NISTIR 5948

MGGHAT User's Guide Version 1.1

William F. Mitchell
U. S. Department of Commerce
Technology Administration
National Institute of Standards and Technology
Information Technology Laboratory
Gaithersburg, MD 20899 USA

January 1997

MGGHAT User's Guide Version 1.1

William F. Mitchell
Applied and Computational Mathematics Division
National Institute of Standards and Technology
Gaithersburg, MD 20899 USA

Abstract

MGGHAT (MultiGrid Galerkin Hierarchical Adaptive Triangles) is a FORTRAN program for the solution of general second order linear self-adjoint elliptic partial differential equations with Dirichlet, natural or mixed boundary conditions on 2D polygonal domains (possibly with holes). MGGHAT uses a finite element method with linear, quadratic or cubic elements (user selectable) over triangles. The adaptive refinement via newest vertex bisection and the multigrid iteration are both based on a hierarchical basis formulation. Run time and a posteriori graphical displays are made with gnuplot. This document is a user's guide for MGGHAT. It explains how to obtain, install and use the software.

Contents

1 Introduction

MGGHAT (MultiGrid Galerkin Hierarchical Adaptive Triangles) is a program for the solution of second order linear elliptic partial differential equations (PDEs) of the form

$$-\frac{\partial}{\partial x}(p\frac{\partial u}{\partial x}) - \frac{\partial}{\partial y}(q\frac{\partial u}{\partial y}) + ru = f \text{ in } \Omega \tag{1}$$

with boundary conditions of the form

$$\begin{aligned} u &= g \quad on\ \partial\Omega_1 \\ p\frac{\partial u}{\partial x}\frac{\partial y}{\partial s} - q\frac{\partial u}{\partial y}\frac{\partial x}{\partial s} + cu &= g \quad on\ \partial\Omega_2 \end{aligned} \tag{2}$$

where $p > 0$, $q > 0$, r, f, c and g are functions of x and y, Ω is a polygonal domain in $\Re^2$ (possibly with holes), $\partial\Omega_1 \cup \partial\Omega_2$ is the boundary of Ω, and $\partial/\partial s$ is differentiation with respect to a counterclockwise parameterization of the boundary $(x(s), y(s))$. The first form of the boundary condition is called Dirichlet, and the second form is called mixed. When $c = 0$, the second form is called natural. When $p = q = 1$, the natural boundary condition reduces to the Neumann boundary condition $\partial u/\partial n = g$ where $\partial/\partial n$ is differentiation in the direction of the outward normal. For more information on specifying Neumann boundary conditions, see section 5.2.

MGGHAT is callable as a FORTRAN subroutine, making it useful for user applications in which a major operation is the solution of an elliptic PDE. This includes systems of PDEs, time dependent PDEs and nonlinear PDEs where the user's `main` program contains an iteration loop in which the major operation is the solution of one or more elliptic PDEs.

MGGHAT solves the elliptic PDE by the finite element method. The elements are continuous, but not differentiable, piecewise linear, quadratic or cubic (degree is user selectable) functions over a triangulation of the 2D domain. Given an initial (coarse) triangulation, an adaptive refinement procedure (based on the newest vertex bisection method) provides the properly graded final nonuniform triangulation. The full multigrid (FMG) method is used to solve the linear system of equations. Details of the methods used can be found in [1], [2], [3].

Send any questions, gripes, or praises concerning MGGHAT to mitchell@cam.nist.gov or na.wmitchell@na-net.ornl.gov.

2 Obtaining the Software

2.1 MGGHAT

MGGHAT can be obtained through *netlib*, *mgnet*, and anonymous ftp sites that mirror these sources.

2.1.1 netlib

netlib is a repository of public domain mathematical software. Source code programs can be obtained from *netlib* by email, anonymous ftp, or a WWW browser.

For email, send a message to the *netlib* server `netlib@ornl.gov`. The software will be sent by return email. To learn how to obtain MGGHAT from *netlib*, send the message `send index from pdes/mgghat`. You do not need a `subject` line. To learn more about *netlib*, send the message `send index`.

The anonymous ftp address is `ftp.netlib.org` . MGGHAT is located in `pdes/mgghat`

The URL for access through WWW is `http://www.netlib.org/`

2.1.2 mgnet

mgnet is the MultGrid Network. MGGHAT can be obtained through *mgnet* by anonymous ftp to `casper.cs.yale.edu` in the directory `mgnet/Codes/mgghat` .

2.2 Other Software

2.2.1 LINPACK/BLAS

MGGHAT uses a few routines from the LINPACK and BLAS packages. You may already have these packages installed on your system; consult with your system administrator. If they are available on your system, you should use the installed version since it is likely to be optimized for your particular computer. If they are not already installed, you can obtain the required routines from *netlib* (see section 2.1.1 for information on obtaining software from *netlib*). MGGHAT uses the following routines from LINPACK and BLAS:

LINPACK sgbco.f sgbsl.f ssifa.f ssisl.f

BLAS scopy.f r1mach.f isamax.f saxpy.f sdot.f sscal.f sswap.f

To obtain them by email, send *netlib* the message

```
send sgbco sgbsl ssifa ssisl from linpack

send scopy from blas
```

`r1mach` is included with the MGGHAT distribution, since this version covers more computer types than the version distributed by *netlib*. The other routines from BLAS are used by the LINPACK routines, and will be automatically sent with the LINPACK request.

If you follow the email instructions, you will receive two return mailings, one for the LINPACK request and one for the BLAS request. Each of these should be saved to a file, for example `linpack.sh` and `blas.sh`. Edit the files to remove the mail header, and unpack them with the commands `sh linpack.sh` (or whatever filename you used) and `sh blas.sh`. At this point `scopy.f` will be in the current directory, and two subdirectories (`linpack` and `blas`) will be created with the other routines. Concatenate all the routines into one file called `linpack.f`. For example, use the command `cat linpack/* blas/* scopy.f > linpack.f`.

2.2.2 gnuplot

Graphical support, both run-time and post-processing, for MGGHAT is provided through *gnuplot*. *gnuplot* is a copyrighted but freely distributable no-cost program for 2D and 3D plots of functions and data points. If you do not already have *gnuplot* installed, it can be obtained by anonymous ftp. The official distribution site is `ftp.dartmouth.edu` . Look in the directory `pub/gnuplot`. Official mirrors are `monu1.cc.monash.edu.au` and `irisa.irisa.fr`.

