# M4D Subroutines - a Programmers Guide

M4D is written so that there is a comment at the top of each .c file with the word ‘contains’ and the names of the subroutines in that file. A list of all the subroutines may then be generated by cd’ing to directory code they typing

```
grep contains *.c
```

The subroutines which start c_, pc_, or ec_ are also M4D commands and are listed and described in the file m4d.commands. Some of the remainder will be of general use to a programmer wanting to add to M4D. Non-command subroutines are described below by category.

## Adding Commands to M4D

To add a new command to M4D add the name and routine to be called to the list in commandlist.c. E.g.

```
else if (strcmp(command,"a_myadd")==0) a_myadd(fpin,fprint);
```

then add the appropriate information to the header file, commandlist.h. E.g.

```
void a_myadd(FILE *fpin, FILE *fprint);
```

and a line to the file makefile to include it in the compilation. E. g.

```
a_myadd.o \
```

if the corresponding file is a_myadd.c. For this example a_myadd will be called if the input command sequence contains

```
c: a_myadd
```

In this example I have prefixed both the command and subroutine with ‘a_’ to make it easy to distinquish from the commands and subroutines that come with M4D.

The header file global.h probably includes the headers you will need for your routine. There are also some useful definitions in macros.h, which I do not globally include, but instead, I copy the definitions into the routines which use them.

## Array Handling Routines

The array handling routines gives you access to the array handling system. These are: arraysize, arraytype, createarray, arraydelete, find, findarrray, need, smalloc, smalloca, and tmalloca located in array.c

int **arraysize**(const char *name)

Returns the size of the array name, or 0 if it does not exist.

char **arraytype**(const char *name)

Returns the character representing the array type, c (char), d (double), f (float), i (int), p or s (pointer). If the array is not found X is returned. (Note M4D currently contains no f - single precision floating point arrays.)

int **arraydelete**(const char *name)

Deletes array *name*. Returns 1 if successful, 0 if array not found.

void* **createarray**(const char *name, int size, char type, int nynew)

Create array *name*, with length *size* in unit of *type* = ‘i’ integer, ‘d’ double, ‘c’ character, ‘f’ float ‘p’ pointer,. If *nynew*=1 and the array already exists with a different length, the error causes an exit. Otherwise a previous array with the same name and a different length or type is deleted and replaced with the new one. Space for the array is allocated and createarray return the pointer to that space. *Name* is copied, so the space for the name may be freed by the calling routine.

void ***find**(const char *name)

Returns pointer to the values in array *name*; returns 0 if not found.

Array ***findarray**(const char *name)

Returns pointer to the Array structure (see arrays.h) for the array *name;* returns 0 if not found.

void ***need**(const char *name)

Returns the pointer to array *name*, An error message is printed and the program terminated is *name* is not found.

void ***smalloc**(int i)

Safe version of malloc which exits if allocation cannot be made. Returns pointer to space allocated.

void ***smalloca**(int i, char c)

Safe malloc for *i* items of type *c*, where *c*= c-character d-double f-float i-integer, or p-pointer or s-string. Returns pointer to space allocated. Exits if error.

void ***tmalloca**(int i, char c)

Like smalloca except that it keeps a list of these temporary allocations and frees them all when called with *i*<0; the last created is the first deleted. Note that tmalloca is automatically called with *i*<0 after the completion of each command.

**Input/Output Routines**

M4D input is read through the input routines. This allows for error checking and the parameter substitutions described in m4d.commands. These routines are in input.c which contains: findstring, findword, readdouble, readfilename, readfloat, readint, readline, readname, readnamep, read1charname, safefopen, setname, setname2, setname3. The print output routine, printout, which allows for levels of print is in m4d.c.

char **findstring**(FILE *fpin, const char *st)
char **findword**(FILE *fpin, const char *st)

Find string *st* in input file fpin. Return last character read or EOF. *findstring* looks for the string whether or not it is embedded in a word. *findword* requires it to be the next item or preceded by white space.

double **readdouble**(FILE *fpin)
float **readfloat**(FILE *fpin)
int **readint**(FILE *fpin)

