M4D Commands

Command Format

Commands are read initially from standard input, or by redirection (<) from an input file (*which must be plain text, NOT a .rtf file*).

Example: c: commandname input needed if any then comments if wanted

The "c:" is an indicator that a command has been found and the next word is taken as the command name. After any needed input is read, items in the input file are ignored (so you may add comments) until the next occurrence of "c:".

Corresponding subroutines are:

```
void c_commandname(fpin,fprint)
```

in file c_commandname.c (unless otherwise specified). fpin is the file pointer for input, and fprint the file pointer for printed output. (Most output now goes through the routine 'printout' instead of using the C 'fprintf(fprint,'…). fpin and fprint are initially standard input and standard output, so that the program may be run, e.g., with

```
a.m4d < in.start > printfile
```

An imbedded plotting package is included. The subroutines corresponding to these commands are in files pc_commandname.c.

Command Input

The names of each variable read by a command are color-coded here to describe the variable type: double, integer, 1-character name, file name, name or string, or multiple types.

Variables of type double or integer may be paramertized. I.e., a numerical value may be read, or the name of an array of the same type and dimension 1 may be substituted. E.g.,

```
c: constant ITERMAX i 1 10
c: infile inn.iter ITERMAX
```

sets ITERMAX=10, then repeats the command sequence in file inn.iter ITERMAX times.

File names may be aliased by setting a string variable. E.g.,

```
c: constant inn.iter s 1 inn.iter.dns
c: infile inn.iter 10
```

This sequence will repeat the commands in inn.iter.dns 10 times.

File names may also have an imbedded integer variable. E.g., with ITER=5,

```
c: arraydump vv#ITER U1 ""
```

dumps U1 to file vv5. The form vv#ITER#more would go to vv5more. The code checks for the alias before parsing the name for an imbedded integer.

Note that 1-character names are actually read as strings, with all but the first character ignored. E.g., if the input is 'end' the character 'e' is the 1-character name.

Programmers note: The options described above are implemented by reading input through the subroutines in input.c.

Alphabetical List of M4D Commands
The number following is month.year for the source version which is documented.
Commands prefixed by ec_ are experimental and/or lack generality.
Commands prefixed by pc_ are associated with the imbedded plotting package.
Superscript ^{S} indicates the command has no input.
Prefix = indicates the documentation may need to be updated.

pc_abcmask 2.15
c_algebra 2.15
c_alias 2.15
c_areaflowint 2.15
c_arraydelete 2.15
c_arraydump 2.15
c_arraydumpmore 2.15
c_arrayhowto 2.15
c_arraylist^S 2.15
c_arrayread 2.15
c_aveijk 2.15
pc_bar 4.15
c_bijrhsmarv^S 2.15
c_bijrhsmarvs^S 2.15
ec_bijrhsmarvex 2.15
c_bijwallmarv^S 1.13
c_bijwallsplat^S 2.15
c_blocksetabc 2.15
c_coefadd 2.15
c_coefconv 4.15
c_coefconvstep 2.15
c_coefcplus 2.15
c_coefdt 2.15
c_coeffix 2.15
c_coefinit 2.15
c_coefrhs 2.15
c_coefvisc 2.15
c_coefviscstep 2.15
c_coefzero 2.15
c_comment 12.15
c_constant 4.15
c_contcpcdu^S 11.13
c_contcpcexit^S 10.14
c_contcpcfixed 4.15
c_contcpcinlet^S 1.13
c_contdu^S 1.13
c_contrhsexit^S 10.14
c_contrhsp^S 1.13
c_contrhsu^S 12.13
c_copy 2.15
c_copy_mtop 2.15
c_copy_ptom 2.15
c_copypart 5.15
c_cvdcinit 12.13
c_cvdcparm 12.13
c_cvdcreset 12.13
c_ddtall^S 12.14
c_edit 2.15
c_editabcd 2.15
c_end^S 5.15
c_eqnsolvebij 2.15
c_eqnsolvep 2.15
c_eqnsolves 2.15
c_eqnupdatem 2.15
c_eqppts2ppts 2.15
c_eqpts2gpts 2.15
c_exitinit 4.15
c_function 2.15
c_geom8print 2.15
c_geomcprint 2.15
pc_giflist 2.15
c_gpts2eqpts 2.15
c_gradprop 2.15
c_gridcorner 2.15
c_gridfrommefp 2.15
pc_gridinfo^S 2.15
c_gridmatch^S 2.15
c_gridtomefp 2.15
c_gridvarmod 2.15
c_if 2.15
pc_image 2.15
c_infile 5.15
c_inletinit 2.15
c_inletreset^S 2.15
c_interp_gtom 2.15
c_interp_gtop 2.15
c_interp_ptod 2.15
c_interp_ptog 4.15
pc_iplaneint 2.15
pc_keyword 2.15
c_lineoutput 2.15
c_lineoutputijk 2.15
pc_lineplot 2.15
c_mirror 2.15
c_momcam 2.15
c_momcamddt^S 10.14
c_momrhsr 2.15
c_omrhscoakley^S 2.15
c_omrhsmarv^S 2.15
c_omwall 8.13
c_omwallcoakley^S 4.15
c_omwallmarv^S 4.15
c_omwallmarvs^S 4.15
pc_picture 2.15
c_ppreset^S 1.13
c_ppts2eqppts 2.15
c_prinstress^S 2.15
c_print 4.15
c_printcontrol 5.15
c_prints 2.15
c_printscoef 2.15
ec_qbijles 12.14
c_qturbrhs 2.15
ec_rayleigh 2.15
c_rmsminmax 2.15
c_set_contar^S 2.15
c_set_cpda 2.15
c_set_cpflop^S 2.15
c_set_cpsleep^S 2.15
c_set_gbij^S 1.13
c_set_pkdk 2.15

Description of M4d Commands
(plot package commands, starting with pc_ described separately below)

algebra Perform algebra on arrays of type double.
Input: *namea, nameb, namec, named, e, f, g*
where *namex* are array names and *e, f* and *g* are constants.

Set *namea*[j]=*nameb*[j]*(*e***namec*[j]+*f*)^*g* + *named*[j],

where "^*g*" indicates to the power *g*. Note that ^*g* is omitted if *e***namec*[j]+*f* is zero or negative. If *namea* does not exist it is created with the minimum length of the other arrays. (All the arrays should have the same length for the routine to make sense, but it is not required.) Arrays *nameb*, *namec* or *named* may be omitted with a blank name (""). If omitted the defaults are *nameb*[j]=1, *namec*[j]=1, *named*[j]=0. If *nameb*, *namec* or *named* is not a blank name and the corresponding array does not exist, the program prints as error message and stops.

Example: c: algebra sum "" add sum .5 0. 1. sets sum=sum+.5*add.

alias Set alias names for parts of an array.
Input: *name, iparts; namepart*[j], for j=0,, j<*iparts*
Create alias names for *iparts* equal size sections of array *name*. These alias names may be used as if they were independent arrays, e.g. in commands c_algebra or c_print, etc. The exceptions are: c_arraydelete, only independent arrays may be deleted and any routine that tries to recreate the array. Note that alias names will not be changed once they are set, unless the array *name* is first deleted, and then recreated. Attempts to reset the alias will be ignored.

Example: c: alias bij 6 b11 b22 b33 b12 b13 b23

areaflowint Area and flow integrals over specified portions of grid planes
Input: *abcijk, value, value1s , value1e, value2s , value2e*
Nameprop, repeat reading *nameprop* until *nameprop*=""
Determine the area, + flow and – flow across part of a plane of constant grid index, i, j or k. *abcijk,* the single character a, b, c, i, j, or k, specifying the plane type. *value*, the plane location specified as a floating point value if a, b, or c is specified, or an integer if i, j, or k is specified. *value1s, value1e* gives the range of the next space grid index (examples: b range if 'a' was specified, i range if 'k' was specified); *value2s,*

value2e the range of the third space grid index. Note when a, b, or c values are specified, the nearest corresponding grid index is used. Similarly all ranges are bounded by the actual grid range. The time index for the averages is the last one (*idim4d*[3]) or , if it exists, *itrange*[1]. Also calculated are the area average and mass averages (+, –, overall) of each on-the-grid-points variables in the nameprop list. Up to 20 properties may be specified. Results are stored in the array *averages* as well as printed with the line indicator AFINT (so they can be culled from the printout with Unix command grep). If you wish to keep the results from more than one plane, copy *averages* to a different name using command 'copy' before the next call to areaflowint.

Example: c: areaflowint a 100. –10. 10. 0. 1. U1 qturb ""

Needs: *idim4d, abcd, xyz, rho, U1, U2, U3.* Uses if available: *itrange, ITER, TIME.* Sets: *averages.*

arraydelete Delete arrays.
Input: *n; name*[j], for j=0,, j<*n*
where *n* is the number of arrays to be deleted, and *name*[j] are the names of the arrays to be deleted. Note, if an array does not exist, or is an alias name, no action is taken.

Example: c: arraydelete 3 ptt sdum rtemp

arraydump Dump arrays to a named file.
Input: *filename, name*, repeat reading *name* until *name*=""

Example: c: arraydump saveme U1 U2 U3 pp ""

Saves U1 U3 U3 and pp on the file saveme (or its alias). Arrays of type double, integer and character may be dumped. Arrays which are not of these types or which do not exist are ignored. For each array the dump contains a one line header with the name, size and type (d, i or c). If the array size is 1, the value is on the same line. Otherwide this is followed by the values in the order stored starting on a new line. The arrays may be read in again using arrayread. Note a formatted dump is used so that the files may also be read by humans.

arraydumpmore Append dump of arrays to a named file.
Input: *filename, name*, repeat reading *name* until *name*=""

Example: c: arraydumpmore saveme dU1 dU2 dU3 ""

Same as array dump except that the arrays are added at the end of the file saveme. Located in c_arraydump.c

arrayhowto Read file name with instructions on how to make arrays.
Input: *namehowtofile*

The file *namehowtofile* contains a list of variables and the commands that create them. When a subroutine needs an array, and it doesn't exist, if arrayhowto has been called, it looks in the file to see if it can make the array. The format of the file is the indicator v: followed by the array name followed by the command and its input. It is suggested that the list contain only instructions for arrays which need to be set once.

Example: c: arrayhowto inn.arrayhowto

File inn.arrayhowto:

```
v: walldist walldist
v: walln2m walldist
v: wherefw set_wherefw –1 s
v: noindfwpts set_wherefw –1 s
v: whoisfw set_wherefw –1 s
```

If, for example, the command c: visccoakley comes before *walldist* exists, the file inn.arrayhowto tells the program *walldist* is made with c: walldist and it does it.

arraylist List information for all arrays: name, size, type, location.

Input: none

Located in arrays.c

Example: c: arraylist

arrayread Read in arrays written using arraydump.

Input: *nyneed, filename*

If *nyneed* > 0, the program stops if file *filename* does not exist. The program also stops if the file cannot be correctly read.

Example: c: arrayread 1 saveme reads the arrays in file saveme.

aveijk Take i, j, and/or k averages.

Input: *namefrom, nameto, symi, symj, symk*

Repeat until *namefrom* = ""

From on, or between the points array *namefrom,* take the specified average and put the results in *nameto*. The size of *nameto* is set as the same size as *namefrom*. *symi, symj, symk* are 1 character names specifying the type of averaging in the i, j, and k directions respectively. Note: no check is made of the actual geometry to see if the averages taken are meaningful.

If *sym*='s', the planes are point-wise averaged in the specified (i, j,or k) direction. Note that all planes are filled with the average. Appropriate for uniform spacing.

If *sym*='a', a spacing weighted average in the direction is taken using the array *abcd* to evaluate the spacing.

If *sym*='m', the point is averaged with its mirror image. Appropriate for grids where the grid spacing mirrors, e.g, b=-100, -99, -95, ……, 5, 1, 0.

If *sym*='M', the point is averaged with the negative of its mirror image.

Note if 2 directions have ‘m’ or ‘M’ specified, then 4 points are averaged with the specified +, – symmetry.
If *sym*='I', a diagonal mirror is used. E.g. if *symj*='I', the j and k indices are exchanged. This requires the j and k dimensions to be the same. Note no averaging is done in the specified direction.
If *sym* is any other character no averaging is done in that direction

Example: c: aveijk U1 U1j s n m U2 U2j s n m U3 U3j s n M ""
Take averages in the i-grid direction, and also average with the mirror image in the k-direction. Appropriate for a channel with x, y, z in the i, j, k grid directions, respectively.

c: aveijk U1 U1r n I n U2 U3r n I n U3 U2r n I n ""
Mirror the velocity across the diagonal of a square channel.

Needs: *idim4d, abcd*. Uses if available each *namefrom*. (Routine stops if there are more than 20 non-available arrays.) Sets each corresponding *nameto*.

bijrhsmarv Evaluate the right hand side of the b_{ij} equations for the MARV model.
Input: none
Set the right hand side of the b_{ij} equations using coefficient set 0 (for convection, etc.) and the source term based on the MARV model.
The source terms for each of the b_{ij} equations for each equation point k may be written as

$$\rho S_{b_n}(k) = beqm[k + n * noindfwpts[0]] + \sum_{j=0}^{j<6} dbij[wherefw[k] + j * nogpts[0]] * beqd[k + (n + 6 * j) * noindfwpts[0]] .$$

where *beqm* is an explicit contribution and *beqd* are implicit center point coefficients for the change in b_{ij} (*dbij*) to be calculated over the time step or iteration. The six equations, n=0,,,,5, correspond to the Cartesian components, $b_{11}, b_{22}, b_{33}, b_{12}, b_{13}, b_{23}$.
This command sets *beqd* in the above equation, the right hand side of the b_{ij} equations, *rhsbij*,

$$rhsbij[k + n * noindfpwts[0]] = beqm[k + n * noindfwpts[0]] - \sum_{j=0}^{j<coef\_n[k]} coef\_c[k][j + coef\_n[k] * jcoef] * namev[coef\_i[k][j] + n * nogpts[0]]$$

and *cambij*[k], a centerpoint relaxation coefficient based on *beqd*.
Also set between-the-points arrays,
gbij, the anisotropy parameter, g,

pkdk, the turbulence production rate, *P*, divided by the turbulence kinetic energy.
If bug=1 (see source code), the array, *beqex*, with several between-the-points parameters used in the pressure-strain model will also be set (pde~, Rt~, geff, fw, fb, c1, c3, c4, c6, fr, capg, c1homo or see print output for current list).