If you choose to use run-time graphics with MGGHAT, you must have *gnuplot* in your executable path. If you do not use graphics at all, you do not need to have *gnuplot* installed.

2.2.3 Tcl/Tk

Graphical displays in MGGHAT can be manipulated with a widget-based menu (but there are alternate ways to select graphical displays; see section 8.1). The widget-based menu is written in Tcl/Tk. Tcl is a "tool command language". Tk is an extension to Tcl that provides an interface to X Windows. *wish* is a simple windowing shell through which the Tcl/Tk program is run.

If you choose to use the widget-based menu for manipulating the graphical displays of MGGHAT, then you must have Tcl/Tk installed and *wish* in your executable path. If you do not use the widget-based menu, you do not need to have Tcl/Tk installed.

On some Linux systems, *wish* is called *wishx*. In this case, you should create a symbolic link from *wish* to *wishx* in a directory that is in your executable path.

If Tcl/Tk is not already installed on your system, the source codes for Tcl and Tk can be obtained by anonymous ftp from `ftp.smli.com` in the directory `pub/tcl`. You should get Tcl version 7.3 and Tk version 3.6.

3 Installation

For a large number of computer systems, installation of MGGHAT should be fairly painless. You will only need to:

1. copy the makefile for your system (for example `makefile.sunos`) to `makefile`
2. edit makefile to indicate where the LINPACK and BLAS routines can be found, as described in section 3.2
3. type `make` to compile a single precision version. On some systems, you can type `make double` to compile an "auto-double" precision version.

On systems for which a makefile is not provided, you will need to modify one of the existing makefiles for your system. This section describes the main system dependencies and parts of the makefile you may need to modify. If you are porting MGGHAT to a new system or have problems with installation, you should also read section 9 on portability.

3.1 System Dependent Routines

MGGHAT was written to be as portable as possible, however there are three routines that are necessarily system dependent: `second`, `system`, and `r1mach`.

`real function second` is used for measuring elapsed CPU time. Since FORTRAN does not have a standard function for measuring elapsed time, this routine is system dependent. The function should return a value such that the difference between two evaluations will provide the intervening user CPU time in seconds. Typically, `second` returns the amount of CPU time since the beginning of the program execution. Three versions are provided with MGGHAT, one of which might work on your system. `second.f` uses the function `etime`, which is available on many UNIX systems. `second.c` interfaces with the C sys include files and function `times`, and should work if you can interface to C properly.

Note, however, that I have encountered systems on which the macro `HZ` (Hertz) is not defined. Finally, second.aix.f is a version that works on the AIX system to which I have access.

`subroutine system` is used to invoke an operating system command. There are two possible problems with this routine. First, the routine itself is not a FORTRAN intrinsic, although it exists on many UNIX versions of FORTRAN. You may need to find an equivalent subroutine on your system, and write a version of `system` that calls that routine. For example, Cray uses `subroutine ishell` for this purpose, so I have provided the file `system.cray.f` with a version of `system` that calls `ishell`. Also, `makefile.cray` is modified to reflect the need to link in this version of `system`.

The second problem is that the operating system commands may differ between systems. MGGHAT assumes the operating system is some variant of UNIX. There are 5 instances of `call system`. *They will be invoked only if the widget based menu for graphics is active.* (In other words, if you do not use the widget based menu, you can provide a dummy routine for `system`.) Four of them issue a `rm` (remove) command; the fifth starts up the Tcl/Tk program *wish*. If your system is not UNIX, you may need to modify the source code to correct these system calls. You can find them by searching for the string `call system` in the file `mgghat.f`.

`real function r1mach` is used for defining constants relevant to the machine dependent arithmetic. Although `r1mach` is actually a BLAS routine, and hence can be obtained from *netlib*, I have included a copy of `r1mach` with MGGHAT. This copy contains the constants for a larger set of machines than the standard versions, including the "auto-double" version for some machines. As with the other BLAS routines, if you have the routine in your system library, you should use that version (see section 3.2). Otherwise, edit `r1mach.f` to select the constants for your machine, and `r1mach8.f` to select the constants for the auto-double version. As provided, these routines are configured for machines with IEEE arithmetic. It is not certain that `r1mach8.f` is correct for all such machines, but for the purposes of MGGHAT the constants are probably close enough.

3.2 Makefile

Several versions of the makefile are provided. The systems they were tested on are:

makefile.aix: xlf 2.3 under AIX 3.2.5 on RS6000

makefile.convex: fc 7.0 under ConvexOS 10.2 on Convex C3820

makefile.cray: cf77 6.0 under UNICOS 7.0.5.1 on Cray Y-MP4E/232

makefile.hpux: f77 09.16 under HP-UX 10.0 on HP 9000 Series 700

makefile.irix: f77 3.5 under IRIX 4.0.5C on SGI R4000-50 VGX

makefile.linux: f2c "tenth edition" and gcc 2.4.5 under SLS 1.03 kernel 0.99p12 on 486DX50

makefile.sunos: f77 1.4 Patch 8 under SunOS 4.1.3 on Sun Sparc 10

For all installations, you need to provide the location of the LINPACK and BLAS routines. This is done through the `LINPACK` and `DLINPACK` variables set near the top of the makefile.

If your system has LINPACK and BLAS installed, you should provide the name of the library containing them, in the format used for linking in a library on your system. On most UNIX systems, if LINPACK is in the library `liblinpack.a` and BLAS is in the library `libblas.a` (ask your system administrator for the actual names of the libraries), you would set
`LINPACK = -llinpack -lblas`
in the makefile.

If your system does not have LINPACK and BLAS installed, then get the source code as described in section 2.2.1, and set
`LINPACK = r1mach.o linpack.o`
to link in your own compiled version. Note that `r1mach` (see section 3.1) is a BLAS routine.

It is also possible to mix the library and source code modes. For example, if you have BLAS in the library `libblas.a` but do not have LINPACK in a library, you can use
`LINPACK = linpack.o -lblas`
to use the library BLAS and source code LINPACK. In this case you should not include the BLAS source code in `linpack.f`.

The variable `DLINPACK` is used the same way as `LINPACK`, except it provides the LINPACK and BLAS for the auto-double version of MGGHAT, on compilers that have an auto-double flag. *NOTE: This is not the double precision version of* LINPACK *and* BLAS. *It is the single precision version compiled with the auto-double flag.* If you use the LINPACK and BLAS source code for the auto-double version of MGGHAT, refer to the object files as `r1mach8.o` and `linpack8.o`.