A group of routines to read particular formats and return the value. *readdouble* and *readint* check to see if a variable parameter is there instead. These routines print a message and exit the program if an error or EOF is encountered.

char ***readline**(FILE *fpin)

*readline* reads one line or up to 80 characters. Trailing blanks are dropped. The string returned is created with *tmalloca* so that it will automatically be freed when the command terminates.

char ***readfilename**(FILE *fpin)
char ***readname**(FILE *fpin)
char ***readnamep**(FILE *fpin)

A group of routines to read strings. They can read names which include spaces if the whole thing is enclosed with apostrophies or quotation marks. *readfilename* allows both aliasing and parsing for an imbedded integer (see m4d.commands). *readname* returns a string created with *tmalloca* so that it will automatically be freed when the command terminates. *readfilename* and *readnamep* return permanent strings which can be freed using *free*. These routines print a message and exit if the string exceeds 200 characters,

char **read1charname**(FILE *fpin)

*read1charname* uses *readname* but returns just the first character.

FILE ***safefopen**(const char *name, const char *how)
Like *fopen*, except the program prints a message and exits if the file cannot be opened.

char * **setname**(char*from)
char * **setname2**(char*from,char*from2)
char * **setname3**(char*from,char*from2,char*from3)
Set permanent strings from 1, 2, or 3 input strings.

int **printout**(const char *type, const char *format, ...)        in m4d.c
An alternative to using fprintf(fprint,format,....) allows normal print on/off to be set by the command c_printcontrol and more print information when errors occur.

## Control Volume Geometry Routines

geomcsubs.c contains routines to calculate geometric properties associated with the continuity (between the points) control volumes. They are: geomcinit, geomcvave, geomcavector and geomcgradvol.

geom8subs.c contains routines to calculate geometric properties associated with the break-up of the continuity control volumes into the 8 parts associated with the corner points. They are: geom8init, geom8vol, geom8volfmid, geom8fx27, geom8areamid, geom8gradvol and geom8gradvxf.

The commands c_geomcprint and c_geom8print can be used to print the information these routines calculate (see m4d.commands).

void **geomcinit**(void)
Call before other geomc routines to locate needed arrays, *idim4d, xyz*.

void **geomcvave**(double *cg, int *ip)
( int ip[4]; double cg[8]; )
Calculates coefficients for contributions of corners to the volume average. Given the p-indices for the continuity control volume, *ip*, the volume average of a property is the sum of the 8 *cg* coefficients times the values at the 8 corner points i-1,j-1,k-1, i,j-1,k-1, i-1,j,k-1 .......i,j,k. (i.e. points ip[0]-1,ip[1]-1,ip[2]-1, etc. to ip[0],ip[1],ip[2])

void **geomcavector**(double (*area)[3], int *ic, int L)
( int ic[4],L; double area[4][3]; )
For a continuity control volume surface in the direction *L* (=0,1 or 2 for i,j,k), determine the net area vector associated with each of the 4 corner points. *ic* lower corner grid index. The 4 corners are at *ic*, then with *ic*[L+1] increased by 1, then ic[L+2] increased by 1, then both increased by 1 (with L rolling, 0,1,2,0,1,2).

double **geomcgradvol**(int *ip, double (*grad)[3])

( int ip[4]; double grad[8][3]; )

Determine the coefficients *grad* so that the gradient of a property, $\partial / \partial x_n$, over the continuity control volume is the sum of *grad*[corner][n] times the values at the 8 corner points i-1,j-1,k-1, i,j-1,k-1, i-1,j,k-1 …….i,j,k. *ip* is the p-index for the continuity control volume. The volume of the continuity control volume is returned. *grad* is formed as a 8 section average to get some non-parallelgram influence. For each section, grad*vol = di x dj d/dk + dj x dk d/di + dk x di d/dj. vol=dk dot (di x dj).

void **geom8init**(void)

Call before other geom8 routines so that needed arrays are available. Needed arrays are *idim4d*, *xyzt*, *cvdc, cvdc3* and for geom8areamid, *xyzdouble*.

void **geom8vol**(double *vol, int *ip)
( double vol[8]; int ip[4]; )

Calculate the 8 volumes of the breakup of the continuity control volume at p-index *ip*.