Example: c: bijrhsmarv

Needs: *idim4d, wherep, noindfwpts, wherefw, whoisfw, U1, U2, U3, rho, qturb, omturb, vlam, bij* and through geometry routine calls, *xyz, cvdc, cvdcdouble.* Uses if available: *itrange, zrotation.* Needs *walldist* if *zrotation* exists.
Sets: *rhsbij, cambij, beqd* (for c_eqnsolvebij), *gbij* (for c_omrhsmarv), *pkdk* (for c_omrhsmarv and c_qturbrhs). Also sets *beqex* if bug=1 (see source code).

bijrhsmarvex Right hand side of the b_{ij} eqs. for MARV family of models.

Input: *namemodel, namec5, namec7, namepkdktot*
The MARV family of models has several dependencies which may be omitted or included. (see 2006 book). The basic model. MA, is for fully-developed shear flow. If *namemodel* includes 'r' or 'R', rapid distortion effects are included. If *namemodel* includes 'v' or 'V', the differences between irrotational flow shear flow are included. If *namemodel* includes 's' or 'S' the changed constant appropriate for the splat wall boundary condition is included.
The remaining input, *namec5, namec7, namepkdktot,* allows modifications to the source term for the b_{ij} equations. This is being considered for instability modeling. They are respectively the names for the between-the-points arrays for c_5, c_7 and P/k in the below equation. The modification is omitted if the corresponding *name*="". If omitted the defaults are $c_5 = 0$, $c_7 = 0$, and P/k is calculated by the subroutine.

$$\begin{aligned} S_{bij} &= -\frac{1}{2}b_{ij}\left((c_1-2)\omega + (2+\frac{1}{2}c_6)\frac{P}{k}\right) + \frac{1}{2}\left(c_3 - \frac{4}{3}\right)S_{ij} \\ &+\frac{1}{2}(c_4-2)(b_{ik}S_{jk} + b_{jk}S_{ik} - \frac{2}{3}\delta_{ij}b_{mn}S_{mn}) \\ &-(b_{ik}\omega_{jk} + b_{jk}\omega_{ik}) - 2(1-f_R)\Omega_k\left(\varepsilon_{ikm}b_{jm} + \varepsilon_{jkm}b_{im}\right) \\ &+\frac{1}{2}c_5\left((b_{ik}\omega_{jk} + b_{jk}\omega_{ik}) + (1-f_R)\Omega_k\left(\varepsilon_{ikm}b_{jm} + \varepsilon_{jkm}b_{im}\right)\right) \\ &+\frac{1}{2}c_7(b_{ik}b_{kn}S_{jn} + b_{jk}b_{kn}S_{in} - 2b_{kj}b_{ni}S_{kn}) \end{aligned}$$

Example: bijrhsmarvex marvs c5 "" pkdktot

This example uses the MARVS model with the addition of a c_5 coefficient and an alternate evaluation of P/k. Do NOT use the array name pkdk for the alternate evaluation.
The arrays needed and set are the same as for bijrhsmarv.

bijrhsmarvs Evaluate the right hand side of the b_{ij} eqs. for the MARVS model.

Input: none

A variation of the MARV model with one constant changed so it is appropriate for use with the splat wall boundary condition. All else the same.
Located in c_bijrshmarv.c

Example: c: bijrhsmarv

bijwallmarv Set b_{ij} on the walls for the MARV model.

Input: none

Example: c: bijwallmarv

Needs: *idim4d, noindwpts, whoisw, whoelse, xyz, U1, U2, U3, omturb, wnear, wnorm, bij*. Uses if available: *itrange*. Modifies: *bij*.

bijwallsplat Set b_{ij} on the walls using "splat" wall reflection.

Input: none

Example: c: bijwallsplat

Needs: *idim4d, noindwpts, whoisw, whoelse, wnear, wnorm, bij*. Uses if available: *itrange*. Modifies: *bij*.

blocksetabc Set up groups for a block solve of pressure-correction equations.

Input: *label*, *ib*, *jb*, *kb*

If *ib*<*idim4d*[0] read *a*[i], for i=0,,,i<*ib*
If *jb*<*idim4d*[1] read *b*[j], for j=0,,,j<*jb*
If *kb*<*idim4d*[2] read *c*[k], for k=0,,,k<*kb*

label sets the 1-character name for this block solve. This label may then be used in the string sequence read by eqnsolvep to determine how to solve the pressure-correction equations. Do NOT use b, c, e, i, j, k, t, I, J or K as these have other meanings.
ib, jb, kb are the number of blocks in the i, j, k grid directions. If *ib, jb,* or *kb* equals the grid dimension in that direction, single lines are used. Otherwise *a*, *b*, or *c* values (corresponding to values in array *abcd)* are read spanning the groups of pressure-correction equations to be treated as a block.

The block solve solution procedure takes the change in pressure (in each solve iteration) to be the same for all points in the block. The blocks are then strung together in the i, j, and then k directions and solved using the tdma algorithm. If multiple levels of block solve are used it is suggested that the coarser ones be made up of groups of finer ones. Eqnsolvep is currently dimensioned to use a maximum of 5 levels of blocks.

Example: c: blocksetabc q 13 6 6
0 .6 1.2 2.1 3. 4. 4.8 5.7 6.7 7.6 8.5 9.4 10.

```
-1 -.6  -.2  .2  .6  1.
-1 -.6  -.2  .2  .6  1.
```

Needs: *idim4d, abcd*. Sets: *blockijkq* (e.g. if *label*='q'). Located in blocksubs.c

coefadd Form a sum of coefficient sets in the *coef_c* coefficient array.

Input: *jto, nfrom;* *jfrom*[j], *fac*[j] for j=0,,j<*nfrom*

Set the *jto* set of coefficients in array *coef_c* to the sum of *fac*[j] times the *jfrom*[j] set of coefficients,

$$coef\_c[k][i+jto*n]=\sum_{j=0}^{j<nfrom} fac[j]*coef\_c[k][i+jfrom[j]*n]$$

for each equation, *k,* and for each point in the equation, *i*=0,,,*i*<*coef_n*[*k*].

Example: c: coefadd 0 2 1 1. 2 1.5   Set 0 = set 1 + 1.5 * set 2.

Needs: *coef_c, coef_n, noindfwpts*. Uses if available *itrange*. Modifies: *coef_c*.

coefconv Set convection coefficients using linear discretization.

Input: *jconv*

Set the *jconv* set of coefficients in array *coef_c* to correspond to convection. coefconv uses convective discretization over a control volume, so that for a convected property *C,* the convection term for the point *m* is

$$\sum_{i=0}^{i<coef\_n[k]} coef\_c[k][i+jconv*coef\_n[k]]*C[coef\_i[k][i]]$$

$$=\sum_{sub-volumes}\overline{\rho U}_j\int\frac{\partial C}{\partial x_j}dVol=\sum_{sub-volumes}\overline{\rho U}_j\oint CdA_j$$

where *k* (=*wherefw*[m]) is the equation corresponding to point *m,* and *coef_n*[*k*] is the number of points in the discretized equations. $\overline{\rho U}_j$ is the volume average value of ρU_j for each continuity control volume. Each continuity control volume is broken up into 8 sub-volumes (consistent with *xyzdouble*) which are assigned to the corner points. The equations are formed for linear variations of the convected property *C* between the grid points.

Example: c: coefconv 1   Coefficient set 1 corresponds to convection.

Needs: *idim4d, wherep, noindfwpts, wherefw, match, whoisfw, nocoefs, U1, U2, U3, rho, xyz, cvdc, cvdcdouble*. Uses if available: *itrange, roundoff*. Modifies: *coef_n, coef_i, coef_c*.

coefconvstep Set convection coefficients using stepwise discretization.

Input: *jconv*

Like coefconv, except that coefficients are folded towards a 7 point coefficient matrix. While the coefficient set will be more stable, it will introduce numerical mixing unless the control volumes are centered and the grid uniform.
Located in c_coefconv.c

Example: c: coefconvstep 1   Coefficient set 1 corresponds to convection.

coefcplus Determine the sum of the positive coefficients in a specified set.
Input: *jcoef, namecplus*

Set *namecplus* for each equation point equal to the sum of the positive coefficients for the *jcoef* set of coefficients.

Example: c: coefcplus 0 cplusa   sets *cplusa* based on coefficient set 0.

Needs: *noindfwpts, nocoefs, coef_n, coef_c*. Uses if available *itrange*. Sets: *namecplus*.

coefdt   Set coefficients for the time term.
Input: *namedt, jdt, nymidbias*

Set the *jdt* set of coefficients in array *coef_c* to correspond to the time term for the time step(s) in *namedt,* so that the change with time for property *C* over the control volume associated with point *m* is

$$\sum_{i=0}^{i<coef\_n[k]} coef\_c[k][i+jdt*coef\_n[k]]*\delta C[coef\_i[k][i]] = \sum_{sub-volumes} \frac{\delta \overline{C}}{\delta t_n}\delta Vol .$$

As for convection (coefconv), the time term is an integral over the sub-volumes associated with the point *m*.

The array *namedt* is contains the time steps, δt. If *namedt* has the dimension 1, the same time step is used over the entire spacial grid. If *namedt* has the dimension of a between-the-points array, the local δt's are used for each continuity control volumes.

The time term is discretized using linear profiles in space unless –
if *nymidbias* = 'y', non-center-point coefficients are folded in to give a more center-point discretization. This option is not tested in this version as of Oct 2014.
Time accuracy needs *namedt* with dimension 1. Also *nymidbias* not= 'y' unless the grid is uniform and the control volumes centered.
Note: The program was set up for 4-d array dimensions, id,jd,kd,nd, with the 4th dimension nd as the time dimension. This however has never been implemented. Only nd=1 has been tested, i.e., one time step at a time.

Example: c: coefdt dt 3 n   Coefficient set 3 corresponds to time term.

Needs: *namedt, rho, idim4d, wherep, noindfwpts, nocoefs, wherefw, whoisfw, match. cltp*. Uses if available, *itrange, roundoff*. Modifies: *coef_n, coef_i, coef_c*.

coeffix   Analyze and optionally fix a set of coefficients based on sum, pose, and dispersion limits.
Input: *jfrom, jto, limsumnz, poselim, displim, nomit,*
omit[i] for i=0,,,i<*nomit*

Omit from consideration by this routine grid points with types *omit*[i]. Arrays *sum, pose, disp* dimensioned on the grid points are set. They are respectively, the sum of

the coefficients, the center point coefficient over the sum of all positive coefficients, and the largest negative coefficient over the center point coefficient. *sum, pose, disp* are calculated before the coefficients are modified. Call coeffix twice to see the 'after' results. Analyze the *jfrom* set of coefficients in array *coef_c*. If *jto* >= 0, set the *jto* set of coefficients to be the 'fixed' values of the *jfrom* coefficients. (The *jfrom* coefficients are unchanged unless *jto=jfrom*.) To control how the coefficients are fixed:
a) Set *limsumnz* > 0 to modify the center point coefficient so that the sum of the coefficients for the point is identically zero. (The center point coefficient is always modified to make sure the sum is >= 0).
b) *C*orrect the coefficients, if necessary, so that the center point coefficients are >= *poselim* times the sum of the positive coefficients.
c) Correct the coefficients, if necessary, so that the most negative coefficient is >= *displim* times the center point coefficient. Note that *displim* is negative.
The method used to correct the coefficients is to use numerical mixing on a point-by-point basis, to reduce the non-center positive coefficients (all by a uniform fraction) and correspondingly increase the center point coefficient so that the required limits are passed. Note if *jfrom=jto* there may be some directional bias in the way the coefficients are fixed.

Example: c: coeffix 2 0 0 .4 –1.5 2 i w Fix set 2, putting results in set 0; ignore i and w points.

Needs: *idim4d, match, noindfwpts, whoisfw, nocoefs, wherefw, coef_n, coef_i, coef_c, clt*. Uses if available *itrange, roundoff*. Sets: *sum, pose, disp*. Modifies (if *jto*>=0): *coef_n, coef_i, coef_c*.

coefinit Create the main coefficient arrays.
Input: *numbercoefs*
numbercoefs = the number of different coefficient sets that will be needed for setting up the flow calculations. For example if you will need separate coefficient values for convection, laminar diffusion and time terms, set *numbercoefs* = 4, and alot the first set (index 0) to the final equations to be solved, and say index 1 to convection, index 2 to laminar diffusion and index 3 to the time term.