You might have the LINPACK library installed on your system, but not have an auto-double version of it installed. You can mix the forms used in the makefile, i.e.,

```
LINPACK = -llinpack -lblas
DLINPACK = r1mach8.o linpack8.o
```

The other variables in the makefile that you may need to modify are:

FFLAGS: flags for the FORTRAN compiler

CFLAGS: flags for the C compiler

F77: the FORTRAN compiler

CC: the C compiler

DOUBLE: the flag to invoke auto-double on the FORTRAN compiler

3.3 Compiling

To compile the single precision version of the MGGHAT, simply type `make` or `make mgghat`. On systems that support automatic promotion to double precision, type `make double` to create a double precision version. Any of these commands will create an executable called `mgghat`; to run the program, type `mgghat`. To remove files created by running `make` and `mgghat`, type `make clean`.

4 MAIN program

MGGHAT is callable as a FORTRAN subroutine, so the user must provide a FORTRAN `main` program that calls `mgghat`. (Actually, `mgghat` can be called from a subroutine, but for simplicity, in this section the calling entity will be referred to as the user's `main` program.) The main program should be in the file `user.f`. It can be very simple (as short as two lines: `call mgghat` and `end`) or a complex program that advances through time, loops through systems, and/or iterates on nonlinear equations and changes user parameters between calls. The only important point is that *any program or routine that changes the value of any of the user parameters must contain the statement* `include 'commons'`. All the parameters are passed through common blocks in the file `commons`, so there is no parameter list attached to the `call mgghat` statement.

5 Problem Definition

The elliptic problem to be solved is defined by the user through three subroutines: `pde`, `bcond`, and `inittr`. These subroutines should be in a file called `user.f`. See the files `user.f.*` for examples of these routines, and section 11 for a description of the examples.

5.1 Equation

The elliptic partial differential equation, i.e. Eq. 1, is defined in `subroutine pde(x,y,p,q,r,f)`. The point (x, y) is passed in through the real variables `x` and `y`. The user defines the functions $p(x, y)$, $q(x, y)$, $r(x, y)$, and $f(x, y)$ from Eq. 1 and returns them through the real variables `p, q, r` and `f`. Notice that the form of the differential operator in Eq. 1 has a minus sign in front of the second order terms. You should be very careful to get the signs correct in the expressions for `p, q, r` and `f`.

5.2 Boundary Conditions

The boundary conditions, i.e. Eqs. 2, are defined in `subroutine bcond(x,y,ipiece,c,g,itype)`. The point (x, y) is passed in through the real variables `x` and `y`, and the boundary segment number (see section 5.3) is passed in through the integer variable `ipiece`. The user defines the functions $c(x, y)$ and $g(x, y)$ from Eq. 2 and returns them through the real variables `c` and `g`, and sets the integer variable `itype` to be 1, 2 or 3 to indicate whether the boundary condition is Dirichlet, natural (Neumann), or mixed (Robin), respectively.

With $c(x, y) = 0$, the second form of the boundary condition is the 'natural' boundary condition for the differential operator in Eq. 1. Often, the desired boundary condition is the Neumann condition $\partial u/\partial n = g$. In many cases it is possible to represent the Neumann condition in the form of the natural condition.

If the outward normal to the boundary of the domain makes an angle α with the x-axis, then $\cos(\alpha) = \partial y/\partial s$ and $\sin(\alpha) = -\partial x/\partial s$. The operator for Neumann boundary conditions is

$$\frac{\partial}{\partial n} = \frac{\partial y}{\partial s}\frac{\partial}{\partial x} - \frac{\partial x}{\partial s}\frac{\partial}{\partial y}$$

Contrast this to the operator for the natural boundary conditions

$$p\frac{\partial y}{\partial s}\frac{\partial}{\partial x} - q\frac{\partial x}{\partial s}\frac{\partial}{\partial y}$$

1. If $p(x, y) = q(x, y) = 1$ (Laplace, Poisson or Helmholtz equation), then the natural boundary condition is Neumann.

2. If $p(x, y) = q(x, y)$ but they are not identically 1, multiply the boundary condition by p to get the correct form. For example, with the mixed condition

$$\frac{\partial u}{\partial n} + \tilde{c}u = \tilde{g}$$

the correct form for the mixed natural boundary condition is

$$p\frac{\partial u}{\partial n} + p\tilde{c}u = p\tilde{g}$$

So the coefficient of u is $p\tilde{c}$ and the right hand side is $p\tilde{g}$ and in `subroutine bcond` you set `c =` $p\tilde{c}$ and `g =` $p\tilde{g}$.

3. If $p \neq q$ and the sides of the domain are parallel to the x and y axes, then from the definition of $\partial/\partial s$ in terms of α, the mixed boundary condition can be expressed as

$$-p\frac{\partial u}{\partial x} - p\tilde{c} = -p\tilde{g} \text{ on the left sides} \Rightarrow \quad \mathsf{c} = -p\tilde{c} \text{ and } \mathsf{g} = -p\tilde{g}$$

$$-q\frac{\partial u}{\partial y} - q\tilde{c} = -q\tilde{g} \text{ on the bottoms} \Rightarrow \quad \mathsf{c} = -q\tilde{c} \text{ and } \mathsf{g} = -q\tilde{g}$$

$$p\frac{\partial u}{\partial x} + p\tilde{c} = p\tilde{g} \text{ on the right sides} \Rightarrow \quad \mathsf{c} = p\tilde{c} \text{ and } \mathsf{g} = p\tilde{g}$$

$$q\frac{\partial u}{\partial y} + q\tilde{c} = q\tilde{g} \text{ on the tops} \Rightarrow \quad \mathsf{c} = q\tilde{c} \text{ and } \mathsf{g} = q\tilde{g}$$

4. If $p \neq q$ and the sides are not parallel to the axes, it is not, in general, possible to express Neumann conditions. However, in some special cases it might be possible.

5.3 Domain and Initial Triangulation

The domain Ω is defined by the initial triangulation, and hence can be any polygonal shape. The initial triangulation is defined in `subroutine inittr` which you should place in file `user.f`.

Each side of the polygon is assigned a boundary segment number. This positive integer is sent to the subroutine `bcond` to help identify which form of the boundary condition should be used.

If your domain is rectangular, then you can use the sample `inittr` provided in the files `user.f.*`. The sample routine triangulates the rectangular domain $(ax, bx) \times (ay, by)$ with a uniform grid consisting of `ngridx` $\times$ `ngridy` rectangles cut into triangles along one of the diagonals. You need only edit subroutine `inittr` to define `ax, bx, ay, by, ngridx` and `ngridy` near the beginning. `ngridx` and `ngridy` must be even.

