,
    +' ipost = ',i2
    +' interpol = ',i2)
    write(*,103) xa,xb,yc,yd,tolmax
103 format(/' floating point input parameters ',
    +' xa = ',f6.3,' xb = ',f6.3,' yc = ',f6.3,' yd =
',f6.3,
    +' tolerance (error control) =      ',e10.3)
c
c      initialization call
c
    write(*,104) intl
104 format(/' discretization call to muh2cr', ' intl =
', i2)
    call
muh2cr(iprm,fprm,work,iwork,cofc,rndcr,rhs,phi,mgopt,ie
rror)

```

```

        write (*,105) ierror,iprm(16)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
        if (ierror.gt.0) call exit(0)
C
C      attempt solution
C
        intl = 1
        write(*,106) intl,method,iguess,maxcy
106 format(/' approximation call to muh2cr',
+/ ' intl = ',i2, ' method = ',i2,' iguess = ',i2, '
maxcy = ',i2)
        call
muh2cr(iprm,fprm,work,iwork,cofcr,bndcr,rhs,phi,mgopt,ie
rror)
        write (*,107) ierror
107 format(' ierror = ',i2)
        if (ierror.gt.0) call exit(0)
C
C      compute and print maximum norm of error
C
        errmax = 0.0
        do j=1,ny
y = yc+(j-1)*dly
        do i=1,nx
            x = xa+(i-1)*dlx
            call exacr(x,y,pxx,pxy,pyy,px,py,pe)
            errmax = amax1(errmax,abs((phi(i,j)-pe)))
        end do
        end do
        write(*,108) errmax
108 format(' maximum error = ',e10.3)
C
C      attempt fourth-order
C
        call muh24cr(work,iwork,cofcr,bndcr,phi,ierror)
        write (*,109) ierror
109 format(/' muh24cr test ', ' ierror = ',i2)
        if (ierror.gt.0) call exit(0)
C
C      compute and print maximum norm of error
C
        errmax = 0.0
        do j=1,ny

```

```

y = yc+(j-1)*dly
do i=1,nx
  x = xa+(i-1)*dlx
  call exacr(x,y,pxx,pxy,pyy,px,py,pe)
  errmax = amax1(errmax,abs((phi(i,j)-pe)))
end do
end do
write(*,108) errmax
end

subroutine cofcr(x,y,cxx,cxy,cyy,cx,cy,ce)
c
c      input pde coefficients at any grid point (x,y) in
the solution region
c      (xa.le.x.le.xb,yc.le.y.le.yd) to muh2cr
c
implicit none
real x,y,cxx,cxy,cyy,cx,cy,ce
cxx = 1.+y**2
cxy = 2.*x*y
cyy = 1.+x**2
cx = y
cy = x
ce = -(x*y)
return
end

subroutine bndcr(kbdy,xory,alfa,beta,gama,gbdy)
c
c      input mixed "oblique" derivative b.c. to muh2cr
c      at upper y boundary
c
implicit none
integer kbdy
real xory,alfa,beta,gama,gbdy
integer
intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero

common/itmuh2cr/intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,
ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero

```

```

real xa,xb,yc,yd,tolmax,relmax
common/ftmuh2cr/xa,xb,yc,yd,tolmax,relmax
real x,y,pxx,pxy,pyy,px,py,pe
if (kbdy.eq.4) then
c
c      y=yd boundary (nyd must equal 2 if this code is to
be executed).
c      b.c. has the form
alfyd(x)*px+betyd(x)*py+gamyd(x)*pe = gbdy(x)
c      where x = y or x. alfa,beta,gama,gbdy
corresponding to alfyd(x),
c      betyd(x),gamyd(x),gbdy(y) must be output.
c
y = yd
x = x or y
alfa = -x
beta = 1.+x
gama = -x
call exacr(x,y,pxx,pxy,pyy,px,py,pe)
gbdy = alfa*px + beta*py + gama*pe
return
end if
end

subroutine exacr(x,y,pxx,pxy,pyy,px,py,pe)
c
c      this subroutine is used for setting an exact
solution
c      to test subroutine muh2cr.
c
implicit none
real x,y,pxx,pxy,pyy,px,py,pe
pe = (x*y)**5
px = 5.* (x*y)**4*y
py = 5.* (x*y)**4*x
pxx = 20.* (x*y)**3*y*y
pxy = 25.* (x*y)**4
pyy = 20.* (x*y)**3*x*x
return
end

```

## TMUH2CR

```
C
C      file tmuh2cr.f
C
C      * * * * * * * * * * * * * * *
* * * * * * *
C      *
*
C      *               copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *               all rights reserved
*
C      *
*
C      *               MUDPACK version 5.0.1
*
C      *
*
C      *               A Fortran Package of Multigrid
*
C      *
*
C      *               Subroutines and Example Programs
*
C      *
*
C      *               for Solving Elliptic Partial Differential
Equations          *
C      *
*
C      *               by
*
C      *
*
C      *               John Adams
*
```

```
C      *
*
C      *                                of
*
C      *
*
C      *            the National Center for Atmospheric
Research           *
C      *
*
C      *            Boulder, Colorado (80307)
U.S.A.          *
C      *
*
C      *            which is sponsored by
*
C      *
*
C      *            the National Science Foundation
*
C      *
*
C      *      * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C
C ... purpose
C
C      test program for the mudpack solver muh2cr
C
C ... required MUDPACK files
C
C      muh2cr.f, mudcom.f
C
C
*****
* *
C
*****
* *
C
C      sample program/test driver for muh2cr
C
C
*****
```