Example: c: coefinit 4

Needs: *noindfwpts*. Creates: *coef_n, coef_i, coef_c, nocoefs*, Sets *nocoefs*[0] = *numbercoefs*. The arrays *coef_n, coef_i, coef_c* are the main coefficient arrays and are dimensioned *noindfwpts*[0], so that there is a set of coefficients for each independent grid point for which equations will be needed.

coefrhs Update the right hand side of an equation using the coefficient array.
Input: *namev, namerhs, oldnew, jcoef, fac*
namev, name of the property array stored on the grid points.
namerhs, name of right hand side array stored for each independent grid point.
If *oldnew* begins with 'n' (or *namerhs* does not exist) create *namerhs* and initializes all values to zero before updating. Otherwise *oldnew* must begin with 'o' (for old).

For each independent grid point, i, subtract from the right-hand side, the sum of the *jcoef* set if coefficients times the property at the points, times the factor, *fac,*

$$namerhs[i] -= fac * \sum_{j=0}^{j<coef\_n[i]} coef\_c[i][j + coef\_n[i] * jcoef] * namev[coef\_i[i][j]].$$

Example: c: coefrhs U1up U1rhs old 3 -1. add up-time to rhs U1 equation.

Note: if *namev* contains multiple properties, such as array *bij*, *namerhs* is assumed to be correspondingly dimensioned and is updated for each property,

$$namerhs[i + k * noindfpwts[0]] -=$$
$$fac * \sum_{j=0}^{j<coef\_n[i]} coef\_c[i][j + coef\_n[i] * jcoef] * namev[coef\_i[i][j] + k * nogpts[0]]$$

for k=0, k< arraysize(*namev*)/*nogps*[0].

Needs: *noindfwpts, nogpts, coef_n, coef_i, coef_c, namev*. Use if available: *itrange, roundoff*. Update or set: *namerhs.*

coefvisc Set viscous coefficients using linear interpolation.

Input: *namevisc, jvisc*

Set the *jvisc* set of coefficients in *coef_c* to correspond to the viscous term where *namevisc* is the viscosity array The size of *namevisc* most be either the size of a between-the-points array corresponding to an isotropic viscosity, or 6 times that for a nonisotropic, μ_{ij}. The viscous term for a property C for the control volume about point m is

$$\sum_{i=0}^{i<coef\_n[k]} coef\_c[k][i + jvisc * coef\_n[k]] * C[coef\_i[k][i]] = -\int \frac{\partial}{\partial x_i} \mu_{ij} \frac{\partial C}{\partial x_j} dVol = -\oint \mu_{ij} \frac{\partial C}{\partial x_j} dA_i$$

where k (=*wherefw*[m]) is the equation corresponding to point *m,* and *coef_n*[k] is the number of points in the discretized equations.

The viscosity array is stored between the points, i.e. for the continuity control volumes and has dimensions (id-1)*(jd-1)*(kd-1)*nd*n with n=1 for isotropic viscosity and n=6 for nonisotropic viscosity. The same viscosity is used for each sub-area within the continuity control volume. For non-isotropic viscosities, the viscosity array contains the values for μ_{xx} for each cont. c.v. followed by μ_{yy}, μ_{zz}, μ_{xy}, μ_{xz}, μ_{yz} and the viscosity array is symmetric so that $\mu_{ij} = \mu_{ji}$. Each sub-area contributes to the equations for the two points on either side. $\partial C / \partial x_j$ is discretized using tri-linear profiles between the points.

Example: c: coefvisc vlam 2 uses viscosity vlam for coef. set 2.

Needs: *namevisc, idim4d, wherep, noindfwpts, wherefw, match, whoisfw, nocoefs, xyz, cvdc, cvdcdouble, xyzdouble*. Uses if available: *itrange, roundoff*. Modifies: *coef_n, coef_i; coef_c*.

coefviscstep Set viscous coefficients using biased interpolation.

Input: *namevisc, jvisc*

Same as coefvisc, except that the discretization for $\partial C/\partial x_j$ is biased towards the two points either side of each sub-area for stability.
Located in c_coefvisc.c.

Example: c: coefvisc vturb 3 uses viscosity vturb for coef. set 3.

coefzero Set specified coefficient set to zero for particular point types.

Input: *namecn, nameci, namecc, jrep, jzero, nycenter1, nczero,*
czero[j] for j=0,,, j<*nczero*

namecn, nameci, namecc are the names of the _n, _i, _c parts of the coefficient arrays. *jrep* is the number of coefficient sets in *_c*. (If *namecc* = *coef_c*, the stored variable, *noceofs*[0] is used instead of *jrep*.)
Set the *jzero* set of coefficients in _c to zero for the specified point types. *nczero* is number of point types for which the coefficients are set to zero, and *czero*[j[are the point types (1 character names). If *nycenter1* > 0, the equation center point coefficient is set to 1 for the modified equations (unless that equation has no coefficients).

Example: c: coefzero cpflop_n cpflop_i cpflop_c 3 2 0 1 i

Needs: *noindfwpts, whoisfw, nocoefs, clt, namecn, nameci*. Use if available: *itrange, roundoff*. Modifies: *namecc*.

comment Read and print a one line comment of up to 80 characters.

Input: *comment*

Example: c: comment This is my comment!

constant Set constants or arrays of constants.

Input: *name, type, length,*
values[j] for j=0,, j<*length*

name – name of array. *type* – array type = i,d,c, or s; i-integer, d-double, c-single character, s-names (strings of characters, wrap in quotes to set if the name contains spaces). Note that the values in arrays of type i, d or c may be modified using the command edit.

Examples: c: constant itrange i 2 0 0
c: constant roundoff d 1 1.e-9

contcpcdu Set coefficients of pressure correction equation based on δU.

Input: none

Sets coefficients *cpc* for pressure correction equations using abbreviated momentum equations for each continuity control volume. The first set of coefficients, corresponds to changes in continuity based on changes in velocity on the grid points,

$$\sum_{j=0}^{j<cpc\_n[k]} cpc\_c[k][j] * \delta P[cpc\_i[k][j]] = \oint_{c.v.\,k} \rho \delta U_i dA_i \ .$$

The second set of coefficients, give the changes using a mid-face (similar to Rhie-Chow) correction, so that the sum of the first and second set correspond to a mid-face corrections in velocity,

$$\sum_{j=0}^{j<cpc\_n[k]} \left(cpc\_c[k][j] + cpc\_c[k][j + cpc\_n[k]] + \right) * \delta P[cpc\_i[k][j]] = \oint_{c.v.\,k} \rho \delta U_i^{mid-face} dA_i$$

Note: the meaning of the first and second set of coefficients is changed by command contcpcfixed.

Example: c: contcpcdu

Needs: *idim4d, wherep, noindppts, rho, csym, clt, matchpc, cam, cplus, noindfpts, wherefw, cpflop_n, cpflop_i, cpflop_c, volmom, xyz, xyzp*. Uses if available: *itrange, roundoff*. Sets *cpc_n, cpc_i, cpc_c*.

contcpcexit Set pressure correction coefficients for the exit boundary condition.

Input: none

Use command exitinit to set the parameters used. Call after contcpcdu and contcpcfixed. If this routine is not used, the default is fixed exit pressure, or a repeating pressure (see variable *dpdx*).

Example: c: contcpcexit

If *flowout* exists, needs*: idim4d, wherep, noindppts, flowoutprop, cpc_n, cpc_i, cpc_c, wherefw, cam, cplus, rho, cpflop_n, cpflop_i, cpflop_c, xyz*. Uses if available: *itrange, roundoff*. Modifies: *cpc_n, cpc_i, cpc_c*. Located in exitsubs.c.

contcpcfixed Fix up *cpc* for stable pressure-correction equations.

Input: *facrc, facrcperm, poselim,displim,poselimperm,displimperm*

Modify the first and second set of coefficients in *cpc_c* so that the first set is the set to be solved by eqnsolvep and the second set corresponds to the permanent *P* corrections for continuity. Use command contcpcdu before this command.

First add *facrc* of "Rhie-Chow" correction to the coefficients to be solved and set the permanent part to be *facrcperm* of this correction:

$$cpc\_c[k][j] = cpc\_c[k][j] + facrc * cpc\_c[k][j + cpc\_n[k]],$$

$$cpc\_c[k][j + cpc\_n[k]] = facrcperm * cpc\_c[k][j + cpc\_n[k]].$$

Set *facrc* and *facrcperm* between 0 and 1 with *facrcperm<=facrc.*

Next check the coefficient set to be solved to see if each equation passes the well-posed equation limits: the center point coefficient divided by the sum of the positive coefficients greater that *poselimperm,*

$$cpc\_c[k][j_{center-point}] \Big/ \sum_{positive-coefficients} cpc\_c[k][j] > poselimperm,$$

and the most negative coefficient divided by the center point coefficient is greater than *displimperm*,

$$cpc\_c[k][j_{lowest,most-negative}]/cpc\_c[k][j_{center-point}] > displimperm.$$

When an equation is found which does not pass the limits, direct numerical mixing is added to reduce all the positive non-center point coefficients by the same fraction so that the limits are satisfied. As a non-center point coefficient is reduced, the center point of the equation is increased by the same amount. The equation corresponding the non-center point coefficient is modified similarly,

$$cpc\_c[k][j_{center}] += C_{add}, \quad cpc\_c[k][j_{positive}] -= C_{add},$$

$$cpc\_c[k_{eq-of-j-positve}][j_{center-this-eq}] += C_{add},$$

$$cpc\_c[k_{eq-of-j-positve}][j_{center-of-original-eq}] -= C_{add}.$$

For *poselimperm* and *displimperm* the corrections are made to both the coefficients to be solved and the permanent correction. The exercise is then repeated using *poselim* and *displim* modifying only the coeffients to be solved. Set *poselim* and *poselimperm* <=1, and *displim* and *displimperm* <= –1.

The continuity (pressure-correction) equations for incompressible flow are then:

$$\sum_{j=0}^{j<cpc\_n[k]} cpc\_c[k][j] * \delta p[cpc\_i[k][j]]$$

$$= -\oint_{c.v.k} \rho U_i dA_i - \sum_{j=0}^{j<cpc\_n[k]} cpc\_c[k][j+cpc\_n[k]] * pp[cpc\_i[k][j]]$$

The last term, the permanent *P* correction, is a non-physical addition to the continuity equation, except that *facrcperm*=0.5 may be approximately second order since the flux for each surface is the average of that for the corner points and the mid-face values. Using *facrcperm*=1 and *poslimperm*=1 allows convergence of the discretized continuity equations to machine zero, and gives a smooth pressure solution. However, the upfront error introduced by the permanent *P* correction significantly degrades both continuity and the pressure solution. Experience suggests that *facrcperm*=1 does little harm since the “Rhie-Chow” correction is used to interpolate between points on the surface of the continuity control volume whose corners are anchored by actual grid points. (This differs from many methods which use the approximation to interpolate normal to the control volume surface.)

The coefficients are reordered and *cpc1_n* is set so that only the first *cpc1_n*[k] coefficients are greater than roundoff, i.e. the center-point coefficient times tol, where tol=1.e-8 or *roundoff*[0], if *roundoff* is set. This allows the solution procedure for the pressure-correction equations to be more efficient.

Example: c: contcpcfixed 1. 1. 1. –1. 0 –20

This suggested set of parameters uses the full “Rhie-Chow” correction, and full numerical mixing in the coefficients for δp, but only minimal backup numerical mixing in the permanent *P* correction. Consistent with this the pressure correction equations should be solved a few (but not too many times) updating the right hand side inbetween.

Needs: *idim4d, noindppts, wherep, whoisp, cpc_n, cpc_i, cpc_c*. Uses if available: *itrange, roundoff*. Modifies: *cpc_c* (and *cpc_n, cpc_i* if necessary). Sets *cpc1_n*.

contcpcinlet Set pressure-correction coefficients for the inlet boundary.
Input: none
Set pressure correction equation coefficients to maintain specified velocity ratios at the inlet relative to a specified total pressure. Use command inletinit to set the parameters used. Coded for incompressible flow. Call after contcpcdu and contcpcfixed.

Example: c: contcpcinlet

If *flowinn* exits, needs: *idim4d, wherep, pp, rho, flowinprop*. Use if available: *itrange*. Modifies: *cpc_n, cpc_i, cpc_c*. Located in inletsubs.c.

contdu Update the velocities to satisfy continuity.
Input: none
Update the velocity, U_n, to satisfy continuity based on abbreviated momentum and the pressure changes calculated from the continuity equation. For each flow point, k, the change in the velocity is

$$\delta U_n[k] = -\frac{\sum_{j=0}^{j<cflop\_n[i]} cpflop\_c[i][j+(n-1)*cflop\_n[i]]*\delta p[cpflop\_i[i][j]]}{cam[i]+cplus[i]}$$

for n=1,2,3, and where i=*wherefw*[k]. See inletreset to update velocities on the inlet boundary if appropriate.

Example: c: contdu

Needs: *idim4d, match, wherefw, cpflop_n, cpflop_i, cpflop_c, cam, cplus, dp*. Uses if available: *itrange*. Updates: *U1, U2, U3*. Also updates if available: *dU1, dU2, dU3*.

contrhsexit Set r.h.s of continuity for exit boundary condition.
Input: none
Call after contrhsdu. Exit boundary condition parameters for continuity set by exitinit.

Example: c: contrhsexit

If *flowout* exists, needs: *idim4d, wherep, flowoutprop, rhns, pp, rho, U1, U2, U3*. Use if available: *itrange*. Modifies *rhsc*. Located in exitsubs.c.

contrhsp Add to the r.h.s of continuity the retained pressure modification.
Input: none
When called after contrhsu, it yields

$$rhsc[k] = -\oint_{c.v.\,k} \rho U_i dA_i - \sum_{j=0}^{j<cpc\_n[k]} cpc\_c[k][j+cpc\_n[k]] * pp[cpc\_i[k][j]]$$

Example: c: contrhsp

Needs: *noindppts, cpc_n, cpc_i, cpc_c, pp*. Use if available: *itrange*. Modifies *rhsc*.

contrhsu Set right hand side of continuity equation based on mass fluxes.
Input: none
Sets $rhsc[k] = -\oint_{c.v.\,k} \rho U_i dA_i$ for each continuity control volume.