For nonrectangular domains, definition is a little more difficult. The file `inittr.L` contains an example for an L shaped domain. The domain is defined by the initial triangulation. The domain is restricted to be polygonal and all corners of the polygon must be vertices of the initial triangulation. The initial triangulation is specified by providing the coordinates of the vertices and the three vertices of each triangle. This is given by six variables:

`integer nvert` number of vertices

`integer ntri` number of triangles

`real xvert(i)` x coordinate of the i^{th} vertex `real yvert(i)` y coordinate of the i^{th} vertex

`integer vertex(i,j)` i^{th} vertex of the j^{th} triangle. Vertex 3 is called the peak.

`integer neigh(i,j)` negative of boundary segment number for the side opposite the i^{th} vertex of triangle j. Does not need to be set for non-boundary sides.

The following restrictions are imposed on the initial triangulation:

1. Each vertex is always in the same position of every triangle in which it occurs. For example, if vertex i is the first vertex of triangle j (`vertex(1,j)=i`) then it cannot be the second vertex of triangle k (`vertex(2,k)=i`), it would have to be the first vertex (`vertex(1,k)=i`).
2. A vertex which is not a peak (first and second vertices) can be in at most 8 triangles.
3. A vertex which is a peak (third vertices) can be in at most 4 triangles.

6 User Parameters

There are several parameters the user can set to effect the operation of the program. Except for those in 'commons' (section 6.1), these are set by assignment statements in the user's program that calls `mgghat` and passed through common blocks. Note the user must have the statement `include 'commons'` in the program where these parameters are set. All the parameters have default values that are set in block data.

6.1 Parameters in 'commons'

The parameters in the file `commons` are parameters in the sense of the FORTRAN `parameter` statement. Most of them are used for setting array dimensions. These all appear near the top of `commons`, and are changed by editing the file `commons`. The total number of words of memory used is approximately

$$(16r^4 - 48r^3 + 66r^2 - 47r + 47)v$$

where r is `ndord` and v is `ndvert`.

`integer ndvert` is the maximum number of vertices in the grid. This is the most important parameter concerning the amount of memory required. Default: 1000

`integer ndlev` is the maximum number of levels for the adaptive refinement procedure. Default: 40

`integer ndord` is the maximum polynomial order allowed. In the current version of MGGHAT there is no reason to have this larger than 4. If you know you will only be using, for example, the second order method (linear elements, specified by `iorder=2`), then setting `ndord=2` will substantially reduce the memory requirements. Default: 4

`integer ndsave` is the maximum number of refinement steps that will be saved for *gnuplot* convergence files. It has very little effect on memory usage, and the current setting of 100 should be sufficient. Default: 100

`integer ndrow0` and `ndband` are memory allocations for the band matrix storage form of the matrix from the initial grid. `ndrow0` should be as large as the number of nodes in the initial grid, which is bounded by $(r-1)^2 v_0$ where r is the polynomial order (specified by `iorder`) and v_0 is the number of vertices in the initial grid. `ndband` is difficult to estimate for an arbitrary initial triangulation. It depends on the way the vertices are ordered and the polynomial order. For the sample rectangular domain in the example `user.f` files, it is bounded by $(\texttt{iorder}-1)^2(\texttt{ngridy}+2)$, where `ngridy` is defined in the sample `subroutine inittr`. If the values you supply are not large enough, you will get an error message at run time informing you how large they should be. Default: `ndrow0`=100, `ndband`=48

`character tmpdir` is the only parameter not used for dimensioning. This is the name of a directory in which to write temporary files with messages passed between mgghat and the widget–based menu for graphics. *It is recommended that this directory be on a local disk, not on a remote disk served by the network file server, to avoid overloading the network.* If you change it, make sure you also change the declaration `character*5` to have enough characters for the directory name. Default: `/tmp/`

6.2 Termination Criteria

`integer mxvert` Stop before there are this many vertices. Default: ndvert

`integer mxtri` Stop before there are this many triangles. Default: ndtri (=2*ndvert)

`integer mxlev` Stop when there are this many levels. Default: ndlev

`integer mxnode` Stop before there are this many nodes. Default: ndnode (=ndvert*(ndord-1)**2)

`real mxtime` Stop before this many seconds of CPU time. Default: 43200. (12 hours)

`real tol` Stop when the error estimate (an estimate of the energy norm of the error, relative to the norm of the solution) is below this tolerance. Default: 0.

6.3 Output Control

`integer ioutpt` I/O unit for printed output. Default: 6

`integer gpfile` I/O unit for gnuplot files. Default: 4

`integer outlev` Output level, i.e., the amount of printed output. The usable values are:

0 No output, except error messages

1 Header plus summary at end of execution

2 Information after each phase of execution

3 low level of debug information

4 medium level of debug information

5 high level of debug information

Values of 3, 4 and 5 are probably not useful to normal users. Default: 2

`integer gptri` Controls whether or not a data file is written with the triangulation information for post processing with *gnuplot*. If 0, do not write the file; if nonzero write the file.
Default: 0

`integer gpsol` Controls whether or not a data file is written with the solution on a rectangular grid for post processing with *gnuplot*. If 0, do not write the file; if $N > 0$, write a file with the solution on an $N \times N$ grid. Default: 0

`integer gpconv` Controls whether or not a data file is written with convergence information for post processing with *gnuplot*. If 0, do not write the file; if nonzero write the file.
Default: 0

`logical gquiet` Make the graphics be quiet. If true, you will not be prompted for the selection of graphics during initialization. Default: false

`logical menuon` If true, use the widget-based menu for graphics selection. If `gquiet` is false, you also have the opportunity to activate the widget-based menu during initialization.
Default: false

`logical pltsel(100)` This is the plot selections. It determines which graphics choices will be displayed. If `pltsel(i)` is true where `i` is one of the values listed in section 8.1.1, the corresponding plot will be among those displayed. Default: all false

6.4 Problem Definition

`logical nuniq` If true, the user is indicating that the solution of the problem is nonunique, which occurs, for example, when $r(x, y) = 0$ and the boundary conditions are Neumann everywhere. In this case, the parameters `nuniqx, nuniqy,` and `nuniqv` must also be set. Default: false

`real nuniqx` and `nuniqy` The x and y coordinates of a point at which the solution is provided to tie down a nonunique solution. The point must be a vertex in the initial triangulation. Default: 0. for both

`real nuniqv` The value of the solution at the point `(nuniqx,nuniqy)`. Default: 0.