```

**
C
*****
**
C
C
C      a sample program/test driver for muh2cr is listed
below. it
C      can be executed as an initial test. the output is
listed
C      for the test case described.
C
C      test muh2cr below by solving the nonseparable
elliptic pde
C      with cross derivative term
C
C      (1.+y**2)*pxx + (1.+x**2)*ppy + 2.*x*y*pxy +
C
C      y*px + x*py - (x*y)*pe = r(x,y)
C
C      on a grid as close to 60 by 70 as muh2cr size
constraints
C      allow. the solution region is the unit square.
assume a
C      mixed derivative boundary condition at y=1 of the
form
C
C      -x * dp/dx + (1+x) * dp/dy - x * pe =
gbdyd(x) .
C
C      and specified (Dirchlet) boundary conditions
elsewhere. the
C      exact solution
C
C      p(x,y) = (x*y)**5
C
C      is used to set the right hand side, boundary
conditions, and
C      compute the error.
C
C      red/black gauss-seidel point relaxation is used
along with the
C      the default multigrid options. one full multigrid
cycle reaches

```

```

c      discretization level error for this problem.
Choosing grid
c      size parameters
c
c      ixp = 15, jyq=9, iex=3,jey=4
c
c      yields a 61 by 73 grid.  The grid coarsening is
c
c      61 X 73 > 31 X 37 > 16 X 19 > 16 X 10
c
c      The coarsest 16 X 10 grid has too much resolution
for effective
c      error reduction with relaxation only.  muh2cr uses
a direct
c      method whenever the 16 X 10 grid is encountered
which maintains
c      multigrid convergence efficiency
c
c ****
c      output (32 bit floating point arithmetic)
c
c ****
c
c      muh2cr test
c
c      integer input arguments
c      intl = 0 nxa = 1 nxb = 1 nyc = 1 nyd = 2
c      ixp = 15 jyq = 9 iex = 3 jey = 5
c      nx = 61 ny = 145 iguess = 0 maxcy = 1
c      method = 0 work space estimate = 151335
c
c      multigrid option arguments
c      kcycle = 2
c      iprer = 2
c      ipost = 1
c      interpol = 3
c
c      floating point input parameters
c      xa = 0.000 xb = 1.000 yc = 0.000 yd = 1.000
c      tolerance (error control) = 0.000E+00
c
c      discretization call to muh2cr intl = 0
c      ierror = 0 minimum work space = 149055
c

```

```

c      approximation call to muh2cr
c      intl = 1 method = 0 iguess = 0
c      ierror = 0
c      maximum error = 0.412E-03
c
c
c ****
**  

c      end of output
c
c ****
**  

c
c      program tmuh2cr
c      implicit none
c
c      set grid size params
c
c      integer iixp,jjyq,iiex,jjey,nnx,nny,llwork
c      integer iixp1,jjyq1
c      parameter (iixp = 15 , jjyq = 9, iiex = 3, jjey =
5 )
c      parameter(iixp1 = iixp+1, jjyq1 = jjyq+1)
c      parameter (nnx=iixp*2** (iiex-1)+1,
c      nny=jjyq*2** (jjey-1)+1)
c
c      estimate work space for point relaxation (see
muh2cr.d)
c
c      parameter (llwork=(7* (nnx+2) * (nny+2)+44*nnx*nny) /3
)
c      real phi(nnx,nny),rhs(nnx,nny),work(llwork)
c      integer iwork(iixp1,jjyq1)
c
c      put integer and floating point argument names in
contiguous
c      storeage for labelling in vectors iprm,fprm
c
c      integer iprm(16),mgopt(4)
c      real fprm(6)
c      integer
intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero

```

```
common/itmuh2cr/intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,
ny,
+
iguess,maxcy,method,nwork,lwrkqd,itero
    real xa,xb,yc,yd,tolmax,relmax
    common/ftmuh2cr/xa,xb,yc,yd,tolmax,relmax
    equivalence(intl,iprm)
    equivalence(xa,fprm)
    integer i,j,ierror
    real
dlx,dly,x,y,cxx,cxy,cyy,cx,cy,ce,pxx,pxy,pyy,px,py,pe,er
rmax
c
c      declare coefficient and boundary condition input
subroutines external
c
        external cofcr,bndcr
c
c
c      set input integer arguments
c
        intl = 0
c
c      set boundary condition flags
c
        nxa = 1
        nxb = 1
        nyc = 1
        nyd = 2
c
c      set grid sizes from parameter statements
c
        ixp = iixp
        jyq = jjyq
        iex = iiex
        jey = jjey
        nx = nnx
        ny = nny
c
c      set multigrid arguments (w(2,1) cycling with fully
weighted
c      residual restriction and cubic prolongation)
c
```

```

mgopt(1) = 2
mgopt(2) = 2
mgopt(3) = 1
mgopt(4) = 3
c
c      set for one cycle
c
c      maxcy = 1
c
c      set no initial guess forcing full multigrid
cycling
c
c      iguess = 0
c
c      set work space length approximation from parameter
c statement
c
c      nwork = llwork
c
c      set point relaxation
c
c      method = 0
c
c      set end points of solution rectangle in (x,y)
c space
c
c      xa = 0.0
c      xb = 1.0
c      yc = 0.0
c      yd = 1.0
c
c      set mesh increments
c
c      dlx = (xb-xa)/float(nx-1)
c      dly = (yd-yc)/float(ny-1)
c
c      set for no error control flag
c
c      tolmax = 0.0
c
c      set right hand side in rhs
c      initialize phi to zero
c
c      do i=1,nx

```