Example: c: contrhsu

Needs: *idim4d, wherep, noindppts, U1, U2, U3, rho, cltp, xyz*. Uses if available: *itrange*. Sets: *rhsc*.

copy Copy an array.
Input: *namefrom, nameto*
Make a duplicate copy of the array *namefrom*, copying it to *nameto*. The *namefrom* array must exist and be of the type d(double), i(integer), or c(character).

Example: c: copy xyzp xyzpold

Needs: *namefrom*. Sets: *nameto* if *namefrom*

copy_mtop Copy between-points array to pressure-points array.
Input *namefrom, nameto, nysym*
Copy the values in between-points array *namefrom* to corresponding points in pressure-grid array *nameto*. Outside points in *nameto* are set to zero unless *nysym* >0; then they are copied from the inside, or averaged across repeating boundaries. Note this is a straight copy, it does not interpolate to the p-points locations. Array *namefrom* must be of type double.

Example: c: copy_mtop srate sratep 1

Needs: *idim4d, namefrom* and if *nysym*>0 *csym*. Sets: *nameto*.

copy_ptom Copy pressure-points array to between-points array.
Input *namefrom, nameto*
Copy the values in p-points array *namefrom* to corresponding points between-points array *nameto*. Note this is a straight copy, it does not interpolate. Array *namefrom* must be of type double.

Example: c: copy_ptom pp pm

Needs: *idim4d, namefrom*. Sets: *nameto*.

copypart Copy sections of an array.
Input for the 'from' array, then for the 'to' array:

name, *ndims*,
dimname[j], *id*[j], *is*[j], *ie*[j] for j=0,, j<*ndims*

name – array name.
ndims – number of dimensions, must have *ndims*<=10.
dimname[j] – a 1-character name for the dimension.
id[j] – the dimension size.
is[j] – starting index for copy
ie[j] – last index for copy
If *dimname*[j] for the dimension is *i, j, k,* or *t, idim4d*[0, 1, 2, or 3 respectively] is added to *id* (if array *idim4d* exists) to obtain the dimension size.

If the 'to' Array does not exist it is created. Values in the 'from' array from *isfrom* to *iefrom* are copied to the 'to' array from *isto* to *ieto*. *is* and *ie* are limited to the dimension size. Note that *is* and *ie* are specified 'Fortran' style so that the first value is specified by *is*=1. Sufficient values are copied to fill the specified parts of the 'to' array. As the 'to' array indices are indexed, the index with the corresponding name in the 'from' array is indexed to determine the value to be copied.

Notes: Arrays of type d(double), f(float), i(integer) and c(character) may be copied. Also the 'to' array must be of the same type as the 'from' array.

Example: c: copypart xyz 5 i 0 1 5 j 0 8 8 k 0 1 2 t 0 1 2 L 3 1 3
xyz 5 i 0 1 5 j 0 7 7 k 0 1 2 t 0 1 2 L 3 1 3

The x,y,z values at the 8th point in the j direction are copied to the 7th point, for the range covering the first 5 i points, the first 2 k point and the first 2 t points. For i,j,k,t, the dimensions are understood to be those of the grid. The dimension for L is 3 (x,y,z for this array).

Example: c: copypart xyz 5 i 0 1 200 j 0 1 200 k 0 1 200 t 0 1 1 L 3 2 2
y 3 i 0 1 200 j 0 1 200 k 0 1 200

An array *y* is created containing the first set (t index) of y values in the array *xyz.*

Example: c: copypart pp 3 i 1 2 2 j 1 1 1 k 1 1 1
pcon 3 i 1 1 200 j 1 1 200 k 1 1 200

The value of pp at the second point in the i direction, and first in the j, and k directions is used to fill the array pcon ('fill' - provided the i, j, and k dimensions are less that 200).

Needs: *namefrom*. Sets or update *nameto*.

cvdcinit Initialize the control volume divide arrays.
Input: *flim, limd*,
typefrom[j], *typeto*[j], *flimd*[j] for j=0,, j<*limd*
The dividing up of the continuity control volumes to form contributions to the control volumes for momentum and other primary variable equations are initialized at 0.5, i.e. in the middle (half way between 0 and 1). The movement of these divide points are limited by the input parameters.

flim provides a general limitation so that *cvdc* values are >= *flim, and* <1-*flim*. Set *flim*>0 (to make sure all points have a control volume) and <0.5, a value which would fix the control volumes as centered. This parameter is used by cvdcreset.

limd, the number of point type restrictions to be applied. When the grid points that border a continuity control volume are of more than one type (for example near wall volumes) it can be useful to apply limitations. For each restriction, the relative divide from *typefrom*[j] to *typeto*[j] is greater than *flimd*[j].

Example: c: cvdcinit .001 3 f i .999 f w .5 w f .5

The restriction 'f i .999' forces the divide point to move almost all the way from points of type f towards points of type i. The double restriction 'f w .5' and 'w f .5' causes the control volumes to be divided evenly between w (wall points) and f (adjacent near wall flow points). This may be appropriate for viscous flow with a close near wall point, especially for heat transfer calculations.

Needs: *idim4d, wherep, clt, match, matchpc, matchside, csym*. Sets: *cvdc, cvdcdouble*. Located in cvdcsubs.c.

cvdcparm Reread the control volume divide parameters.
Input: *flim, limd*,
typefrom[j], *typeto*[j], *flimd*[j] for j=0,, j<*limd*
See cvdcinit for parameter description. No arrays set or modified. Located in cvdcsubs.c.
Example: c: cvdcparm .001 3 f i .999 f w .5 w f .5

cvdcreset Recalculate the control volume divide arrays.
Input: *nameisovisc, namedt, t2clim*
The control volume divide arrays, *cvdc* and *cvdcdouble* are reset based on
(a) convection using the current velocity field,
(b) the parameter limitations described in cvdcinit,
(c) an isotropic viscosity array, if array *nameisovisc* exists, and
(d) a time step array, if *namedt* exists,
(e) the parameter, *t2clim*, limiting the influence of the time step to be no more than *t2clim* times the influence of convection.

The purpose of dividing up the continuity control volumes in a variable way is to try to obtain naturally stable equations with a strong center point coefficient without having to add numerical mixing. If the equations are dominated by viscosity, the natural choice is centered control volumes, i.e. an equal breakup into 8 parts with each 1/8th assigned to each corner point. If however convection dominates, the stable choice is to assign the volume to the points on the downstream side, i.e. as is commonly used in space marching boundary layer codes.

The algorithm: The convection term, $\overline{\rho U_i}\frac{\partial}{\partial x_i}$, for the continuity control volume is approximated and turned into a set of coefficients for the 8 (2x2x2) corner points,

$C_{i\pm,j\pm,k\pm}$ for the + and – sides of the i, j, and k grid directions, Positive contributions to coefficients from the isotropic viscosity, V_n (n=i, j or k), and the time term, T are then estimated. For the i grid direction,

$$cvdc_i = .5\left(1+\frac{\sum_{j\pm,k\pm}(-C_{i+,j\pm,k\pm} + -C_{i-,j\pm,k\pm})}{S+V_i+\min(t2clim*S,T)}\right),$$

and similarly for the j and k directions. Note that *cvdc* varies between 0 and 1, with 0.5 representing an equal breakup, i.e. centered control volumes. *cvdc* is then averaged between neighboring control volumes to obtain values for edges and faces stored in array *cvdcdouble*. The result is that each continuity control volume is broken into 8 parts, one for each corner point, with faces that match the faces of the neighboring control volumes.

Example: c: cvdcreset vlam dt 10.

Needs: *idim4d, cvdc, cvdcdouble, wherep, clt, U1, U2, U3, rho, xyz, match, matchpc, matchside, csym*. Use if available: *vlam, dt, itrange*. Modifies: *cvdc, cvdcdouble*. Located in cvdcsubs.c.

ddtall Calculate a variety of $1/\delta t$, timescales.

Input: None

Set between the points array *ddtall*[im,jm,km,6], for six different $1/\delta t$, for each non-zero, non-solid continuity control volume.

n=0 convection - CFL number, e.g. for Cartesian, $\max(U_1/\delta x, U_2/\delta y, U_3/\delta z)$.

n=1 laminar viscosity, if *vlam* exists e.g. for Cartesian, $\dfrac{vlam}{\rho}\left(\dfrac{1}{\delta x^2}+\dfrac{1}{\delta y^2}+\dfrac{1}{\delta z^2}\right)$.

n=3 absolute vorticity, $\sqrt{2W_{ij}W_{ij}}$

n=4 strain rate, $\sqrt{2S_{ij}S_{ij}}$

n=5 $\max\limits_{i=1,3}\sqrt{\left|\sum_{k=1}^{3}\left(\dfrac{\partial U_i}{\partial x_k}+2\varepsilon_{ink}\Omega_n\right)\left(\dfrac{\partial U_k}{\partial x_i}+2\varepsilon_{kni}\Omega_n\right)\right|}$ a center-point relaxation $1/\delta t$ useful for converging steady separated flow.

n=2 $\max\limits_{i=1,3}\sqrt{\max\left(0,\sum_{k=1}^{3}\left(\dfrac{\partial U_i}{\partial x_k}+2\varepsilon_{ink}\Omega_n\right)\left(\dfrac{\partial U_k}{\partial x_i}+2\varepsilon_{kni}\Omega_n\right)\right)}$ an alternate ‘safe’ $1/\delta t$.

Also set *ddtmax*[0] as the maximum $1/\delta t$ for the CFL condition over the entire grid.

Example: c: ddtall

Needs: *idim4d, wherep, U1, U2, U3, rho, xyz, matchpc, cltp*. Use if available: *vlam, zrotation*. Sets: *ddtall, ddtmax.*

edit Edit an array of type character, integer or double.

Input: *name, type, ndims*
1charname[j], *id*[j],   for j=0,, j<*ndims*
action, value,
is[j],*is*[j]   for j=0,, j<ndims
repeat reading the *action, value, is, ie,* until an invalid action occurs

Edit array *name*. *name* will be created if it doesn't exist and filled with zero. The *type* must be c for character, i for integer or d for double. *ndims* is the number of dimensions for the array. *id*[j] is the size of each dimension. If *1charname* for the dimension is *i, j, k,* or *t, idim4d*[0, 1, 2, or 3] is added to *id* (if array *idim4d* exists) to obtain the dimension size.
The first letter of *action* is all that is used. The possible choices are s for set, a for add and m for multiply. For character arrays the only choice is s for set. The action is taken for the range *is*[j], through *ie*[j], where *is*[j]=1 corresponds to the first value. *is* and *ie* are both bounded by the corresponding dimension.
If the *action* is set, the specified range is set to *value,* if *action*=a, *value* is added, Edit an on-the-points array using *abcd* and if *action*=m*,* the specified range is multiplied by *value.*

Example:

```
c: edit clt c 4  i 0   j 0   k 0   t 0
        set i 1 1   1 100   1 100   1 100
        set e 100 100   1 100   1 100   1 100
        end
```

If all the dimensions are less that 100, this sets *clt* on the first i-plane equal to i, and then equal to e on the last i-plane.

Uses if available: *idim4d*. Sets or updates: *name.*

editabcd Edit an on-the-points array using *abcd* to determine the range.

Input: *name, type, mrep*
action, value, as,ae,bs,be,cs,ce,ds,de
if *mrep*>1: *ms,me*
repeat reading *action* through *de* (or *me*) until an invalid action occurs.

Edit array *name*. The dimension of *name* are the grid dimensions times *mrep. name* will be created if it doesn't exist and filled with zero. The *type* must be c for character, i for integer or d for double. The first letter of *action* is all that is used. The possible choices are s for set, a for add and m for multiply. For character arrays the only choice is s for set. The action is taken for the range determined by finding *as, ae* in the 'a' part of *abcd,* and similarly for *b, c,* and d, and for *m*=*ms, m*<*me*. Note the first value for *m* is 0 (C code style). If the *action*=s, the specified range is set to *value,* if *action*=a, *value* is added, and if *action*=m*,* the specified range is multiplied by *value.*

<u>Example</u>:   c: editabcd cc d 1   set 10.   1. 1.2 0. 1. 0. 1. 0. 1.   end

sets cc equal to 10 for the range a=1 through 1.2, and b,c,d =0,1.

Needs: *idim4d*. Update *name.*

end Terminate the program.

end is not a separate routine but part of the main program, m4d.c

<u>Example</u>: c: end

eqnsolvebij Solve bij equations.

<u>Input</u>: *ncab, oldnewch, nomit,*

omit[j]   for j=0,, j<nomit

nameproc, iterproc

repeat reading last line until *nameproc* begins with e or *iterproc*=0

ncab, name of a center point relaxation array, dimensioned *noindfwpts*. If *ncab* does not exist, the array *cambij* is used. This array is (temporarily) added to the center point coeffcient to provide the relaxation.

If *oldnewch* begins with 'n' (for new) all values in the array *dbij* are initialized to zero. Otherwise *oldnewch* must begin with 'o' (for old).

nomit, number of point types to be omitted from consideration by this routine.

omit[j], the point types (1 character names) to be omitted.

nameproc, solution procedure sequence to be used.

iterproc, number of times the procedure sequence is to be repeated.