6.5 Method Definition

Except possibly for `unifrm`, the parameters in this section are probably not useful to most users.

`logical unifrm` If true, use a uniform refinement of the grid instead of the adaptive refinement. Default: false

`real mgfreq` Determines how often to switch from the refinement phase to the solution phase. Refinement continues until the number of vertices has been increased by a factor of `mgfreq`. Must be greater than 1. Default: 2.

`integer nu1` and `nu2` The number of (half) red-black Gauss-Seidel iterations to perform before (`nu1`) and after (`nu2`) coarse grid correction. Default: `nu1`=1 and `nu2`=2

`integer ncyc` The number of multigrid cycles in each solution phase. Default: 1

7 Other User Routines

This section describes three more routines that the user must provide (`true, truex` and `truey`) and four routines that are useful to the user (`solut, ssolut, save` and `restor`).

7.1 true, truex and truey

If the true solution to Eq. 1 is known, you can supply it in `function true(x,y)` and the x and y derivatives in `function truex(x,y)` and `function truey(x,y)`, respectively. These function subroutines should be in the file `user.f`. With these functions defined, you can measure the error between the true and computed solutions. If you do not know the true solution, you should use `true = 0.0`, in which case the printed and graphical values of the "error" are the norm of the computed solution.

7.2 solut

`function solut(x,y,iderv,t)` is provided to evaluate the computed solution or its derivatives at any point in the domain. The parameters are:

`real x,y` (input) The point at which to evaluate the solution.

`integer iderv` (input) Which derivative of the solution to evaluate:

1 – $\partial^2 U/\partial x^2$

2 – $\partial^2 U/\partial x \partial y$

3 – $\partial^2 U/\partial y^2$

4 – $\partial U/\partial x$

5 – $\partial U/\partial y$

6 – U

`integer t` (input and output) On input, `t` should be a guess of which triangle the point (x, y) is in. If you do not know, then set it equal to 1 for the first call, and use the returned value for subsequent calls. On output, `t` is the triangle the point is in.

A second version of solut is provided for evaluating solutions that have been saved using the save/restore procedures (see section 7.3). The saved-solution evaluation is performed by `function ssolut(x,y,iderv,t,rwrk,iwrk)` where `x, y, iderv` and `t` are as above, and `rwrk` and `iwrk` contain the saved state, as in section 7.3.

7.3 save/restor

The subroutines `save` and `restor` can be used to save the state of common blocks immediately after a return from `subroutine mgghat`, and restore a saved state, respectively.

In Version 1.1, `save` is useful for programs where more than one elliptic PDE is solved, and the solution of one is used in defining the other (see the 'system of equations' and 'time dependent' examples in section 11). The solution contained in the saved state can be evaluated with the function `ssolut` (see section 7.2).

In Version 1.1 of MGGHAT, `restor` is not very useful since MGGHAT reinitializes the data structures on every call. It is intended to be used in a later version that will have additional capability.

`subroutine save(rwrk,iwrk,lwrk)` copies all data structures in the common blocks into the three arrays `real rwrk`, `integer iwrk`, and `logical lwrk`. These arrays should be dimensioned with the parameters `ndrwrk`, `ndiwrk`, and `ndlwrk`, respectively. These parameters are in '`commons`'.

8 Graphics

Graphical displays are provided through *gnuplot*, so it should be possible to use them on any system that has *gnuplot* installed. If you do not currently have *gnuplot*, see section 2.2.2 for instructions on how to obtain it. MGGHAT interfaces with *gnuplot* through a set of C routines written by Przemek Klosowski of NIST that allow FORTRAN programs to send commands to *gnuplot*. (See the file `gnuplt.c`.)

8.1 Run-time Graphics

There are 29 displays supported as graphics during the execution of MGGHAT. Each display is contained in its own *gnuplot* window. The selection of which displays to produce is made either through the setting of user parameters, terminal input during initialization, or a widget based menu. See section 6.3 for more detail on user parameters that effect graphical displays.

8.1.1 Available Displays

The available displays are of three types: functions, triangulations, and convergence plots. Functions can be plotted as a surface plot (shows the function on a uniform rectangular grid, with hidden lines, see Fig. 1), a contour plot, facets (shows the function surface constructed of triangles from the adaptive grid, see Fig. 2), surface plot with the triangulation in the plane below it (see Fig. 3), and facets with the triangulation in the plane below it. The functions that can be plotted are the computed solution, the true solution, the computed and true solutions on the same graph, and the error. The last four can only be plotted if the true solution is provided by the user (see section 7.1). The triangulation can be displayed by itself (see Fig. 4). Convergence plots are log-log graphs of the number of nodes or cpu time vs. the estimate of the energy norm of the error (relative to the norm of the solution), the true energy norm of the error, the maximum norm (L_∞ norm) of the error, or the error estimate and energy norm together (see Fig. 5). If the true solution is not provided, only the error estimate makes sense.

The actual graphical displays, with their identification number, are:

1 – Computed Solution; Surface

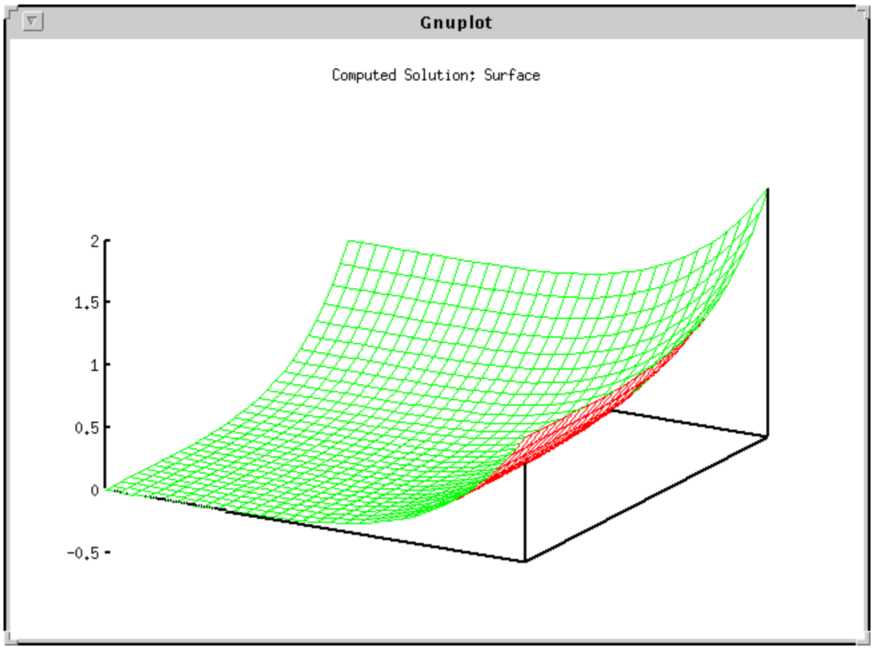

Figure 1: Surface plot.