```

x = xa+float(i-1)*dlx
do j=1,ny
    y = yc+float(j-1)*dly
    call cofcr(x,y,cxx,cxy,cyy,cx,cy,ce)
    call exacr(x,y,pxx,pxy,pyy,px,py,pe)
    rhs(i,j) =
cxx*pxx+cxy*pxy+cyy*pyy+cx*px+cy*py+ce*pe
    phi(i,j) = 0.0
end do
end do

c
c      set specified boundaries in phi at x=xa,xb and
y=yc
c
do j=1,ny
y = yc+float(j-1)*dly
call exacr(xa,y,pxx,pxy,pyy,px,py,pe)
phi(1,j) = pe
call exacr(xb,y,pxx,pxy,pyy,px,py,pe)
phi(nx,j) = pe
end do
do i=1,nx
x = xa+float(i-1)*dlx
call exacr(x,yc,pxx,pxy,pyy,px,py,pe)
phi(i,1) = pe
end do
write(*,100)
100 format(//' muh2cr test ')
        write (*,101) (iprm(i),i=1,15)
101 format(/' integer input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxn = ',i2,' nyc =
',i2,' nyd = ',i2,
+/' ixp = ',i2,' jyq = ',i2,' iex = ',i2,' jey =
',i2
+/' nx = ',i3,' ny = ',i3,' iguess = ',i2,' maxcy =
',i2,
+/' method = ',i2, ' work space estimate = ',i7)
        write (*,102) (mgopt(i),i=1,4)
102 format(/' multigrid option arguments ',
+/' kcycle = ',i2,
+/' iprer = ',i2,
+/' ipost = ',i2
+/' interpol = ',i2)
        write(*,103) xa,xb,yc,yd,tolmax

```

```

103 format(/' floating point input parameters ',
+/' xa = ',f6.3,' xb = ',f6.3,' yc = ',f6.3,' yd =
',f6.3,
+/' tolerance (error control) =    ',e10.3)
C
C      initialization call
C
      write(*,104) intl
104 format(/' discretization call to muh2cr', ' intl =
', i2)
      call
muh2cr(iprm,fprm,work,iwork,cofcr,bndcr,rhs,phi,mgopt,ie
rror)
      write (*,200) ierror,iprm(16)
200 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)
C
C      attempt solution
C
      intl = 1
      write(*,106) intl,method,iguess
106 format(/' approximation call to muh2cr',
+/' intl = ',i2, ' method = ',i2,' iguess = ',i2)
      call
muh2cr(iprm,fprm,work,iwork,cofcr,bndcr,rhs,phi,mgopt,ie
rror)
      write (*,107) ierror
107 format(' ierror = ',i2)
      if (ierror.gt.0) call exit(0)
      if (ierror .le. 0) then
C
C      compute and print maximum norm of error
C
      errmax = 0.0
      do j=1,ny
      y = yc+(j-1)*dly
      do i=1,nx
      x = xa+(i-1)*dlx
      call exacr(x,y,pxx,pxy,pyy,px,py,pe)
      errmax = amax1(errmax,abs((phi(i,j)-pe)))
      end do
      end do
      write(*,201) errmax

```

```

201 format(' maximum error = ',e10.3)
      end if
      end

      subroutine cofcr(x,y,cxx,cxy,cyy,cx,cy,ce)
c
c      input pde coefficients at any grid point (x,y) in
c      the solution region
c      (xa.le.x.le.xb,yc.le.y.le.yd) to muh2cr
c
c      implicit none
      real x,y,cxx,cxy,cyy,cx,cy,ce
      cxx = 1.+y**2
      cxy = 2.*x*y
      cyy = 1.+x**2
      cx = y
      cy = x
      ce = -(x*y)
      return
      end

      subroutine bndcr(kbdy,xory,alfa,beta,gama,gbdy)
c
c      input mixed "oblique" derivative b.c. to muh2cr
c      at upper y boundary
c
c      implicit none
      integer kbdy
      real xory,alfa,beta,gama,gbdy
      integer
      intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,ny,
      +
      iguess,maxcy,method,nwork,lwrkqd,itero

common/itmuh2cr/intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nx,
ny,
      +
      iguess,maxcy,method,nwork,lwrkqd,itero
      real xa,xb,yc,yd,tolmax,relmax
      common/ftmuh2cr/xa,xb,yc,yd,tolmax,relmax
      real x,y,pxx,pxy,pyy,px,py,pe
      if (kbdy.eq.4) then
c
c      y=yd boundary (nyd must equal 2 if this code is to

```

```

be executed) .
c      b.c. has the form
alfyd(x)*px+betyd(x)*py+gamyd(x)*pe = gbdy(x)
c      where x = yorx.    alfa,beta,gama,gbdy
corresponding to alfyd(x),
c      betyd(x),gamyd(x),gbdy(x) must be output.
c
      y = yd
      x = xory
      alfa = -x
      beta = 1.+x
      gama = -x
      call exacr(x,y,pxx,pxy,pyy,px,py,pe)
      gbdy = alfa*px + beta*py + gama*pe
      return
      end if
      end

      subroutine exacr(x,y,pxx,pxy,pyy,px,py,pe)
c
c      this subroutine is used for setting an exact
solution
c      to test subroutine muh2cr.
c
      implicit none
      real x,y,pxx,pxy,pyy,px,py,pe
      pe = (x*y)**5
      px = 5.* (x*y)**4*y
      py = 5.* (x*y)**4*x
      pxx = 20.* (x*y)**3*y*y
      pxy = 25.* (x*y)**4
      pyy = 20.* (x*y)**3*x*x
      return
      end

```