Procedure choices: c, center point update based on the sum of the positive coefficients in *coef_c* and the source term center point coefficients, *beqd*;

i, j, k, or t, spacial connected update for each component of *dbij* separately using tri-diagonal matrix algorithm strips. Options i, j, or k connects together lines of points in the i, j, or k grid directions. Option t forms natural strings based on the coefficients.

Set of *dbij* equations solved, for j=0, j<6:

$$\sum_{k=0}^{k<coef\_n[i]} coef\_c[i][k]*dbij[coef\_i[i][k]+j*N_{pts}]$$

$$-\sum_{m=0}^{m<6} dbij[whois[i]+m*N_{pts}]*beqd[i+(j+6*m)*N_{eqs}]$$

$$+ncab[i]*dbij[whois[i]+j*N_{pts}] \qquad = beqm[i+N_{eqs}*j]$$

where $N_{eqs} = noindfwpts[0]$ is the number of equation points and $N_{pts} = nogpts[0]$ is the number of grid points. Changes to *dbij* are not made for equation points with no coefficients, and those with point types listed in *omit*. Omission of appropriate shear components on symmetry planes is also made if the array *csym* exists. Because the equations for the normal components of bij sum to zero, the equations for db_{11} are replaced by $db_{11} = -db_{22} - db_{33} - b_{11} - b_{22} - b_{33}$.

Example: c: eqnsolvebij cab new 2 i w
          tc 10 end 0

initializes the *dbij* values to zero, (A) use the t procedure to determine *ddbij* , update *dbij* and determine the current equation residuals, use the c procedure to determine *ddbij*, update *dbij* and determine the current equation residuals. Repeat from (A) 9 (10-1) times.

Needs: *idim4d, coef_n, coef_i, coef_c, nogpts, noindfwpts, wherefw, match, clt, beqm, beqd, bij*. Uses if available: *itrange, csym,ncab*. If *ncab* is not available, needs *cambij*. Sets or modifies: *dbij*.

eqnsolvep Solve pressure correction equations.

Input: *nameproc*, *iterproc*,

repeat reading last line until *nameproc* begins with e or *iterproc*=0

nameproc, solution procedure name sequence to be used.
iterproc, number of times the procedure sequence is to be repeated.

Procedure choices: c, center point update based on the sum of the positive coefficients; t, the coefficient set is formed into natural strings based on the values and each string is solved using a tri-diagonal matrix algorithm; i, j, or k, solve using a tri-diagonal matrix algorithm for i, j or k lines of points; I, J, or K, solve the equations summed over i, j, or k planes to give a 1-d update of the pressure (J and K options may be untested); b, exit-boundary equations; any other letter, using blocks set up by blocksetabc.

Equations solved: $$\sum_{k=0}^{k<cpc\_n[i]} cpc\_c[i][k]\, dp[cpc\_i[i][k]] = rhsc[i] \ ,$$

for all $dp[whoisp[i]]$ in the *itrange* and for which there are positive coefficients, except points with $cltp[whoisp[i]] =$'o'. The omission of type 'o' points allows for 1 point to be omitted when a repeating boundary in the flow direction is used so that the set of continuity equations is not overspecified. After the solution procedure ends, values at sleeping pressure points are calculated, and if *ipfix* exists *dp* is set to zero at this point and all other are changed relative to it.

Example: c: eqnsolvep   I 1 ijk 100 c 1 end 0

Needs: *idim4d, noppts, noindppts, wherep, whoisp, cpc_n, cpc_i, cpc_c, rhsc, matchpc, cpsleep, cltp*, and e.g. *blockijkx* for and any block procedure 'x' specified in *nameproc*. Use if available: *itrange, ipeqexit, ipfix*. Use *cpc1_n* if available instead of *cpc_n*. Sets: *dp, eqnerrp*.

eqnsolves Solve on-the-grid-points equations, omitting selected point types.

Input: *nrhs, ncam, nchange, oldnewch, nomit,*

omit[j]   for j=0,, j<nomit

nameproc, *iterproc*

repeat reading last line until *nameproc* begins with e or *iterproc*=0
nrhs, name of right hand side array, dimensioned *noindfwpts*.
ncam, name of a center point relaxation array, dimensioned *noindfwpts.*
nchange, name of property change array, dimensioned *nogpts*.
If *oldnewch* begins with ‘n’ all values in the array *nchange* are initialized to zero. Otherwise *oldnewch* must begin with ‘o’ (for old).
nomit, number of point types to be omitted from consideration by this routine.
omit[j], the point types (1 character names) to be omitted.
nameproc, solution procedure sequence.
iterproc, number of times the procedure sequence is to be repeated.
Procedure choices: c, center point update based on the sum of the positive coefficients; t, the coefficient set is formed into natural strings based on the values and each string is solved using a tri-diagonal matrix algorithm; i, j, or k, solve using a tri-diagonal matrix algorithm for i, j or k lines of points.

Equations solved:

$$\sum_{k=0}^{k<coef\_n[i]} coef\_c[i][k]\, change[coef\_i[i][k]] + cam[i]change[whois[i]] = rhs[i],$$

for all *change*[*whois*[*i*]] in the *itrange* and for which there are positive coefficients and for which the point types do not match those in *omit*. After the solution procedure ends, values at matching points are set.

Example: c: eqnsolves rhsU1 cam dU1 new 2 i w
c 1 end 0

Example: c: eqnsolves rhscc "" cc old 2 i I
tc 20 end 0

Needs: *nrhs, coef_n, coef_i, coef_c, nogpts, idim4d, noindfwpts, wherefw, whoisfw, match, clt*. Uses if available: *ncam, itrange*. Sets or updates: *nchange*.

eqnupdatem Update the velocity from the momentum equations.
Input: *nysavch,nomit*
omit[j] for j=0,, j<nomit
Do a single center-point update of the velocities equivalent to eqnsolves with procedure c 1 and then adding the changes to the velocity components, *U1, U2, U3.* If *nysavch*>0, arrays *dU1, dU2, dU3* are created and saved. No changes are made for point types *omit*[j] (input single character names). *csym* is used if available to omit changes as appropriate for symmetry planes.

Example: c: eqnupdatem 1 2 i w

Needs: *idim4d, nogpts, noindfwpts, wherefw, whoisfw, match, clt, coef_n, coef_i, coef_c*. Uses if available: *cam, csym, itrange*. Modifies: *U1, U2, U2.* Sets if *nysavch*>0: *dU1, dU2, dU2.*

eqppts2ppts Set a p-points array from an equation p-points array.

Input: *nameveq, namevg, nymatch*

nameveq is the name of the equation p-points array and *namevg* is the name of the p-points array. Set *nymatch* > 0 to resolve sleeping p-points. If *nymatch*=1, sleeping points are set with simple averages or if *nymatch*>1, using *cpsleep*.

Example: c: eqppts2ppts rhsc rhscp 0

Needs: *nameveq, noindppts, noppts, matcpc, whoisp* and if *nymatch*>1, *cpsleep*. Sets *namevg*.

eqpts2gpts Set an on-the-grid-points array from an equation points array.

Input: *nameveq, namevg, nymatch*

nameveq is the name of the equation points array and *namevg* is the name of the on the points array. Set *nymatch* > 0 to resolve matching points.

Example: c: eqpts2gpts rhsU1 rhsU1g 1

Needs: *nameveq, idim4d, noindfwpts, match, whoisfw*. Sets *namevg*.

exitinit Initialize parameters for the exit pressure boundary condition.

Input; *nout*

for j=0,, j<*nout*: *type*[j], *dL*[j], (*ist*[j][n], n=0,,n<4),
(*iend*[j][n], n=0,,n<4), (*ifix*[j][n], n=0,,n<4) *value*[j]

For each of the *nout* exit boundary regions, specify:
type[j], the type of exit boundary = 1, 3 or 4, see below.
dL, the grid direction (1,2, or 3 for i,j, or k) signed (+–) to point into the flow domain.
The exit boundary region extends from pressure grid point indices *ist*[j] through *iend*[j[. Note that the indices are read Fortran style with the value for the first point = 1. Set *ist*[j][n]=*iend*[j][n] in the grid direction normal to the exit boundary.
ifix[j] is a point on the exit boundary.
value[j] is the fixed mass flow rate for *type* =3 or 4.
The code limits *ist, iend* and *ifix* to the pressure grid dimensions.
type=1: unchanged fixed pressure at point *ifix*, with a uniform change in pressure between the exit plane and the neighboring interior plane.
type=3: flow rate across exit fixed at *value*, with a uniform change in pressure between the exit plane and the neighboring interior plane.
type=4: flow rate across exit fixed at *value*, with a uniform change in pressure across the exit.
The companion commands which implement the boundary condition are contcpcexit and contrhsexit.

Example: c: exitinit 1 1 –1 200 1 1 1 200 200 200 1 200 2 4 1 .7

Needs: *idim4d, matchpc*. Sets: *flowout, flowoutprop*. Located in exitsubs.c.

function Take trigonomic or other functions of an array.

Input; *namev0, namefunction, namev1*

namev2 if needed by the function

Functions needing one input array, *namefunction* = tan, sin, cos, tanh, tandeg, log, log10, exp, abs.
Functions needing two input arrays: arctan2, arctan2deg, max, min.
If *namev0* does not exist, it is created with the larger of the sizes of *namev1* or *namev2*. *namev1* or *namev2* may also be constants (arrays of dimension 1).

Examples: c: function v0 exp v1 sets v0[k]=exp(v1[k]), k=0,k<arraysize(v1).

c: function v0 max v0 ZERO with ZERO dimensioned 1

gives v0[k]=max(v0[k],ZERO[0]), k=0,k<arraysize(v0).

Log or log10 of a negative value is set to the log of the absolute value, with a printed warning. Log or log10 of zero is set to –50, with a printed warning.

Needs: *namev1* and for functions with 2 arguments, *namev2*. Sets: *namev0*.

geom8print Print geometry parameters calculated by geom8 routines.
Input: *is*[j],*ie*[j] for j=0,, j<4
The range of the 4 grid indices, *is*[j],*ie*[j], are for the lower corner of the 'half' size control volumes in array *xyzdouble*. First point at 0,0,0,0.

Example: c: geom8print 0 10 0 10 0 1 0 1 0 1 for first 10 half c.v in i,j directions

Needs: *idim4d, xyz, cvdc, cvdcdouble*. Uses if available: *xyzdouble*. Located in geom8subs.c.

geomcprint Print geometry parameters calculated by geomc routines.
Input: *is*[j],*ie*[j] for j=0,, j<4
The range of the 4 grid indices, *is*[j],*ie*[j], are for the lower corner of the continuity control volumes in array *xyz*. First point at 0,0,0,0.

Example: c: geomcprint 0 10 0 10 0 1 0 1 0 1 for first 10 cont. c.v in i,j directions

Needs: *idim4d, xyz*. Located in geomcsubs.c

gpts2eqpts Make an equation point array from a grid point array.
Input: *namevg, nameveq*
Sets *nameveq*[k]=*namevg*[*whoisfw*[k]], k=0,*noindfwpts*[0].

Example: c: gpts2eqpts U1 u1eq

Needs: *namevg, whoisfw, noindfwpts*. Sets: *nameveq*.

gradprop Take gradients of properties stored on the grid points.
Input: *per*,
name until *name*=""
For each property *name* which must be dimensioned *nogpts*[0], calculate between-the-points values of grad *name*, $\frac{d\,name}{dx_i}$. If *per*=3 the results are stored in 3

separate arrays with *ddx1, ddx2 and ddx3* appended to *name*. If *per=1*, the results are stored in 1 array with *ddxi* appended to name (all values of $d\,name/dx_1$, followed by $d\,name/dx_2$, then $d\,name/dx_3$.) If *roundoffd* exists components of each gradient, which are less than *roundoff*[0] times the sum of the absolute values of the components, are set to zero. For non-valid (zero volume) continuity control volumes the results are zero, unless *cpsleepm* exists and they can be found by interpolation.

Example: c: gradprop 3 U1 ""   gives U1ddx1, U1ddx2, U1ddx3

Needs: *idim4d, wherep, xyz,* each *name*. Uses if available *cpsleepm, roundoffd*. If *per*=3, sets each *nameddx1, nameddx2, nameddx3*. If *per*=1 sets each *nameddxi*.

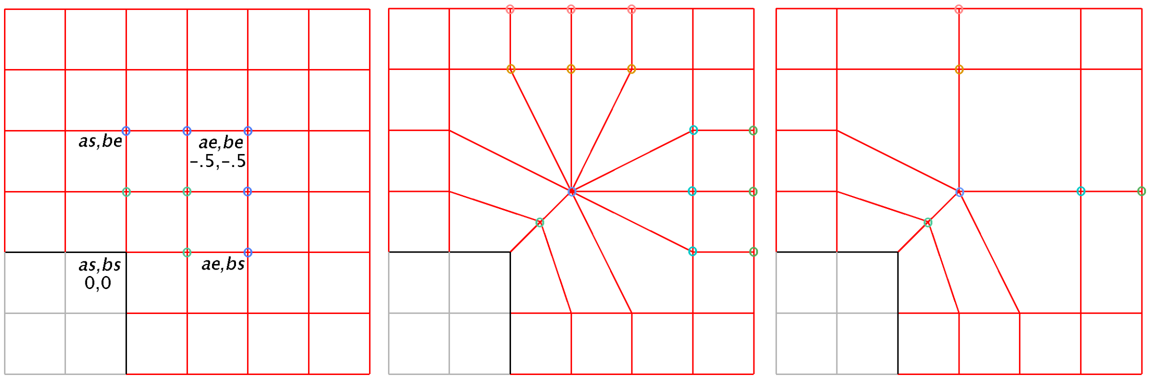
gridcorner Fix grid at corners to be effectively O-grid instead of box-grid.