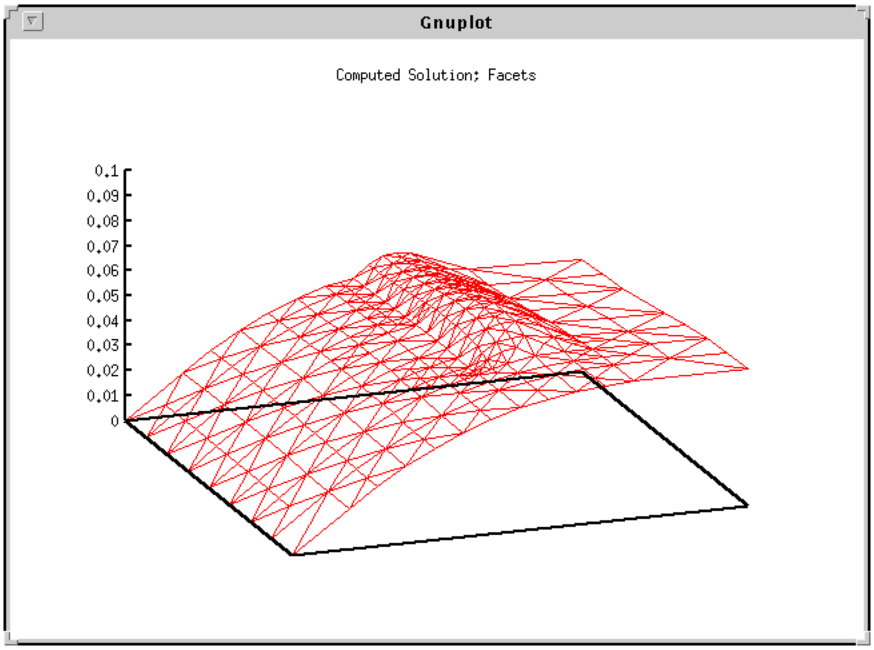

Figure 2: Facet plot.

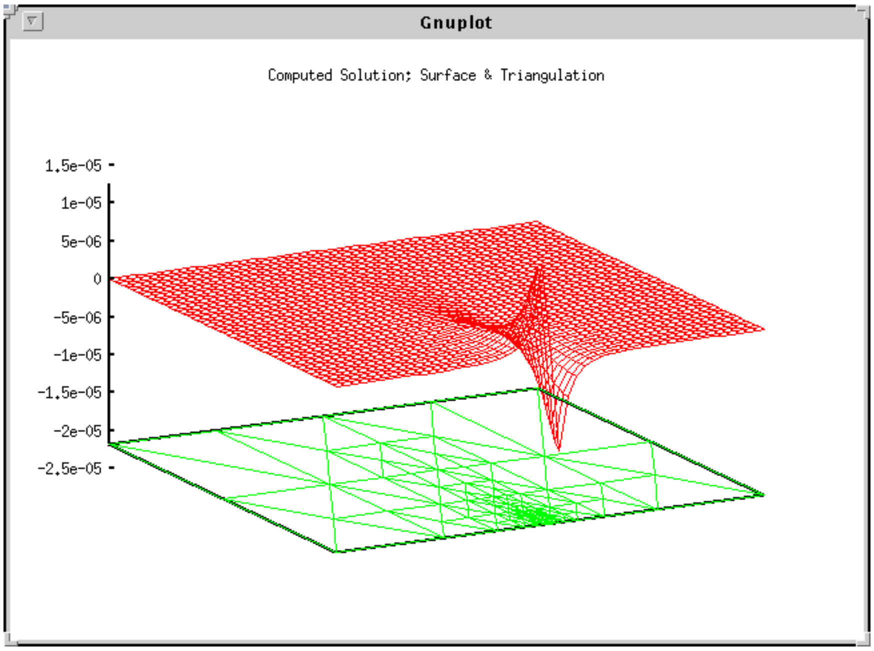

Figure 3: Surface plot with triangulation.

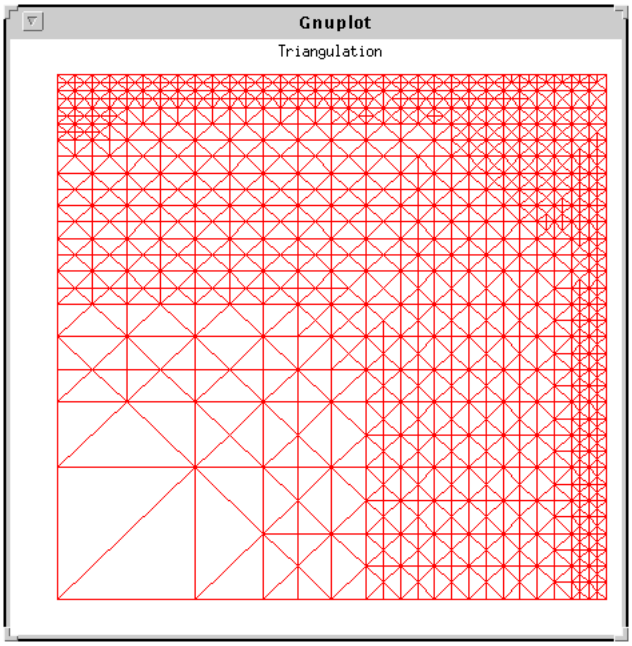

Figure 4: Triangulation.