```
C
C      file tmuh3.f
C
C      * * * * *
* * * * *
C      *
*
C      *           copyright (c) 2008 by UCAR
*
C      *
*
C      *           University Corporation for Atmospheric
Research          *
C      *
*
C      *           all rights reserved
*
C      *
*
C      *           MUDPACK version 5.0.1
*
C      *
*
C      *           A Fortran Package of Multigrid
*
C      *
*
C      *           Subroutines and Example Programs
*
C      *
*
C      *           for Solving Elliptic Partial Differential
Equations          *
C      *
*
C      *           by
*
C      *
*
C      *           John Adams
*
C      *
```

C \* of  
\*  
C \*  
\*  
C \* the National Center for Atmospheric  
Research \*  
C \*  
\*  
C \* Boulder, Colorado (80307)  
U.S.A. \*  
C \*  
\*  
C \* which is sponsored by  
\*  
C \*  
\*  
C \* the National Science Foundation  
\*  
C \*  
\*  
C \*  
\* \* \* \* \* \* \* \*  
C  
C ... purpose  
C  
C test program for the MUDPACK solver muh3  
C  
C ... required MUDPACK files  
C  
C muh3.f, mudcom.f, mud3ln.f, mud3pn.f  
C  
C  
C \*\*\*\*\*  
\*  
C  
\*\*\*\*\*  
\*  
C  
\*\*\*\*\*  
\*  
C  
C sample program/test driver for muh3  
C  
C  
\*\*\*\*\*  
\*\*

```

C
*****
** 
C
C
C      a sample program/test driver for muh3 is below. it
can be
C      executed as an initial test.  the output is listed
for the
C      test case described.
C
C      test the driver below by solving the nonseparable
3-d elliptic pde
C
C      pxx+pyy+pzz-y*z*px-x*z*py-x*y*pz-x*y*z*pe =
r(x,y,z)
C
C      on a 41 by 21 by 57 grid superimposed on the
(x,y,z) region
C
C      [0.5,1.0] X [1.0,2.0] X [0.25,0.75]
C
C      the solution is specified at x=0.5,1.0 and
y=1.0,2.0 and there are
C      mixed derivative conditions of the form
C
C      dp/dz - (x*y)*p = g(x,z) at z=0.25
C
C      dp/dz + (x*y)*p = h(x,y) at z=0.75
C
C      one full multigrid cycle using the default
multigrid options
C      and line relaxation in the z direction is
executed.  The
C      exact solution
C
C      pe(x,y,z) = exp(x*y*z)
C
C      is used to set the right hand side of the pde,
boundary conditions
C      and compute error.  Choosing grid arguments
C
C      ixp = 5, jyq = 5, kzr = 7, iex = 4, jey = 3, kez
= 42

```

```

C
C      gives the grid coarsening
C
C      41 X 21 X 57 > 21 X 11 X 29 > 11 X 6 X 15 > 6 X
6 X 8
C
C      The coarsest 6 by 6 by 8 grid has too many points
for effective
C      error reduction with relaxation alone. muh3
utilizes a direct
C      method whenever this grid is encountered within
multigrid cycling
C      thus maintaining rapid multigrid convergence,
C
C ****
C      output (32 bit floating point arithmetic)
C ****
C
C      muh3 test
C
C      input arguments
C      intl = 0 nxa = 1 nxb = 1 nyc = 1 nyd = 1
C      nze = 2 nzf = 2
C      ixp = 5 jyq = 5 kzs = 7
C      iex = 4 jey = 3 kez = 4
C      nx = 41 ny = 21 nz = 57 iguess = 0 maxcy = 1
C      method = 3 work space length input = 779587
C      xa = 0.50 xb = 1.00
C      yc = 1.00 yd = 2.00
C      ze = 0.25 zf = 0.75
C      tolmax = 0.000E+00
C
C      multigrid options
C      kcyle = 2
C      iprer = 2
C      ipost = 1
C      interpol = 3
C
C      discretization call to muh3 intl = 0
C      ierror = 0 minimum work space = 776711
C
C      approximation call to muh3

```

```

c      intl = 1 method = 3 iguess = 0 maxcy = 1
c      ierror = 0
c      maximum error = 0.193E-04
c
c
***** *****
c      end of output
c
*****
c
c      program tmuh3
c      implicit none
c
c      set grid sizes with parameter statements
c
c      integer
iixp,jjyq,kkzr,iiex,jjey,kkez,llwork,nnx,nny,nnz
      integer mmx,mmy,mmz,llslld
      parameter(iixp=5,jjyq=5,kkzr=7)
      parameter (mmx=iixp+1,mmy=jjyq+1,mmz=kkzr+1)
      parameter (iiex=4,jjey=3,kkez=4)
      parameter (nnx = iixp*2** (iiex-1)+1)
      parameter (nny = jjyq*2** (jjey-1)+1)
      parameter (nnz = kkzr*2** (kkez-1)+1)
c
c      set work space length estimate for method=3 (see
muh3.d).
c      this will probably overestimate required space
c
      parameter (lls = 13*(nnx+2)*(nny+2)*(nnz+2) )
      parameter (lld = mmx*mmy*mmz*(2*mmx*mmy+1))
      parameter (llwork = lls+lld)
c
c      dimension solution,right hand side, and work
arrays
c
      real
phi(nnx,nny,nnz),rhs(nnx,nny,nnz),work(llwork)
      integer iw(mmx,mmy,mmz)
      integer iprm(23),mgopt(4)
      real fprm(8)
c

```