void **geom8volfmid**(double *vol, double (*fmid)[3], int *ip)
( double vol[8],fmid[8][3]; int ip[4]; )

Calculate the 8 volumes of the breakup of the continuity control volume at p-index *ip* and the interpolation parameters (fmid[8][3]) to located the center of each of the 8 volumes within the continuity control volume.

void **geom8fx27**(double (*f)[3], double (*xg)[3], int *ip)
( double f[27][3],xg[27][3]; int ip[4]; )

Calculate the 3x3x3 (=27) grid *xg*, for the breakup of the continuity control volume at p-index *ip*. Also the 3x3x3 interpolation parameters, *f*, which locate the points relative to the 8 corners of the continuity control volume.

void **geom8areamid**(double *area, double *xmid, int *id, int L)
( double area[3],xmid[3]; int id[4],L; )

Calculate the *L* direction spatial area vector, *area* using *xyzdouble*, also calculate *xmid*, the x,y,z, midpoint (4 pt average) of the area segment. *id* is the index of the lower corner of the area in array *xyzdouble*. For out of range *id* values, *area* and *xmid* are zero vectors.

void **geom8gradvol**(double (*gradv)[8][3], int *ip)
( double gradv[8][8][3]; int ip[4]; )

For the *n*th of the 8 subvolumes in the continuity control volume at *ip*,

$$\int \frac{\partial \phi}{\partial x_i} dV = \oint \phi \, dA_i = \sum_{k,8 \text{ corner pts}} gradv[n][k][i]\phi_k$$

void **geom8gradvxf**(double (*gradv)[8][3], double (*x)[3], double (*f)[3])

( double gradv[8][8][3],f[27][3],x[27][3]; )

Given the 3x3x3 interpolation parameters, *f*, and the 3x3x3 grid, *x*, for a continuity control volume, calculate *gradv* for the 8 subvolumes as described for geom8gradvol.

<u>**Interpolation  and Grid Index Routines**</u>

findex, find1dinterp, find2dinterp, interp3d, findex3d in interp.c, x3line in c_wallnorm.c and iexpand in iexpand.c

int **findex**(double a, double *a1d, int id, double *f)

Find the interpolation info *j*.*f*[0] so that *a*=*a1d*[j] +*f*[0]* (*a1d*[j+1]–*a1d*[j]). Note *a1d* must be monotonic increasing. Range limited to 0, *id*–1. Returns *j*. *f*[0].

void **find1dinterp**(double *x, double (*xp)[3], double *ff, int nylimit)
(  double x[3],xp[2][3],ff[1]; int nylimit; )

Determine by a normal projection the fractional distance, *ff*[0], best locating *x* along the two-point line *xp*. If *nylimit*>0, limit *ff*[0], 0<=*ff*[0]<=1.

int **find2dinterp**(double *x, double (*xp)[2][3], double *ff, int nylimit, int itermax)
( double x[3],xp[2][2][3],ff[2]; int nylimit,itermax; )

Find interpolation parameters *ff* to best locate point *x* in the quadrilateral *xp*. An iterative procedure is used starting with *ff* and using up to *itermax* iterations. If *nylimit* > 0 ff[0] and ff[1] are limited to the range 0, 1. Returns the number of iteations used.

int **findex3d**(double *xx, double **x, int *id, int *ilo, int *ihi,
double tol, double *fout)
double xx[3],*x[3], fout[3],tol;  int id[3],ilo[3],ihi[3];

Find 3d interpolation indices, *fout*, locating *xx* in arrays *x*[0], *x*[1], *x*[2], representing x,y,z respectively, and  dimensioned *id*. Limit the index range for the result from *ilo* to *ihi*.  An iterative procedure is used to determine each component of *fout* to within *tol*. The routine returns the number of iterations used if successful, and minus the number of iterations if unsuccessful. *fout* is a non- integer index, so that, for example, *fout*[0] = 2.61 means that in the i[0] direction the point *xx* is 61% of the way from i[0]=2 to i[0]=3. The companion routine, interp3d, may be used to interpolate property values from the *x*[] grid to the point *xx*.

void **interp3d**(double *x, int *idim, double *f, double *xout)
( double *x,f[3],*xout; int idim[3];  )