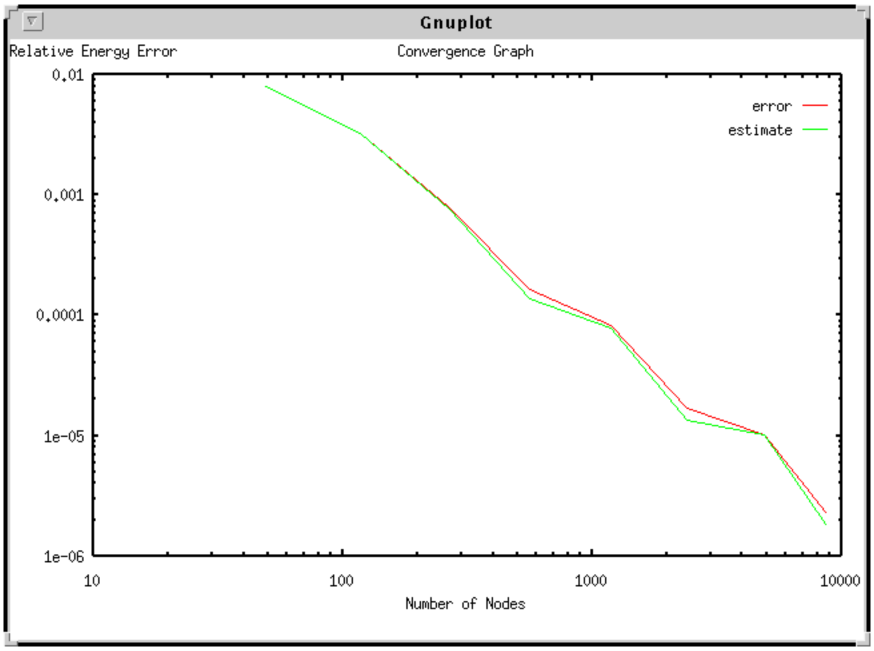

Figure 5: Convergence plot.

2 – Computed Solution; Contour

3 – Computed Solution; Facets

4 – Computed Solution; Surface and Triangulation

6 – Computed Solution; Facets and Triangulation

11 – True Solution; Surface

12 – True Solution; Contour

13 – True Solution; Facets

14 – True Solution; Surface and Triangulation

16 – True Solution; Facets and Triangulation

21 – Computed & True Solutions; Surface

22 – Computed & True Solutions; Contour

23 – Computed & True Solutions; Facets

24 – Computed & True Solutions; Surface and Triangulation

26 – Computed & True Solutions; Facets and Triangulation

31 – Error; Surface

32 – Error; Contour

33 – Error; Facets

34 – Error; Surface and Triangulation

35 – Error; Facets and Triangulation

41 – Triangulation

51 – Convergence; Nodes vs. Energy Error

52 – Convergence; Nodes vs. Maximum Error

53 – Convergence; Nodes vs. Error Estimate

54 – Convergence; Nodes vs. Energy Error and Error Estimate

61 – Convergence; Time vs. Energy Error

62 – Convergence; Time vs. Maximum Error

63 – Convergence; Time vs. Error Estimate

64 – Convergence; Time vs. Energy Error and Error Estimate

8.1.2 User Parameters

The selection of displays can be made directly through the user parameter `pltsel`. This is a logical array for which if the i^{th} entry is `.true.` then display number i of section 8.1.1 will be displayed.

8.1.3 Text Prompt

By default, you will be prompted with yes/no questions to determine if you want any graphics, and if you want to use the widget-based menu. (This prompt can be disabled by setting gquiet ('graphics quiet') to `.true.`) If you choose to have graphics but not use the widget, then a command line menu, equal to the one in section 8.1.1, will be displayed. You can enter any number of displays as a list of integers on one input line. The integers can be separated by any non-digit characters.

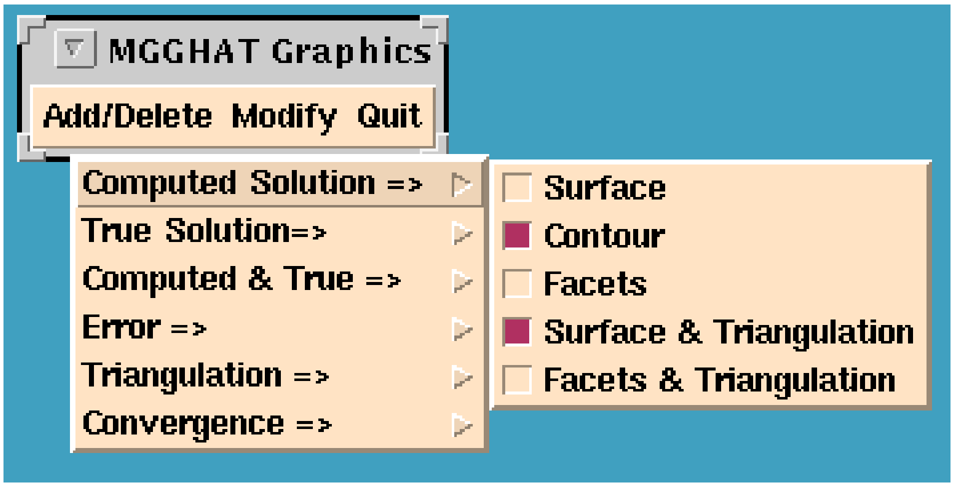

Figure 6: Add/delete widget.

8.1.4 Widget-based Menu

If you have Tcl/Tk installed, you can activate a widget-based menu for control of the graphical displays. This is the only approach that allows you to change the selection of the displays (add more or remove some) while the program is executing. Fig. 6 illustrates the add/delete widgets. The menu also allows you to rotate the view of 3D plots and change the density of isolines in the surface plots (Fig. 7). If `gquiet` is `.false.` (default), you can activate the widget by responding yes to both text prompts. If you have set `gquiet` to be `.true.`, you can activate the menu by setting `menuon` to `.true.` and starting the menu manually with the command `wish < grmenu &`.

8.2 Post-processing Graphics

gnuplot can also be used for graphical displays long after execution has terminated. The user parameters `gptri`, `gpsol`, and `gpconv` can be used to request that data files be produced with information about the triangulation (`gptri.dat`), solution (`gpsol.dat`), and convergence history (`gpconv.dat`), respectively. These files are suitable for input to *gnuplot*. The directory `mgghat/graphics` contains examples of how to input these files into *gnuplot* to produce post-processing graphics. The files of the form `whatever.gps` are *gnuplot* scripts, or command files. They can be invoked by entering the command `load 'whatever.gps'` at the *gnuplot* prompt. The following scripts are provided:

contour.gps Contour plot of the solution.

facets.gps The solution as facets.