```

c      put integer and floating point arguments names in
contiguous
c      storeage labelling
c
integer
intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex,jey,
+
kez,nx,ny,nz,iguess,maxcy,method,meth2,nwork,
+
lwrkqd,itero

common/itmud3/intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,i
ex,jey,
+
kez,nx,ny,nz,iguess,maxcy,method,meth2,nwork,
+
lwrkqd,itero

real xa,xb,yc,yd,ze,zf,tolmax,relmax
common/ftmud3/xa,xb,yc,yd,ze,zf,tolmax,relmax
real
dlx,dly,dlz,x,y,z,cxx,cyy,czz,cx,cy,cz,ce,errm
real pxx,pyy,pzz,px,py,pz,pe
integer i,j,k,ierror
equivalence(intl,iprm)
equivalence(xa,fprm)

c
c      declare coefficient and boundary condition input
subroutines external
c
external cof,bndc
c
c      set for initial call
c
intl = 0
c
c      set boundary condition flags
c
nxa = 1
nxb = 1
nyc = 1
nyd = 1
nze = 2
nzf = 2
c
c      set grid sizes from parameter statements

```

```
c
ixp = iixp
jyq = jjyq
kzr = kkzr
iex = iiex
jey = jjey
kez = kkez
nx = nnx
ny = nny
nz = nnz

c
c      set for one multigrid cycles
c
maxcy = 1

c
c      set work space length approximation from parameter
statement
c
nwork = llwork

c
c      set method of relaxation--line in the z direction
c
method = 3

c
c      meth2 only used in planar relaxation--but set
c
meth2 = 0

c
c      set full multigrid cycling by flagging no initial
guess at the finest
c      grid level
c
iguess = 0

c
c      set end points of solution region in (x,y,z) space
c
xa = 0.5
xb = 1.0
yc = 1.0
yd = 2.0
ze = 0.25
zf = 0.75

c
c      set default multigrid options
```

```

c
mgopt(1) = 2
mgopt(2) = 2
mgopt(3) = 1
mgopt(4) = 3
c
c      set mesh increments
c
dlx = (xb-xa)/float(nx-1)
dly = (yd-yc)/float(ny-1)
dlz = (zf-ze)/float(nz-1)
c
c      set for no error control
c
tolmax = 0.0
c
c      set right hand side in rhs and phi to zero
c
do k=1,nz
z = ze+(k-1)*dlz
do j=1,ny
y = yc+float(j-1)*dly
do i=1,nx
x = xa+float(i-1)*dlx
call cof(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
rhs(i,j,k) =
cxx*pxx+cyy*pyy+czz*pzz+cx*px+cy*py+cz*pz+ce*pe
phi(i,j,k) = 0.0
end do
end do
end do
c
c      set specified values at x and y boundaries in phi
c
x = xa
do k=1,nz
z = ze+(k-1)*dlz
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(1,j,k) = pe
end do
end do

```

```

x = xb
do k=1,nz
z = ze+(k-1)*dlz
do j=1,ny
y = yc+float(j-1)*dly
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(nx,j,k) = pe
end do
end do
y = yc
do k=1,nz
z = ze+(k-1)*dlz
do i=1,nx
x = xa+float(i-1)*dlx
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(i,1,k) = pe
end do
end do
y = yd
do k=1,nz
z = ze+(k-1)*dlz
do i=1,nx
x = xa+float(i-1)*dlx
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
phi(i,ny,k) = pe
end do
end do
write(6,50)
50 format(//' muh3 test ')
C
C      print input arguments
C

write(6,100) intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex
,jey,kez,
+
nx,ny,nz,iguess,maxcy,method,nwork,xa,xb,yc,yd,ze,zf,
+
tolmax,mgopt(1),mgopt(2),mgopt(3),mgopt(4)
100 format(/' input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =
',i2,' nyd = ',i2,
+/' nze = ',i2, ' nzf = ',i2,
+/' ixp = ',i2,' jyq = ',i2,' kzr = ',i2,

```

```

+/' iex = ',i2, ' jey = 'i2, ' kez = 'i2,
+/' nx = 'i3,' ny = 'i3,' nz = 'i3, ' iguess =
'i2, ' maxcy = 'i2,
+/' method = 'i2, ' work space length input = ',i7,
+/' xa = 'f5.2,' xb = 'f5.2,
+/' yc = 'f5.2,' yd = 'f5.2,
+/' ze = 'f5.2,' zf = 'f5.2,
+/' tolmax = ' ,e10.3
+/' multigrid options '
+/' kcycle = ',i2
+/' iprer = ',i2
+/' ipost = ',i2
+/' intpol = ',i2 )

C
C      discretize pde
C
      write(*,104) intl
104 format(/' discretization call to muh3', ' intl =
', i2)
      call
muh3(iprm,fprm,work,iw,cof,bndc,rhs,phi,mgopt,ierror)
      write (*,105) ierror,iprm(22)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)
C
C      approximate pde
C
      intl = 1
      write(*,106) intl,method,iguess,maxcy
106 format(/' approximation call to muh3 ',
+/' intl = ',i2, ' method = ',i2,' iguess = ',i2, '
maxcy = ',i2)
      call
muh3(iprm,fprm,work,iw,cof,bndc,rhs,phi,mgopt,ierror)
      write (*,107) ierror
107 format(' ierror = ',i2)
      if (ierror.gt.0) call exit(0)
C
C      compute and print maximum error
C
      call error(nx,ny,nz,phi,errm)
      write(*,108) errm
108 format(' maximum error  =  ',e10.3)

```