Interpolate in the array, *x*, for the value, *xout*[0], at the non-integer index, *f*. Note that, *idim,* the dimensions of *x*, must be >=2 in all three directions.

void **x3line**(int m1, int m2, int m3, int iall, double *xyz, double *xx)
( int m1,m2,m3,iall; double *xyz, xx[3]; )

Given the indices of 3 grid points, *m*1, *m*2, *m3*, interpolate to determine a vector, *xx*, which is parallel to the line of points at point *m2*. *iall* is the total number of grid points and *xyz* their coordinates.

void **iexpand**(int k, int *idim, int *i)
( int k,idim[4],i[4]; )
Expand index *k* into its 4 components *i*. based on dimensions *idim*. ( The companion macro, **In4**(i,idim), in macros.h returns *k* from *i*. )

## Control Volume Determination Routines

cvdclimit1, cvdcactive, cvdccorner, cvdclines, cvdcface, cvdcfill, cvdcsleep in cvdcsubs.c, and cmatchfollow in cmatchfollow.c. All are associated with command c_cvdcreset.

**cvdcactive** - set *cvdc* values for each continuity control volume. See c_cvdcreset in m4d.commands for the details.
**cvdclimit1** - limit *cvdc* based on the overall limitation and point types. See command c_cvdcinit.
**cvdcsleep** - fills in *cvdc* values for sleeping contiuity control volumes.
**cvdcface** - fills in values of *cvdcdouble* for continuity control volume faces.
**cvdclines** - fills in values of *cvdcdouble* for the edges of continuity control volumes.
**cvdccorner** - check and correct *cvdc* and *cvdcdouble* at grid corners (collapsed volumes of ‘c-grid’ corners).
**cvdcfill** - check and complete *cvdc* and *cvdcdouble* for end values and repeating boundaries.
**cmatchfollow** - follow through sleeping volumes to match both sides of a continuity control volume face.

## Coefficient Array Service Routines

coefcadd, coefcombine, coefcenterfirst, coeforder, coeforderv in coefsubs.c and coefprint1 in c_set_cpflop.

void **coefcadd**(int iw, int *cn, int **ci, double **cc, int nocoefs, int jcoef, int n, int *ipt, double *cadd)
Add to the coefficient arrays for equation *iw*, at the *n* points given in *ipt*, the coefficient additions *cadd*. The coefficient arrays are specified by *cn, ci, cc*, with a total of *nocoefs* sets of coefficients. The addition is made to the *jcoef* set of coefficients. Reset *cn, ci, cc* so that effectively,
for ci[iw][j]=ipt[k], add cadd[k] to cc[iw][j+cn[iw]*jcoef], k=0,,n-1

void **coefcenterfirst**(int icen, int iw, int *cn, int **ci, double **cc, int jrep)
If there are any coefficients for the *iw*’th equation, the coefficient set is rearranged so that point *icen* is made the first point in list *ci*, with *icen* being added to the list if it was not already there. *jrep* is the number of coefficient sets in *cc*.

void **coefcombine**(int iw, int *cn, int **ci, double **cc, int irep, double tol)

For independent point iw, determine roundoff levels (abs_max*tol), combine coefficients with the same ci[iw][i] indices, and irep (no of coefs per ci index), then clean off roundoff, omit zero points and consolidate the coefficient arrays. jrep is the number of coefficient set in cc.

void **coeforder**(int icen, int iw, int *cn, int **ci, double **cc, int jrep)

Calls coefcenterfirst, then rearrange the points so that the remaining points are in increasing index order.

void **coeforderv**(int icen, int iw, int *cn, int **ci, double **cc, int jrep, int jorder)

Calls coefcenterfirst, then rearrange the remaining points so that the first set of coefficients are in decreasing order.

void **coefprint1**(int i, int *cn, int **ci, double **cc, int jrep, FILE *fprint)

Print the coefficient array for equation i. Used by c_set_cpflop and blockcoef.