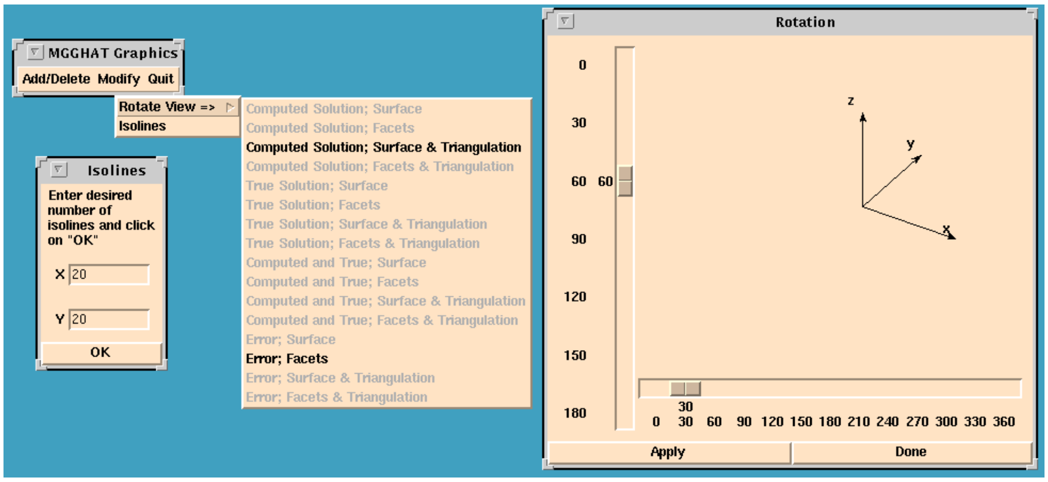

Figure 7: 3D plot view control.

solution.gps Surface plot of the solution.

triangle.gps The triangulation.

sol_and_tri.gps The solution and triangulation on one plot.

converge.gps Convergence graph of nodes vs. energy error and error estimate.

These scripts are intended to be illustrative examples of how to use *gnuplot* for post-processing, and are not all inclusive. They are not necessarily compatible with each other, i.e., you might have to exit *gnuplot* and restart it before loading a second script.

The sample data in this directory is what you should get by running the Poisson example with `iorder = 2`, `gptri = 1`, `gpconv = 1`, and `gpsol = 20`.

9 Portability

MGGHAT is written in (almost) standard conforming FORTRAN 77. The programs *flint* and *ftnchek* were both used to check for compliance with the ANSI FORTRAN 77 standard, and for potential portability problems. There is only one feature of the program that is flagged as not standard conforming – the use of the `include` statement. However, this extension is supported by all compilers that MGGHAT has been tested under. All of the `include` statements are of the form `include 'commons'`. If your compiler does not support the `include` statement, you will need to replace all the `include` statements with the text of the file `commons`.

There are a few other features that may cause portability problems:

1. **double backslash.** In one place a character*1 variable is assigned the string consisting of two backslashes. The second backslash is required because many systems interpret the backslash to mean the next character is special, and two backslashes designate the backslash character. It is not expected that this will cause a problem on systems without this interpretation, because the string of two characters should be truncated to one character when assigned to the character*1 variable, and however the truncation is done it should give a single backslash.

2. **interface to C.** Interfacing between FORTRAN and C is system dependent. There are 3 subroutines in `gnuplt.c` that are written in C and called from FORTRAN. Also, there is one in `second.c`, which you may or may not use (see the discussion about `function second` in section 3.1). There are three versions of `gnuplt.c`; the second one, `gnuplt.no_.c`, differs from the first by not appending an underscore character to the function names. The third version is for Crays. You should use the one that matches the rules of your system. If you are unable to interface between FORTRAN and C on your system, note that these routines are called only if run-time graphics are invoked. If you do not use run-time graphics, you can replace them with a FORTRAN file with 'dummy' subroutines `gpopen`, `gnuplt`, and `gpclos`.

3. **system dependent subprograms.** `subroutine system`, `function second`, and `function r1mach` are not intrinsic functions, and are system dependent. See section 3.1 for configuration of these three routines.

4. **makefile.** The makefile is, of course, system dependent. See section 3.2.

10 Upward Compatibility

If you are using MGGHAT Version 1.0, it should be simple to move to Version 1.1.

1. Concatenate the three files `main.f`, `prob.f`, and `inittr.f` into a single file called `user.f`.

2. If you have your own version of `second.f`, use it.

3. If you assign graphics selections in the main program, change the assignment to match the new graphics manipulation (see section 8.1).

4. The assignment of parameter values in the main program is no longer necessary (if default values are used), and they can be removed. On the other hand, it doesn't hurt to leave them there.

11 Examples

MGGHAT comes with three example user files: a Poisson solver, a system of equations, and a time dependent problem. The first is a simple example, and the latter two are more complicated. To run a particular example, copy the file `user.f.whichever` to `user.f` before compiling.

11.1 Poisson Equation

The file `user.f.poisson` contains the user-supplied subprograms for the solution of Poisson's equation on a rectangle. All the default parameters are used. It illustrates the use of different types of boundary conditions on different sides. It is also a good example to become acquainted with controlling the graphical displays.

11.2 System of Equations

The file `user.f.system` contains an example of solving a system of equations. The system of two elliptic equations is solved by iteratively solving each equation, using the most recently computed solution of the other equation in the coefficients and right hand side. This illustrates the use of subroutines `save` and `ssolut` to save the state and evaluate the saved solution.

11.3 Time Dependent Problem

The file `user.f.timedep` contains an example of solving a (parabolic) time dependent problem. The Crank Nicolson method is used for discretizing the first order time derivative. Each iteration advances the solution one time step. This also illustrates the use of subroutines `save` and `ssolut` to save the state and evaluate the saved solution. Additionally, it shows how to call *gnuplot* directly to create graphics.

References

[1] W. F. Mitchell. Unified multilevel adaptive finite element methods for elliptic problems. Ph.D. thesis, Technical report UIUCDCS-R-88-1436, Department of Computer Science, University of Illinois, Urbana, IL, 1988. Available by anonymous ftp from casper.cs.yale.edu in mgnet/papers/Mitchell/thesis.ps

[2] W. F. Mitchell. Adaptive refinement for arbitrary finite element spaces with hierarchical bases. *J. Comp. Appl. Math.*, Vol. 36, pp. 65-78, 1991.

[3] W. F. Mitchell. Optimal multilevel iterative methods for adaptive grids. *SIAM J. Sci. Statist. Comput.*, Vol. 13, pp. 146-167, 1992.