```

    end

    subroutine error(nx,ny,nz,phi,errm)
C
C      compute the error in the estimate in phi
C
      implicit none
      integer nx,ny,nz
      real phi(nx,ny,nz),errm
      real xa,xb,yc,yd,ze,zf,tolmax,relmax
      common/ftmud3/xa,xb,yc,yd,ze,zf,tolmax,relmax
      real dlx,dly,dlz,x,y,z,pxx,pyy,pzz,px,py,pz,pe
      integer i,j,k
      dlx = (xb-xa) / (nx-1)
      dly = (yd-yc) / (ny-1)
      dlz = (zf-ze) / (nz-1)
      errm = 0.0
      do k=1,nz
        z = ze+(k-1)*dlz
        do j=1,ny
          y = yc+float(j-1)*dly
          do i=1,nx
            x = xa+float(i-1)*dlx
            call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
            errm = amax1(errm,abs(phi(i,j,k)-pe))
          end do
        end do
        end do
        end do
        return
      end

      subroutine cof(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
C
C      input pde coefficients at grid point (x,y,z) in
C      the solution region
C      to subroutine muh3
C
      implicit none
      real x,y,z,cxx,cyy,czz,cx,cy,cz,ce
      cxx = 1.0
      cyy = 1.0
      czz = 1.0
      cx = -y*z
      cy = -x*z

```

```

cz = -x*y
ce = -(x*y*z)
return
end

subroutine bndc(kbdy,xory,yorz,alfa,gbdy)
C
C      input mixed derivative conditions at z boundaries
to muh3
C
implicit none
integer kbdy
real xory,yorz,alfa,gbdy
real xa,xb,yc,yd,ze,zf,tolmax,relmax
common/ftmud3/xa,xb,yc,yd,ze,zf,tolmax,relmax
real x,y,z,pxx,pyy,pzz,px,py,pz,pe
if (kbdy.eq.5) then
C
C      z = ze (lower z boundary)
C
z = ze
x = xory
y = yorz
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
alfa = -(x*y)
gbdy = pz + alfa*pe
return
end if
if (kbdy.eq.6) then
C
C      z=zf (upper z boundary)
C
z = zf
x = xory
y = yorz
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
alfa = (x*y)
gbdy = pz + alfa*pe
return
end if
end

subroutine exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
C

```

```
c      this subroutine is used to set an exact solution
for testing muh3
c
implicit none
real x,y,z,pe,px,py,pz,pxx,pyy,pzz
pe = exp(x*y*z)
px = y*z*pe
py = x*z*pe
pz = x*y*pe
pxx = (y*z)*px
pyy = (x*z)*py
pzz = (x*y)*pz
return
end
```

TMUH34

```
C
C      file tmuh34f
C
C      * * * * * * * * * * * * * * * * * * * *
* * * * * * * *
C      *
C      *
C      *
C      *           copyright (c) 2008 by UCAR
*
C      *
C      *
C      *           University Corporation for Atmospheric
Research          *
C      *
C      *
C      *           all rights reserved
*
C      *
C      *
C      *           MUDPACK version 5.0.1
*
```



```

C
C ... purpose
C
C      test program for the MUDPACK solver muh34
C
C ... required MUDPACK files
C
C      muh34.f, udcom.f, mud3ln.f, mud3pn.f
C
C
C
C ****
*
C
C ****
*
C
C      sample program/test driver for muh34
C
C
C ****
**
C
C ****
**
C
C      a sample program/test driver for muh3 is below. it
can be
C      executed as an initial test. the output is listed
for the
C      test case described.
C
C      test the driver below by solving the nonseparable
3-d elliptic pde
C
C      pxx+pyy+pzz-y*z*px-x*z*py-x*y*pz-x*y*z*pe =
r(x,y,z)
C
C      on a 41 by 21 by 57 grid superimposed on the
(x,y,z) region
C
C      [0.5,1.0] X [1.0,2.0] X [0.25,0.75]
C

```

```

c      the solution is specified at x=0.5,1.0 and
y=1.0,2.0 and there are
c      mixed derivative conditions of the form
c
c      dp/dz - (x*y)*p = g(x,z) at z=0.25
c
c      dp/dz + (x*y)*p = h(x,y) at z=0.75
c
c      two full multigrid cycle using the default
multigrid options
c      and line relaxation in the z direction is
executed. The
c      exact solution
c
c      pe(x,y,z) = exp(x*y*z)
c
c      is used to set the right hand side of the pde,
boundary conditions
c      and compute error. Choosing grid arguments
c
c      ixp = 5, jyq = 5, kzs = 7, iex = 4, jey = 3, kez
= 42
c
c      gives the grid coarsening
c
c      41 X 21 X 57 > 21 X 11 X 29 > 11 X 6 X 15 > 6 X
6 X 8
c
c      The coarsest 6 by 6 by 8 grid has too many points
for effective
c      error reduction with relaxation alone. muh3
utilizes a direct
c      method whenever this grid is encountered within
multigrid cycling
c      thus maintaining rapid multigrid convergence.
After muh3 is
c      called with two multigrid cycles, muh34 is called
for a fourth
c      order estimate
c
c ****
c      output (64 bit floating point arithmetic)
c
*****

```