## Equation Service and Solution Routines

eqnrhs, eqnrhsbij, eqncplus, eqnerror, eqnerrorwho, eqncenter, eqncenterbij in eqnsubs.c. TDMA routines tdmaalloc, tdmafree, tdmafromcoef, tdmasolve, tdmaijkcoef, tdmar in tdmasubs.c.

**eqnrhs** - calculate the current equation residuals considering the current change in the variable.
**eqnrhsbij** - calculate the current equation residuals for the bij equations.
**eqncplus** - set *cplus* as the sum of the positive coefficients in the equation.
**eqnerror** - update the error tracking array. For each active equation the error is calculated as rhs[j]/cplus[j].
**eqnerrorwho** - update the error tracking array noting the point with the worst error.
**eqncenter** - do a center point update of the change in the property using the current equation residuals and a center point coefficient array.
**eqncenterbij** - do a center point update for the 6 components of *dbij* using the current equation residuals, a shared center point coefficient array for relaxation, and the 6x6 coefficient matrix linking the *dbij* components.

**tdmaalloc** - allocate space for a Tdma (tridiagonal matrix algorithm) structure.
**tdmafree** - free the linked list of Tdma structures.
**tdmafromcoef** - organize a set of coefficients into a list of Tdma structures based on the interdependency of the equations as given by the coefficients.
**tdmasolve** - solve the equations as a list of Tdma structures.
**tdmaijkcoef** - organize i, j, or k lines of equations into a list of Tdma structures.
**tdmar** - the tri-diagonal matrix algorithm for repeating boundaries.

## Additional Pressure Equation Routines

omitpsleep in c_set_cpflop.c, blockcoef, blockrhs, blocktdma, blocktop for block pressure solve in blocksubs.c, eqnbndry in eqnsubs.c, psolve1d in psolve1d.c, psleepcalc in psleepcalc.c.

int **omitpsleep**(int iw, int *cn, int **ci, double **cc, int jrep, int *matchpc, double *cpsleep, int iallp)

Alter coeffcient array at *iw* to resolve sleeping p-points in terms of active pressure points. Returns a value greater than 0 if sleeping points were found.

The blocks for the multi-block pressure solution option are set up using command c_blocksetabc. These are used if requested by the input for command c_eqnsolvep.
**blockcoef** - set up the coefficients for the specified block pressure groups.
**blockrhs** - evaluate the current residual for the block pressure groups.
**blocktdma** - solve, using 2 i,j,k TDMA passes, the block pressure group equations.
**blocktop** - put the block pressure changes back to the full pressure grid.

**eqnbndry** - update the change in pressure at the exit boundary.
**psolve1d** - 1-d solver for pressure change update, i, j, or k direction.
**psleepcalc** - update p-grid property at sleeping points from non-sleeping points.

## Companion Routines for Commands

Several commands use command routine to process input and output and have a companion routine of the same name, without the c_ to do the work. See m4d.commands for a description. The companion routines are:

**areaflowint** - Used by c_areaflowint and pc_iplaneint.
**arraydump** - Used by c_arraydump and c_arraydumpmore.
**arrayhowto** - Used by need in arrays.c.
**bijrhsmarv** - Used by c_bijrhsmarv and c_bijrhsmarvs.
**coefconv** - Used by c_coefconv and c_coefconvstep.
**coefrhs** - Used c_coefrhs, c_momrhsr, c_omrhscoakley, c_omrhsmarv, c_qturbrhs.
**coefvisc** - Used by c_coefvisc and c_coefviscstep.
**interp_ptod** - Used by c_interp_ptod.
**interp_ptog** - Used by c_interp_ptog
**omwall** - Used by c_omwallcoakley, c_omwallmarv, c_omwallmarvs, c_omwall.
**ppreset** - Used by p_ppreset.
**valueatp** - Used by c_valueat.