Input: *cname*, *as*, *ae*, *bs*, *be*, *cs*, *ce*

Repeat until first character in *cname* = *'e'*

So far *cname*='ab' is the only corner type implemented. The corner is specified by 1-d 'grid' values in the array *abcd*. The corner begins at a=*as*, b=*bs,* and extends to a=*ae*, b=*be*. The routine collapses the box corner into an O-corner making sure the hidden coincident points are collocated. In the following example gridcorner is used to bring together points outside the corner of the box. Then gridvarmod is used to collapse the extraneous lines. The result is an O-grid corner imbedded in a box mesh. If points are added to what is already an imbedded O-grid, use gridcorner again to realign hidden points.

Box grid   after gridcorner   after gridvarmod



Example: c: gridcorner ab 0 -.5 0 -.5 0 1 end

c: gridvarmod "" a omit 0 -1 -.5 1 0 1 a omit -.5 1 -.5 1 0 1
b omit -.5 1 0 -1 0 1 b omit -.5 1 -.5 1 0 1 end

Needs: *idim4d, xyz. abcd*. Modifies: *xyz*. Sets: *ijkcorner*.

gridfrommefp Make a 4d grid starting with a MEFP style 3d grid.

Input: *mefpgridfilename,*

idim4d[j] for j=0,, j<4

if *idim4d*[0]>0, read *a*[j] for j=0,,j<*idim4d*[0]

if *idim4d*[1]>0, read *b*[j] for j=0,,j<*idim4d*[1]

if *idim4d*[2]>0, read *c*[j] for j=0,,j<*idim4d*[2]
d[j] for j=0,,j<*idim4d*[3]

The MEFP style 3d grid is read from file *mefpgridfilename*.

Input from file *mefpgridfilename*:

1 line comment
igm,jgm,kgm, (grid dimensions), lcoor (assumed to be 1)
(bm(j), j=1,jgm), (cm(k), k=1,kgm)
(am(i), (ltm(i,j,k),x(i,j,k),y(i,j,k),z(i,j,k),j=1,jgm, k=1,kgm),i=1,igm)

idim4d[j] are the 4 dimensions for the calculation grid, 3 space and time. If any of the space dimensions are set to zero, they are copied from the MEFP grid. Final space dimensions need to be >=2, while the time dimension is >=1. *a*[j], *b*[j] and *c*[j] are used to interpolate in the MEFP grid to set the x,y,z values. The same x,y,z values are set for all values of the time index.
gridfrommefp stores *a,b,c,* and *d* sequentially in array *abcd*. gridfrommefp also sets a point type array, *clt* with the character 'w' indicating a wall point (ltm=3,4), 'f' a flow point (ltm=1,2) and 's' an internal solid point (ltm=5).

Example: c: gridfrommefp ellipse1.5

```
40 27 2 1
-6 -5.5 -5. -4.5 -4 -3.5 -3 -2.5 -2 -1.5
        -1.25 -1 -0.8 -0.6 -0.4 -0.2 -0.05 0 0.1 0.2
        0.3 0.4 0.5 0.6 0.7 0.8 0.9 1 1.1 1.2
        1.3 1.5 2 2.5 3 4 5 6 7 8.25
0 0.0817 0.185 0.278 0.395 0.509 0.619 0.724 0.824 0.912
        1 1.05 1.2 1.4 1.6 1.8 2 2.25 2.5 3
        4 5 6 7.1 8.2 9.3 10.4
0 1
0.   / input master name grid_dims a,b,c,d
```

Sets *idim4d, abcd, xyz, clt.* Use if available *roundoff* (for ltm interpolation).

gridmatch Determine grid points that have matching locations.

Input: none

Sets array *match* so that all points which will have the same locations (coincident points to within *roundoff,* and repeating boundaries) give the same value for *match*[j]. Coincident points in the i, j and k grid directions as well as repeating boundaries are considered. If *match*[k]=k it represents itself, i.e. it is a primary point. Grid corners in i,j plane, where the box grid is effectively turned into a C-grid, are checked for with the sleeping points corrected for consistency if needed. Note that C-grid corners must have overall, $\delta i = \delta j$ (see m4d.geometry). Dependent grid points are determined from *match* so that *whoelse is* set.
Sets *idim4dp*, the dimensions of the p-grid. Sets *matchpc*, two active p-point indices to be used to interpolate for sleeping p-points. *matchpc* is set for all p-points. The first index is −1 if the p-point is not sleeping. Also for the p-points a six-sided

character array, *matchside* is set indicating if, and how sides of sleeping continuity control volumes match up.

Example: c: gridmatch

Needs: *idim4d, csym, xyz*. Uses if available: *roundoff, t4d*. Sets grid info *match, nogpts, whoelse, idim4dp, matchpc, noppts, matchside*. Modifies round-off inconsistencies in *xyz, t4d*, if any.

gridtomefp Dump 3d grid for one time index, *it*.

Input: *filename, it*

The format corresponds to that needed by the Grid option of the graphics program Pictw. This is the same as the format read from file *mefpgridfilename* in command gridfrommefp.

Example: c: gridtomefp gridfile 0

Needs *xyz, abcd, idim4d, clt*.

gridvarmod Modify the grid and on-the-points variables using *abcd*.

Input: *name* until *name*=""

cmod, fororo, as,ae,bs,be,cs,ce

repeat read *cmod, etc.* until *cmod* is not equal to a, b, c,

For *cmod* =a, b, or c, move parts of an i, j, or k grid plane a fractional distance towards a neighboring one, and correspondingly interpolate the variables *name* to the new locations.

For *cmod*=a, *as* is the plane to be moved and the fractional distance, *frac* is set to *ae*. If *frac* = –1, the points are made coincident with those on the previous plane, if *frac* = 1, the points are made coincident with those on the next plane, otherwise, –1 < *frac* < 1, the specified fractional move towards the neighboring plane is made. If *fororo*=f, move the plane for the range b=*bs,,be* and c=*cs,,ce*. If *fororo* is not equal to 'f', the move is made for all but the specified range.

Similarly if *cmod*=b, the plane moved is *bs*, *frac*=*be* and the range is a=*as,,ae* and *cs,,ce*. And similarly for *cmod*=c.

If there are points coincident with the points being moved, they are also moved. Note, no *a,b,c*, values for the planes are changed.

Example: c: gridvarmod "" a omit 0 -1 -.5 1 0 1 a omit -.5 1 -.5 1 0 1
b omit -.5 1 0 -1 0 1 b omit -.5 1 -.5 1 0 1 end

This example, shown under in the description of the command gridcorner, modifies no variables but moves the grid plane at a=0 to be coincident with the previous i-plane for all but the range b= –0.5 through 1, and c=0 through 1. Similarly the grid line at a= –0.5 is moved to be coincident with the next i-plane. Grid planes in the b direction are then similarly moved.

Note that all the variables, *name*, must be on-the-points variables. This however may include multiple variables such as *bij* which has the dimensions of the grid times 6.

Needs: *idim4d, abcd, xyz*. Modifies: *xyz,* all variables *name.*

if Conditionally find a string to omit sections of an input file.

Input: *nif1, leq, nif2,*
ifpass, if *ifpass*="find" read *stringp,*
iffail, if *iffail*="find" read *stringf*

nif1 and *nif2* are either the names of arrays of type integer or double (but only the first values are used) or integer or floating point constants.
leg is the single character, <, =, > or %. The operator % may only be used if both arrays are of type integer.
If the conditional statement is true, and if *ifpass* equals "find", the string *stringp* is found in the input file and reading of commands will continue below that. If the conditional statement is false and if *iffail* equals "find", the string *stringf* is found in the input file and reading of commands continues below that.
If either of the arrays *nif1* or *nif2* does not exist and cannot be interpreted as constants, the conditional statement fails.

Example:
```
c: if  TEN  >  ZERO  find  endinit  continue
c: constant TEN i 1 10
c: constant ZERO i 1 0
endinit
```
will look for endinit if TEN[0] > ZERO[0]. If TEN or ZERO do not exist they will be set.

Example:
```
c: if ITER  %  10  continue  find  endten
c: infile inn.each10iter 1
endten
```
will read the input file inn.each10iter if the number of iterations, ITER[0], is evenly divisible by 10.

infile Read command input from a different file.

Input: *filename, loop*

Read command input from *filename* repeating it *loop* times. When the loop is finished, command input returns to the input file it was previously reading from and proceeds with the next command in that file. Up to 20 input files may be stacked. infile is not a separate routine but part of m4d.c

Example:

Standard input file:
```
c: infile initc 1
c: infile pairs 50
c: infile dump 1
c: end
```
File *pairs*:
```
c:  infile mom 2
c: infile ave 1
```

This executes the commands in *initc*, then *mom* twice, then *ave* once, then continues to repeat the commands in *mom* twice then *ave* once until it has done this a total of 50 times. When that is complete it executes the command in *dump*.

inletinit Read information to set a total pressure/velocity ratio inlet boundary.

Input: *nin*

for k=0,,k<*nin* read

(*is*[j][k],*ie*[j][k] for j=0,,j<4),*pt*[k],*U1r*[k],*U2r*[k],*U3r*[k]

For each of the *nin* inlet regions, the region extends from *is* through *ie,* where *is* and *ie* are specified with the first point at 1,1,1,1. *pt* is the reference total pressure for the region, and the velocity components are set relative to the potential flow velocity (for incompressible flow), U1 = U_potential flow * U1r, and similarly for U2 and U3. This information for the inlet boundary is stored in arrays *flowinn* and *flowinprop* which are described in m4d.variables. If U1r, U2r and U3r are all zero for any region, *pt*[k],*U1r*[k],*U2r*[k],*U3r*[k] are determined from the current values with separate velocity ratios set for each grid point in the inlet region

Example; c:inletinit 1 1 1 1 200 1 200 1 200 0. .9397 .3420 0

sets a total pressure of 0, an inlet velocity corresponding to potential flow but at an angle of 20 degrees from the x-direction.
The companion command which use this information are contcpcinlet and inletreset.
Warning the algorithm for this inlet boundary condition is only marginally stable as of Oct 2014. It is better to use fixed velocity, with a boundary sufficiently far upstream so that the changes in pressure do not modify the total pressure significantly if you can.

Needs: *idim4d, clt*, may also need *pg, rho, U1, U2, U3.* Sets: *flowinn, flowinprop.* Located in inletsubs.c.

inletreset Reset the velocity on the inlet boundary.

Input: none

The velocities are reset on the inlet boundary using the parameters in *flowinn* and *flowinprop* set using inletinit. Also, neighboring pressures on the outside of the flow region are set to match those on the inside.

Example: c: inletreset

Note: inletreset is ignored if *flowinn* does not exist. Otherwise –
Needs: *idim4d, xyz, xyzp, rho, flowinprop.* Uses if available: *itrange*, Modifies: *pp,U1,U2,U3.* Located in inletsubs.c

interp_gtom Interpolate variables from the regular grid to between the points.

Input: *n,*

namefrom[j], *nameto*[j] for j=0,,j<*n*

Interpolate *n* variables. *namefrom*[j], on-the-grid-points variables, to between-the-grid-points variables, *nameto*[j]. Set *n* <=20. Currently a simple 8-point average is used for the interpolation.

Example: c: interp_gtom 1 vlamg vlam

Needs*: idim4d,* each *namefrom*[j]. Sets: each *nameto*[j].

interp_gtop Interpolate variables from the regular grid to the pressure grid.
    Input: *n,*
        namefrom[j], *nameto*[j]   for j=0,,j<*n*
Interpolate *n* variables. *namefrom*[j], on-the-grid-points variables, to the pressure grid variables, *nameto*[j]. Set *n* <=20.

Example: c: interp_gtop 1 pg pp

Needs*: idim4d, cvdc, matchpc, cpsleep,* each *namefrom*[j]. Use if available: *itrange*. Sets: each *nameto*[j].

interp_ptod Interpolate from the pressure grid to the double grid (*xyzdouble*).
    Input: *n,*
        namefrom[j], *nameto*[j]   for j=0,,j<n
Interpolate *n* variables, *namefrom*[j], from the pressure grid, to *nameto*[j], variables on the *xyzdouble* grid points. Set *n* <=20.

Example: c: interp_ptod 1 pp pd

Needs: *idim4d,* each *namefrom*[j]*, xyzp, xyzdouble*. Use if available: *itrange*. Sets: each *nameto*[j].

interp_ptog Interpolate from the pressure grid to the regular grid.
    Input: *n,*
        namefrom[j], *nameto*[j]   for j=0,,j<*n*
Interpolate *n* variables. *namefrom*[j], from the pressure grid, to the on-the-grid-points variables, *nameto*[j]. Set *n* <=20.

Example: c: interp_ptog 2 pp pg ppadd pgadd

Needs: *idim4d,whoelse,* each *namefrom, xyz, xyzp*. Use if available: *itrange*. Sets: each *nameto*[j].

lineoutput Make a lineplot file with results along lines of grid points.
    Input: *filename,*
        name, until *name*="" (max 20 on-the-grid-points names)
        clindir
        if *clindir*=a,b,c or d, *a,b,c,d* (start location), *abcdend* (end location)
        if *clindir*=i,j,k, or n*, i,j,k,n* (start location), *ijknend* (end location)
        Repeat from read *clindir* until *clindir* is not a,b,c,d,i,j,k, or n

Example: c: lineoutput linenearwall angle U1 U2 pg ""

```
b -1.01 0 0 0 1  a -1 1.01 0 0  1  b 1.01 1 0 0 0 e
```

Put on file linenearwall or its alias*, values of *angle, U1, U2,* and *pg*, along the grid line for
a=–1.01, *c*=0*, d*=0, from *b*=0 through *b*=1*,* then along the grid line for
b=1.01, *c*=0*, d*=0, from *a*=–1 through *a*=1*,* then along the grid line for
a=1.01, *c*=0*, d*=0, from *b*=1 through *b*=0.
Corresponding results in file linenearwall have the form:

```
m4d lineoutput                          one line header
  6                                     number of variables
abcd idir angle U1 U2  pg               names of variables
0  1  0 0.0280755 0  0.0108516          values listed
…..
0  1  180 -0.137497 0 -2.13535
```

*An alias to the file name may be set by first using,
e.g., c: constant linenearwall s 1 linenwcase42
The results will then be saved in file linenwcase42 instead of the file named linenearwall.