```

C
C      muh3 test
C
C      input arguments
C      intl =  0 nxa =  1 nxm =  1 nyc =  1 nyd =  1
C      nze =  2 nzf =  2
C      ixp =  5 jyq =  5 kzm =  7
C      iex =  4 jey =  3 kez =  4
C      nx =  41 ny =  21 nz =  57 iguess =  0 maxcy =  1
C      method = 3 work space length input = 776771
C      xa =  0.50 xb =  1.00
C      yc =  1.00 yd =  2.00
C      ze =  0.25 zf =  0.75
C      tolmax =  0.000E+00
C
C      multigrid options
C      kcycle = 2
C      iprer = 2
C      ipost = 1
C      interpol = 3
C
C      discretization call to muh3 intl = 0
C      ierror = 0 minimum work space = 776711
C
C      approximation call to muh3
C      intl = 1 method = 3 iguess = 0 maxcy = 2
C      ierror = 0
C      maximum error = 0.180E-04
C
C      muh34 test ierror = 0
C      maximum error = 0.179E-05
C
C
C ****
C      end of output
C
C ****
C
C      program tmuh34
C      implicit none
C
C      set grid sizes with parameter statements

```

```

C
      integer
iixp,jjyq,kkzr,iiex,jjey,kkez,llwork,nnx,nny,nnz
      integer mmx,mmy,mmz
      parameter(iixp=5,jjyq=5,kkzr=7)
      parameter (mmx=iixp+1,mmy=jjyq+1,mmz=kkzr+1)
      parameter(iiex=4,jjey=3,kkez=4)
      parameter (nnx = iixp*2** (iiex-1)+1)
      parameter (nny = jjyq*2** (jjey-1)+1)
      parameter (nnz = kkzr*2** (kkez-1)+1)

C
C      set minimal required work space (see tmuh3.f)
C
      parameter (llwork = 776771)

C
C      dimension solution,right hand side, and work
arrays
C
      real
phi(nnx,nny,nnz),rhs(nnx,nny,nnz),work(llwork)
      integer iw(mmx,mmy,mmz)
      integer iprm(23),mgopt(4)
      real fprm(8)

C
C      put integer and floating point arguments names in
contiguous
C      storeage labelling
C
      integer
intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex,jey,
+
kez,nx,ny,nz,iguess,maxcy,method,meth2,nwork,
+
lwrkqd,itero

common/itmud3/intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,i
ex,jey,
+
kez,nx,ny,nz,iguess,maxcy,method,meth2,nwork,
+
lwrkqd,itero

      real xa,xb,yc,yd,ze,zf,tolmax,relmax
      common/ftmud3/xa,xb,yc,yd,ze,zf,tolmax,relmax
      real
dlx,dly,dlz,x,y,z,cxx,cyy,czz,cx,cy,cz,ce,errm

```

```
real pxx,pyy,pzz,px,py,pz,pe
integer i,j,k,ierror
 equivalence(intl,iprm)
 equivalence(xa,fprm)

c
c      declare coefficient and boundary condition input
subroutines external
c
      external cof,bndc
c
c      set for initial call
c
      intl = 0
c
c      set boundary condition flags
c
      nxa = 1
      nxb = 1
      nyc = 1
      nyd = 1
      nze = 2
      nzf = 2
c
c      set grid sizes from parameter statements
c
      ixp = iixp
      jyq = jjyq
      kzs = kkzs
      iex = iiex
      jey = jjey
      kez = kkez
      nx = nnx
      ny = nny
      nz = nnz
c
c      set two multigrid cycles
c
      maxcy = 2
c
c      set work space length approximation from parameter
statement
c
      nwork = llwork
c
```

```

c      set method of relaxation--line in the z direction
c
c      method = 3
c
c      meth2 only used in planar relaxation--but set
c
c      meth2 = 0
c
c      set full multigrid cycling by flagging no initial
c      guess at the finest
c      grid level
c
c      iguess = 0
c
c      set end points of solution region in (x,y,z) space
c
xa = 0.5
xb = 1.0
yc = 1.0
yd = 2.0
ze = 0.25
zf = 0.75
c
c      set default multigrid options
c
mgopt(1) = 2
mgopt(2) = 2
mgopt(3) = 1
mgopt(4) = 3
c
c      set mesh increments
c
dlx = (xb-xa)/float(nx-1)
dly = (yd-yc)/float(ny-1)
dlz = (zf-ze)/float(nz-1)
c
c      set for no error control
c
tolmax = 0.0
c
c      set right hand side in rhs and phi to zero
c
do k=1,nz
z = ze+(k-1)*dlz

```

```

do j=1,ny
    y = yc+float(j-1)*dly
    do i=1,nx
        x = xa+float(i-1)*dlx
        call cof(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
        call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
        rhs(i,j,k) =
cxxx*pxx+cyyy*pyy+czz*pzz+cx*px+cy*py+cz*pz+ce*pe
        phi(i,j,k) = 0.0
    end do
end do
end do
c
c      set specified values at x and y boundaries in phi
c
    x = xa
    do k=1,nz
        z = ze+(k-1)*dlz
        do j=1,ny
            y = yc+float(j-1)*dly
            call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
            phi(1,j,k) = pe
        end do
        end do
        x = xb
        do k=1,nz
            z = ze+(k-1)*dlz
            do j=1,ny
                y = yc+float(j-1)*dly
                call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
                phi(nx,j,k) = pe
            end do
            end do
            y = yc
            do k=1,nz
                z = ze+(k-1)*dlz
                do i=1,nx
                    x = xa+float(i-1)*dlx
                    call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
                    phi(i,1,k) = pe
                end do
                end do
                y = yd
                do k=1,nz

```