## Principal Stress and Correction Routines

prinstress, shearijzero in prinstress.c, fixbij fixdbij in fixbij.c, fixdudx fixdudx.c.

double **prinstress**(double (*re)[3], double (*retr)[3], double (*tr)[3], double tol)
 (double re[3][3],retr[3][3],tr[3][3],tol; )

Given a symmetric tensor, *re*, determine the principle stress tensor, *retr*, and the coordinate transformation that gives it, *tr*. The procedure used is iterative, with up to 150 iterations or until the sum of the absolute values of the shear stresses is less than *tol*. The routine returns the sum of the absolute values of the shear stresses.

void **shearijzero**(double (*re)[3], double (*tr)[3], int i, int j)
( double re[3][3], tr[3][3]; int i,j; )

Determine the transformation tensor, *tr*, that gives the shear component, *re*[i][j] equal to zero. Note that *re* is not changed.

double **fixbij**(double *bij, double gtol)
( double bij[6], gtol; )

Scale down *bij*, if needed, so that the Reynolds stress tensor is realizable. Modifies *bij* (if needed) and returns the scaling factor.

double **fixdbij(**double *bij, double *dbij, double gtol)
( double bij[6], dbij[6], gtol; )

Scale down d*bij*, if needed, so that the Reynolds stresses from *bij* + *dbij* are realizable. Modifies *dbij* (if needed) and returns the scaling factor.

double fixdudx(double (*du)[3])
 ( double du[3][3]; )

Fix $du = \partial U_i / \partial x_j$ to satisfy incompressible continuity, taking into consideration zero components which indicate 2-d instead of 3-d flow. Returns the initial error, $\left(\partial U_1/\partial x_1 + \partial U_2/\partial x_2 + \partial U_3/\partial x_3\right) / \left(\left|\partial U_1/\partial x_1\right| + \left|\partial U_2/\partial x_2\right| + \left|\partial U_3/\partial x_3\right|\right)$.

## Other Routines

main, m4dtitle, exitm4d in m4d.c, commandlist in commandlist.c, whoelseadd in c_gridmatch.c, readmefpgrid in readmefpgrid.c,

**main** - main program.
**m4dtitle -** routine to print the program title.
**exitm4d** - all program exits come through this routine.
**commandlist** - list of commands and the routines they call.
**whoelseadd** - used by c_gridmatch to add points to *whoelse*.

void **readmefpgrid**(char *filename, double **axyz, int **alt, double **aabc,
 int *idim)

Allocate temporary space for (using tmalloca), and then read a 3-d MEFP style grid from file *filename*. The addresses for the XYZ, and LT arrays are returned in *axyz* and *alt*. A, B and C are put into a single array (all run together) with address *aabc*. The dimensions are returned in *idim*.

## Plot Package Routines

p_fillcontour, p_glines, p_vectors, p_nextpt2d in .c files of the same name, p_outgif p_newline p_gifheader p_lzw p_set2char p_writebits in p_outgif.c, p_outjgm p_setpixel, p_stline p_stlinex p_stlimited p_fillx2d p_filltri2d p_painttri p_xlin in p_plotsubs.c, p_openfile p_readfont p_symbol p_symbolx in p_symbol.c.

**p_fillcontour** - plot color-fill contours, used by pc_picture.
**p_glines** - plot grid lines, used by pc_picture.
**p_vectors** - plot velocity vectors, used by pc_picture.
**p_nextpt2d** - find next grid line crossing in specified direction. Used by pc_xytocgrid.
**p_outgif** - convert an image to a .gif file.
**p_newline** - find next newline character in an input file.
**p_gifheader, p_lzw, p_set2char p_writebits** - used in writing .gif files.
**p_outjgm** - convert an image to a jgm format file.
**p_setpixel** - set specified pixel to specified color in image.
**p_stline, p_stlinex, p_stlimited** - draw a straight line of specified pixel width and color in an image. p_stline specifies the start and end as pixel integers, p_stlinex specifies the end points as doubles, and p_stlimited uses p_stlinex after limiting the line to a specified box in the image.
**p_fillx2d** - fill in an image, with the specified color, a triangle with y1=y2,.
**p_filltri2d** - break a triangle up into sub-triangles, two points with the same y.
**p_painttri** - break a paint triangle up into individual color segments.
**p_xlin** - linearly interpolate for x, given y and 2 points on the line.
**p_openfile** - open the specified file, searching the current directory and those listed in *p.defaultdir*.
**p_readfont** - read the file font.list and the listed fonts.
**p_symbol** - plot string in image at specified pixel location given as integers.
**p_symbolx** - plot string in image at specified pixel location given as doubles.