Needs: *idim4d, abcd*. Uses each variable *name* if it exists. Note: it is assumed that the variables have the same dimensions as the grid, but this is **not** checked.

lineoutputijk Make a lineplot file for line output of dimensioned variables.
Input: *filename,*
name, until *name*="" (max 20 variables)
ndims
1charname[j], *id*[j],   for j=0,, j<*ndims*
is[j], *ie*[j],   for j=0,, j<*ndims*
Repeat reading *is* and *ie* until *is*[j]=0.
ndims is the number of dimensions, the same for all variables, *name*. *id*[j] is the size of each dimension. If *1charname* for the dimension is *i, j, k,* or *t, idim4d*[0, 1, 2, or 3] is added to *id* (if array *idim4d* exists) to obtain the dimension size. Put on file, *filename* or its alias*, the values of the variables *name* for each range *is*[j], *ie*[j]. Note the range is read Fortran style with first point having the index 1, and the last point *id*[j], and that *is*[j] and *ie*[j] are bounded by 1 and *id*[j]. The lines are formed by indexing all dimensions at once so that diagonal lines may put out. Smaller dimension indices stay at their last value, while larger ones complete their range.

Example: c: lineoutputijk linediag y qturb omturb ""
```
3 i 0 j 0 k 0    2 2 1 23 1 23    0
```

Puts results along the j,k grid diagonal on file linediag:
```
m4d lineoutputijk
6
  i j k
 y qturb omturb
 1 0 0 -1  0 0.0203589
 1 1 1 -0.999 0.000430687 0.0223082
 …
1 22 22 -1.84587e-05 0.920687
```

Note that while the indices are read Fortran style they are output C-style starting at 0.

Example: c: lineoutputijk linemid ym pkdk gbij ""
    3 i -1 j -1 k -1   1 1 100 1 22 22   0

Puts out results for the between-the-points variables (dimesions 1 less that the grid) for i=1, k=22, along the line j=min(100,*idim4d*[1]) down to j=1.

*An alias to the file name may be set by first using,
    e.g., c: constant linemid s 1 linemidcase23
The results will then be saved in file linemidcase23 instead of a file named linemid.

Uses, if available, *idim4d* and each variable *name*. Note: it is assumed that the variables have the specified dimensions, but this is **not** checked.

mirror Mirror variables across an a, b, or c boundary.
    Input: *append, todo, aborc, side,*
    namefrom, sign
    repeat last line until *namefrom* is not found or ""
This routine is useful if, e.g., you have results for a half channel and wish to create starting results for a full channel which include the mirror image.
append - the appendix to be added to *namefrom* for the new array.
Set *todo*='m'. (Other options, besides mirror, were planned but not written.)
Mirror across the *aborc*='a', 'b', or 'c' boundary on the first grid index side if *side*=0, otherwise on the last grid index side. The mirror image values are multiplied by -1 if *sign*=-1, otherwise they are copied. The *namefrom* arrays may be on-the-grid arrays, between the point arrays or pressure grid arrays, or multiples of these.

Example: c: mirror A m c 1 U1 1 U2 1 U3 -1 ""

mirrors U1, U2, U3 across the last k plane, giving arrays U1A, U2A, U3A. The mirror image of U1 and U2 are copies whereas U3mirror = -U3.

Needs: *idim4d*.

momcam Coefficients of δU_i for the abbreviated momentum equations.
    Input: *relax*
Calculate *cplus* and *cam* for the abbreviated momentum equation,

$$\left(cam[k]+cplus[k]\right)\delta U_i[n] = -\sum_{j=0}^{j<cflop\_n[k]} cpflop\_c[k][j+i*cflop\_n[k]]\delta p[cpflop\_i[k][j]]$$

where k=*wherefw*[n].

cplus is evaluated as the sum of the positive coefficients for the first set of coefficient in *coef_c*. *cam* are additional relaxation factors to help converge steady separated flow. They are evaluated as

$$cam = \max_{i=1,2,3}\left(\int_{mom.c.v} \rho \sum_{k=1}^{3}\sqrt{\left|\frac{\partial U_i}{\partial x_k}\right|\left|\frac{\partial U_k}{\partial x_i}\right|}dVol\right)$$

then relaxation is used with the previous values (if any) such that

$$cam = \max(cam_{new}, (1-relax)*cam_{new} + relax*cam_{old}) \;.$$

Example: c: momcam .5

Needs: *idim4d, wherefw, noindfwpts, match, wherep, coef_n, coef_c, U1, U2, U3, rho, xyz, cvdc, cvdcdouble, cltp*. Use if available: *itrange*. Sets: *cplus, cam.*

momcamddt Alternatives to *cam* for the abbreviated momentum equations.
Input: None
cama is an alternative to *cam* (momcam) based on the the 'safe' 1/dt given by ddtall (n=2)

$$cama = \int_{mom.c.v.} \rho \max_{i=1,2,3}\sqrt{\max\left(0, \sum_{k=1}^{3}\left(\frac{\partial U_i}{\partial x_k} + 2\varepsilon_{ink}\Omega_n\right)\left(\frac{\partial U_k}{\partial x_i} + 2\varepsilon_{kni}\Omega_n\right)\right)}$$

Another alternative (which has proved less useful),

$$camb = \int_{mom.c.v.} \rho \max_{i=1,2,3}\sqrt{\max\left(0, \left|\frac{\partial U_i}{\partial x_i}\right|, \sum_{k=1}^{3}\left(\frac{\partial U_i}{\partial x_k} + 2\varepsilon_{ink}\Omega_n\right)\left(\frac{\partial U_k}{\partial x_i} + 2\varepsilon_{kni}\Omega_n\right)\right)}$$

Copy *cama* or *camb* to *cam* to use one of these alternatives. momcamddt also calculates *cplus* the same as momcam.

Example: c: momcamddt

Needs: *idim4d, wherefw, noindfwpts, match, wherep, coef_n, coef_c, U1, U2, U3, rho, xyz, cvdc, cvdcdouble, cltp*. Use if available: *itrange, zrotation*. Sets: *cplus, cama, camb.*

momrhsr Evaluate the right hand side of the momentum equations.
Input: *oldnew, jcoef*
name, until *name*=""
Set the right hand side of the momentum equations.
If *oldnew* begins with 'n' initialize the r.h.s. as zero before adding the specified elements. (Otherwise *oldnew* must begin with 'o' for old).

$rhsUL[i] = 0,$ for each equation-point, *i*, and each velocity component, *L*=1,2,3.

Use coefficient set *jcoef,* for convection, etc., provided *jcoef*>=0,

$$rhsUL[i] \quad += \quad - \sum_{k=0}^{k<coef\_n[i]} coef\_c[i][k+jcoef*coef\_n[i]]*UL[coef\_i[i][k]]$$

where "+=" means that the contribution is added to *rhsUL*[i].
Additional contributions from pressure, Reynolds stresses, isotropic viscosity and rotation are specified by the *name*s read:
<u>If *name* begins with 'p'</u> and is dimensioned *noppts*, e.g. *pp*, the pressure term, $-\oint pdA_i$, evaluated as

$$rhsUL[i] \quad += \quad - \sum_{k=0}^{k<cpflop\_n[i]} cpflop\_c[i][k+L*coef\_n[i]]*pp[cpflop\_i[i][k]].$$

<u>If *name=dpdx*</u>, the pressure term, $-\oint pdA_i$, evaluated as

$$rhsU1[i] \quad += \quad - \sum_{area\ segments} dpdx[0]*x_{area\_center}dA_x.$$

<u>If *name=bij*</u>, and both *bij* and *qturb* exist the Reynolds stress terms, $-\oint \rho\overline{u_i u_L}dA_i$, are added. For each area segment simple averages of *rho, qturb* and the *bij*'s for the points associated with the control volumes on either side are used to evaluate the stresses. Border surfaces use the single value associated with the control volume.
<u>If *name* begins with 'v'</u> and has the same dimensions as an isotropic viscosity, e.g. *vcoakley*, and the arrays *U1ddxi, U2ddxi, U3ddxi* exist, the secondary viscous term, $\oint \mu\frac{\partial U_i}{\partial x_j}dA_i$ is added. Note, no addition is made for the primary viscous term, $\oint \mu\frac{\partial U_j}{\partial x_i}dA_i$, instead its effects are assumed to be included through the *jcoef* coefficient array. Note also that the viscous contribution on the grid perimiter have been omitted, consistent with repeating boundaries and the evaluation of the primary viscous coefficients.
<u>If *name=zrotation*</u>, Coriolis based on rotation about the z axis using volume averages are added,

$$rhsU1[i] \quad += \quad + \sum_{volume\ segments} 2*zrotation[0]*(rho*U2)_{average}dVol,$$

$$rhsU2[i] \quad += \quad - \sum_{volume\ segments} 2*zrotation[0]*(rho*U1)_{average}dVol.$$

If *zrotation* is dimensioned >=3, the centrifugal term, based about the point, $x_{axis} = zrotation[1]$, $y_{axis} = zrotation[2]$ is also added as a volume integral,

$$rhsU1[i] \quad += \quad + \sum_{volume\ segments} (zrotation[0])^2 * rho_{average}\left(x_{average} - x_{axis}\right)dVol,$$

$$rhsU2[i] \quad += \quad + \sum_{volume\ segments} (zrotation[0])^2 * rho_{average}\left(y_{average} - y_{axis}\right)dVol$$

<u>Example</u>: c: momrhsr new 0 pp bij zrotation ""

Needs: *idim4d, noindfwpts*. May need *coef_n, coef_i, coef_c, nocoefs, U1, U2, U3, U1ddxi, U2ddxi, U3ddxi, cpflop_n, cpflop_i, cpflop_c, rhsU1, rhsU2, rhsU3, wherep, wherefw, rho, csym, xyz, xyzdouble*. Use if available: *itrange*. Use if available and specified, *p..., v..., dpdx ,bij, qturb ,zrotation*. Sets or updates: *rhsU1, rhsU2, rhsU3*.

omrhscoakley Right hand side of the *omturb* equations using Coakley's model.
omrhsmarv Right hand side of the *omturb* equations using the MARV model.

Input: none

Set the right hand side of the *omturb* equations using coefficient set 0, for convection, etc. In general, for the production and dissipation source terms, properties are averaged over continuity control volumes, with the results distributed by volume to the surrounding control volumes. Exceptions to provide stability: when production is negative, and for one of the ω's in the dissipation term, ω on the grid point is used. Also ω_{mean} in the dissipation term is limited to the range $0.5\,\omega_{pt}$ to $2\ \omega_{pt}$.

$$rhsomturb[i] = -\sum_{k=0}^{k<coef\_n[i]} coef\_c[i][k + jcoef * coef\_n[i]] * omturb[coef\_i[i][k]]$$

$$+\int (c_{\varepsilon 1} - 1)\rho \frac{P^{+}k}{k}\omega_{mean}\, dVol + \int (c_{\varepsilon 1} - 1)\rho \frac{P^{-}k}{k}\omega_{pt}\, dVol - \int (c_{\varepsilon 2} - 1)\rho\omega_{mean}\omega_{pt}\, dVol$$

Also set relaxation coefficients, *caomturb*[*i*], based on the production and dissipation source terms.

Example c: omrhsmarv

Needs: *idim4d, noindfwpts, wherep, wherefw, qturb, omturb, pkdk, rho, vlam, xyz, cvdc, cvdcdouble, cltp*. omrhscoakley also needs: *walldist*. omrhsmarv also needs: *gbij*. Both use if available: *itrange*. Both set: *rhsomturb, caomturb*.

omwall	Set *omturb* at walls based on input parameters.
omwallcoakley	Set *omturb* at wall for a version of the Coakley model. (No input)
omwallmarv	Set *omturb* at walls using the MARV model. (No input)
omwallmarvs	Set *omturb* at walls using the MARVS model. (No input)

Input for omwall: *ca, cb, cc*

Omwallmarv uses the wall boundary condition, $\omega_{wall} = 0.089 dq/dy_{normal}$.
Omwallmarvs uses the wall boundary condition, $\omega_{wall} = 0.095 dq/dy_{normal}$.
Omwallcoakley uses the wall boundary condition, $\omega_{wall} = 0.56 dq/dy_{normal}$.
Omwall sets the wall boundary condition from

$$c_b\left(\sqrt{\frac{c_a}{\omega}\frac{dq}{dy}} - 1\right) + c_c\left(\frac{\sqrt{\nu}}{\omega^{3/2}}\frac{d\omega}{dy}\right) = 0$$

so that if *cc*=0, the boundary condition is $\omega_{wall} = c_a dq/dy_{normal}$. If *cb*=0, $d\omega/dy_{normal} = 0$ is approximated by $\omega_{wall} = \omega_{near\ wall}$.