```

z = ze+(k-1)*dlz
do i=1,nx
    x = xa+float(i-1)*dlx
    call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
    phi(i,ny,k) = pe
end do
    end do
    write(6,50)
50 format(//' muh3 test ')
C
C      print input arguments
C

write(6,100)intl,nxa,nxb,nyc,nyd,nze,nzf,ixp,jyq,kzr,iex
,jey,kez,
+
nx,ny,nz,iguess,maxcy,method,nwork,xa,xb,yc,yd,ze,zf,
+
tolmax,mgopt(1),mgopt(2),mgopt(3),mgopt(4)
100 format(/' input arguments ',
+/' intl = ',i2,' nxa = ',i2,' nxb = ',i2,' nyc =
',i2,' nyd = ',i2,
+/' nze = ',i2, ' nzf = ',i2,
+/' ixp = ',i2,' jyq = ',i2,' kzr = ',i2,
+/' iex = ',i2, ' jey = ',i2, ' kez = ',i2,
+/' nx = ',i3,' ny = ',i3,' nz = ',i3, ' iguess =
',i2,' maxcy = ',i2,
+/' method = ',i2, ' work space length input = ',i7,
+/' xa = ',f5.2,' xb = ',f5.2,
+/' yc = ',f5.2,' yd = ',f5.2,
+/' ze = ',f5.2,' zf = ',f5.2,
+/' tolmax = ',e10.3
+/' multigrid options '
+/' kcycle = ',i2
+/' iprer = ',i2
+/' ipost = ',i2
+/' interpol = ',i2 )

C
C      discretize pde
C
        write(*,104) intl
104 format(/' discretization call to muh3', ' intl =
', i2)
        call

```

```

muh3(iprm,fprm,work,iw,cof,bndc,rhs,phi,mgopt,ierror)
      write (*,105) ierror,iprm(22)
105 format(' ierror = ',i2, ' minimum work space =
',i7)
      if (ierror.gt.0) call exit(0)
C
C      approximate pde
C
      intl = 1
      write(*,106) intl,method,iguess,maxcy
106 format(/' approximation call to muh3 ',
+/ ' intl = ',i2, ' method = ',i2,' iguess = ',i2, '
maxcy = ',i2)
      call
muh3(iprm,fprm,work,iw,cof,bndc,rhs,phi,mgopt,ierror)
      write (*,107) ierror
107 format(' ierror = ',i2)
      if (ierror.gt.0) call exit(0)
C
C      compute and print maximum error
C
      call error(nx,ny,nz,phi,errm)
      write(*,108) errm
108 format(' maximum error = ',e10.3)
C
C      fourth-order estimate
C
      call muh34(work,iw,phi,ierror)
      write (*,109) ierror
109 format(/' muh34 test', ' ierror = ',i2)
      if (ierror.gt.0) call exit(0)
      call error(nx,ny,nz,phi,errm)
      write(*,108) errm
      end

      subroutine error(nx,ny,nz,phi,errm)
C
C      compute the error in the estimate in phi
C
      implicit none
      integer nx,ny,nz
      real phi(nx,ny,nz),errm
      real xa,xb,yc,yd,ze,zf,tolmax,relmax
      common/ftmud3/xa,xb,yc,yd,ze,zf,tolmax,relmax

```

```

real dlx,dly,dlz,x,y,z,pxx,pyy,pzz,px,py,pz,pe
integer i,j,k
dlx = (xb-xa) / (nx-1)
dly = (yd-yc) / (ny-1)
dlz = (zf-ze) / (nz-1)
errm = 0.0
do k=1,nz
  z = ze+(k-1)*dlz
  do j=1,ny
    y = yc+float(j-1)*dly
    do i=1,nx
      x = xa+float(i-1)*dlx
      call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
      errm = amax1(errm,abs(phi(i,j,k)-pe))
    end do
  end do
  end do
  return
end

subroutine cof(x,y,z,cxx,cyy,czz,cx,cy,cz,ce)
c
c      input pde coefficients at grid point (x,y,z) in
the solution region
c      to subroutine muh3
c
implicit none
real x,y,z,cxx,cyy,czz,cx,cy,cz,ce
cxx = 1.0
cyy = 1.0
czz = 1.0
cx = -y*z
cy = -x*z
cz = -x*y
ce = -(x*y*z)
return
end

subroutine bndc(kbdy,xory,yorz,alfa,gbdy)
c
c      input mixed derivative conditions at z boundaries
to muh3
c
implicit none

```

```

integer kbdy
real xory,yorz,alfa,gbdy
real xa,xb,yc,yd,ze,zf,tolmax,relmax
common/ftmud3/xa,xb,yc,yd,ze,zf,tolmax,relmax
real x,y,z,pxx,pyy,pzz,px,py,pz,pe
if (kbdy.eq.5) then
C
C      z = ze (lower z boundary)
C
z = ze
x = xory
y = yorz
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
alfa = -(x*y)
gbdy = pz + alfa*pe
return
end if
if (kbdy.eq.6) then
C
C      z=zf (upper z boundary)
C
z = zf
x = xory
y = yorz
call exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
alfa = (x*y)
gbdy = pz + alfa*pe
return
end if
end

subroutine exact(x,y,z,pxx,pyy,pzz,px,py,pz,pe)
C
C      this subroutine is used to set an exact solution
for testing muh3
C
implicit none
real x,y,z,pe,px,py,pz,pxx,pyy,pzz
pe = exp(x*y*z)
px = y*z*pe
py = x*z*pe
pz = x*y*pe
pxx = (y*z)*px
pyy = (x*z)*py

```

```
pzz = (x*y) *pz  
return  
end
```

---

## [Return to beginning of this document](#)

MUDPACK, Copyright (C) 2004-2011, Computational Information Systems Laboratory, University Corporation for Atmospheric Research