Example c: omwallmarv

Example c: omwall .089 1 0   results same as c: omwallmarv

Needs: *idim4d, noindwpts, whoisw, wnear, itogether, whoelse, wnorm, qturb, xyz*. Omwall also needs *vlamg*. Use if available: *itrange*. Modifies: *omturb*. All located in omwallsubs.c.

ppreset Reset the pressure for new pressure point locations.
        Input: none
Interpolate *pp* from the old pressure grid, *xyzpold*, to the new pressure grid, *xyzp*.

Example c: ppreset

Needs: *xyzpold, xyzp, idim4d, matchpc, cpsleep*. Use if available: *itrange*. Modifies: *pp*.

ppts2eqppts Make an p-equation point array from a p-point array.
        Input: *namevp, namepeq*
Sets *namepeq*[k]=*namevp*[*whoisp*[k]],   k=0,*noindppts*[0].

Example: c: ppts2eqppts xtraflow rhscadd

Needs: *namevp, whoisp, noindppts*. Sets: *namepeq*.

prinstress Calculate principle component vectors for Reynolds stresses.
        Input: none
If arrays *bij* and *qturb* exist, set the array *Repr* which can be used to plot Reynolds stresses in the (JGM) program x11pictw. If arrays *U1ddxi, U2ddxi,* and *U3ddxi* exist (see command gradprop) also set the array *Stpr*, the principle component vectors for the strain rate tensor, and *VVex*, the relative vorticity vector.

Example: c: prinstress

Needs: *idim4d*. Use if available: *bij, qturb, U1ddxi, U2ddxi, U3ddxi*. Set if needed arrays exist, *Repr, Stpr, VVec.*

print Print regular arrays.
        Input: *name, iformat*
Print array *name*. If *iformat*: =–1, between the points array; 0, on the points array; 1, between and outside the points in space, on the points in time; 2, double the points minus 1 in space, on the points in time; >2 non-standard array, print in groups of *iformat*.

Example: c: print U1 0

Needs: *idim4d* for *iformat*= -1,0,1,2.

printcontrol Turn off/on standard printing.
        Input: *action* If action ='filenew' or 'fileold' read *filename*

M4d, in general, produces lots of printout which initially goes to standard output. It gives each command and its input as executed. This may be turned off with
Example: c: printcontrol off
and turned on again with
Example: c: printcontrol on
Print may also be redirected to a new named file using
Example: c: printcontrol filenew printfilenew
or appended to an old file using
Example: c: printcontrol fileold printfileold
If printfilenew or printfileold = "stdout", print is returned to standard output. Note, this only affects 'normal' printing. Errors, warnings, and requested printout, such as from command print, are always printed.

printcontrol is not a separate routine but part of m4d.c.

prints Print parts of an array.
Input: *name*, *ndims*
1charname[j], *id*[j], *is*[j], *ie*[j] for j=0,,j<*ndims*
If the *1charname* for the dimension is *i, j, k,* or *t, idim4d*[0, 1, 2, or 3] is added to *id* (if array *idim4d* exists) to obtain the dimension size. Print the range *is* to *ie* with *is*=1 for the first value. (Note: The print will show is=0 at the first point.)

Examples: c: prints U1 4 i 0 1 5 j 0 1 5 k 0 1 1 t 0 1 1
c: prints pp 4 i 1 1 5 j 1 1 5 k 1 1 5 1 t 0 1 1

Uses if available: *idim4d*.

printscoef Print parts of a coefficient array.
Input: *namen, namei, namec, nypc, iexi, jrep*
is[j],*ie*[j]. for j=0,,j<4
Names of the parts of coefficient arrays: *namen,* the number of coefficients, *namei,* the indices, and *namec,* the values. *nypc*=0 if the arrays pertain to grid points, =1 if for p-points. *iexi*=0 if the coefficient indices are associated with grid points, =1 for p-points. *jrep* is the number of coefficient sets in *namec*. Print the range *is* to *ie* with *is*=1 for the first value. If *nypc=iexi* the expanded coefficient indices are shown relative to the index associated with the coefficient set.

Example: c: printscoef coef_n coef_i coef_c 0 0 3 1 5 1 5 1 1 1 1

Needs: *idim4d*. If *nypc*=1, needs *wherep*, else needs *wherefw, match.*

qbijles Arrays for a non-standard **(**trial) Reynolds stress based LES.
Input: none
The use of adaptive control volumes opens the door for non-Boussinesq sub grid scale models for LES. This routine provides some between-the-points arrays which are being tested.

Length scale *Lddx2*= $\sqrt{3/(1/\delta x^2+1/\delta y^2+1/\delta z^2)}$ (Cartesian)
The strain-rate *srate* = $\sqrt{2S_{ij}S_{ij}}$ where $S_{ij}=0.5(\partial U_i/\partial x_j+\partial U_j/\partial x_i)$
Anisotropy, b_{ij}, *bijm*= $(\partial U_i/\partial t)(\partial U_j/\partial t)/(2k_t)-\delta_{ij}/3$ where $2k_t=(\partial U_n/\partial t)(\partial U_n/\partial t)$
For production *sijbij* = $S_{ij}b_{ij}$

<u>Example</u> c: qbijles

Needs: *idim4d, wherep, U1, U2, U3, dU1dt, dU2dt, dU3dt, cltp, xyz.* Use if available: *itrange.* Set: *Lddx2, srate, bijm, sijbij.*

qturbrhs Right hand side of the *qturb* equations.
<u>Input</u>: none
Set the right hand side of the *qturb* equations using coefficient set 0, for convection, etc. In general, for the production and dissipation source terms, properties are averaged over continuity control volumes, with the results distributed by volume to the surrounding control volumes. Exceptions to provide stability: when production is negative, and for the dissipation term, *q* on the grid point is used.

$$rhsqturb[i]=-\sum_{k=0}^{k<coef\_n[i]} coef\_c[i][k+jcoef*coef\_n[i]]*qturb[coef\_i[i][k]]$$

$$+\int\frac{\rho}{2}\frac{P^+k}{k}q_{mean}\,dVol+\int\frac{\rho}{2}\frac{P^-k}{k}q_{pt}\,dVol-\int\frac{\rho}{2}\omega_{mean}q_{pt}\,dVol$$

Also set relaxation coefficients, *caqturb*[*i*], based on the production and dissipation source terms.

<u>Example</u> c: qturbrhs

Needs: *idim4d, noindfwpts, wherep, wherefw, qturb, omturb, rho, pkdk, cltp, xyz.* Use if available: *itrange.* Set: *rhsqturb, caqturb.*

rayleigh Calculate Rayleigh instability info for Reynolds stress model addition.
<u>Input</u>: *as, ae, bs, be, cs, ce, iline, cyw, cyi, cypd, cely*
General description: Search for regions where a Helmholtz instability might exist by looking for locations of maximum vorticity, but filtered so near wall regions are omitted. Determine a width for the unstable region and set *ray* =1 in the unstable region. Also set a mixing length *elps* (*L*), and a mixing length viscosity, *vtpd* (L^2"dU/dy") over the region pressure-diffusion is estimated to be important.

In particular: Search mid-volumes over lines of constant i, j, or k for *iline* =1, 2, or 3, respectively. Limit the search to the region of the grid with 1-d grid parameters a=*as* to *ae*, b=*bs* to *be*, c=*cs* to *ce*. Along each line determine the maximum and minimun magnitudes of the velocity, $|U|_{\max}$, $|U|_{\min}$. Then, along each line, look for points of maximum vorticity, W (array *wrate*). Note the maximum value, $W_{\max}$, and its distance from the neareat wall, $y_{W_{\max}}$. Set the width of the unstable region as $L_I=(|U|_{\max}-|U|_{\min})/W_{\max}$. Ignore if too close to the wall,

$y_{W_{\max}} < cyw\, L_I$. Otherwise set *ray*=1 for the region $\left|y_{W_{\max}} - y\right| < cyi\, L_I$. For the region over which pressure-diffusion is to be added, $\left|y_{W_{\max}} - y\right| < cypd\, L_I$, set the mixing length, *elps* = $\min(L_I, cely * y)$ and the viscosity , *vtpd* = *elps*elps** $W_{\max}$.

This test version has only been applied to the 2-d T3L transition test cases in a region where the grid is orthononal and it is assumed that you are searching lines normal to the wall and that is the appropriate direction. Should it prove generally useful a more sophisticated search procedure will be needed.

Example: c: rayleigh 0 10 1 3 0 1   2   cyw cyi cypd cely

Needs: *idim4d, abcd, wherep, cltp, U1, U2, U3, wrate, walldist.* Set: *ray, elpd, vtpd.*

rmsminmax Determine the r.m.s, min and max for arrays.

Input: *name*, until name=""

Determine the pointwise root-mean-square-value of each of the arrays *name*. Print this together with the minimum and maximum values and their location in the array. *ITER* and/or *TIME* are also included in the one-line print (if they exist) so that the print results can be used as a convergence history. (Use, for example, the Unix command “grep RMSMM outputfilename > rmsmmfile” to save just these lines on a separate file.) The array *rmsmm* is also created, saving the r.m.s, and the value of largest magnitude for each of the first 10 *name*s listed.

Example: c: rmsminmax U1 U2 U2 dU1 dU2 dU3 ""

Needs: *idim4d, name.* Use if available: *ITER, TIME.* Set: *rmsmm.*

set_contar Determine aspect ratios of continuity control volumes.

Input: none

The aspect ratio array, *contar*, is set in each direction for each independent pressure grid point. For non-zero continuity control volumes it is the one over the normal length in each grid direction times the minimum of the 3 lengths. E.g. if the grid is Cartesian with spacing $\delta x, \delta y, \delta z$=10,1,2, the *contar* values for that volume are 0.1, 1, 0.5.

Example: c: set_contar

Note calling set_contar is unnecessary (provided the grid points are not moved) since set_cpda, which uses *contar,* will automatically call this routine if *contar* does not exist. Located in c_set_cpda.c.

Needs: *idim4d, noindppts, xyz, whoisp*. Sets: *contar*.

set_cpda Determine coefficients for pressure in the momentum equations.

Input: *cenfac0, ar1, cenfac1, ar2, cenfac2, nomit*

omit[i] for i=0,,,i<*nomit*

Determine the coefficients of *pp* for the pressure term in the momentum equations, $\int \bar{P} dA_i$. Omit for grid points with types (1 character names) *omit*[i]. The array *cpda*

is set giving coefficients for the 8 surrounding pressure points in each of the 3 (dA_i) direction. The input parameters, *cenfac0, ar1, cenfac1, ar2, cenfac2,* determine whether approximate linear interpolation (*cenfac*=0) or centered values (*cenfac*=1) are used for the pressure on each sub-surface. The parameters allow for this to be a function of continuity-control-volume aspect ratio, *contar* which is defined for each continuity control volume in each grid direction (i, j or k) as one over normal length of the control volume times the minimum of the 3 lengths. (Long directions have small values.) *cenfac0* is used for aspect ratio(AR)=0, *cenfac1* for 0<AR<=*ar1*, *cenfac2* for AR>=*ar2*. Set 0<=*contfac*<=1, *ar2*>*ar1*.
The companion command set_cpflop, collects these *cpda* pieces together for use in the momentum equations, resolving matching grid points and sleeping pressure points.

Example: c: set_cpda .0 .05 .5 .15 0 1 i

Uses linear interpolation in directions with AR=0 and greater than 0.15 (~1/7), and the average of linear and centered for 0<AR<.05 (=1/20). Experience suggests that *cenfac* =0.5 is adequate to provide stability when long thin control volumes are used and prevents up/down wobbles in the normal velocity component near walls without compromising the solution. Values of *cenfac* greater than 0.5 are not recommended.

Needs arrays: *idim4d, matchpc, xyzdouble, xyzp*. Use if available: *itrange, roundoff.* Set: *cpda.* If array *contar* does not exist, c_set_contar is used to set it.

set_cpflop Collect pressure coefficients for each independent momentum equation.
Input: none
The coefficient arrays *cpflop* represent $\int \overline{P} dA_L$ for momentum control volumes. *cpflop* is formed using *cpda*. Matching points and sleeping pressure points are resolved so that there is a set of coefficients for each independent velocity flow point and the coefficients are for independent pressure points.

$$\left(\int \overline{P} dA_L\right)_{c.v.i} = \sum_{j=0}^{j<cpflop\_n[i]} cpflop\_c[i][j+nL] * pp[cpflop\_i[i][j]]$$

Example: c: set_cpflop

Needs *idim4d, noindfpts, wheref, cpda, matchpc, cpsleep*. Use if available: *itrange, roundoff*. Sets *cpflop_n, cpflop_i, cpflop_c.*

set_cpsleep Set 2-point coefficients to interpolate for sleeping pressure points.
Input: none

$$pp[i]_{\text{sleeping point}} = cpsleep[i] * pp[matchpc[i]] + cpsleep[i+noppts[0]] * pp[matchpc[i+noppts[0]]] \ .$$

Example: c: set_cpsleep

Needs: *idim4d, csym, matchpc, xyzp*. Sets: *cpsleep.*

set_gbij Calculate turbulence anisotropy parameter.

Input: None
Set the turbulence anisotropy parameter, $g = 1 - 4.5 b_{ij} b_{ji} + 9 b_{ij} b_{jk} b_{ki}$ using between the points values of b_{ij}.

Example: c: set_gbij

Needs: *idim4d, wherep, bij*. Sets: *gbij.*

set_pkdk Calculate turbulence production rate for continuity control volumes.
Input: *name*
The turbulence production rate divided by the turbulence kinetic energy set using array *name*. Either $P/k = -2 b_{ij} S_{ij}$ when *name* is *bij* or $P/k = 2\mu_t S_{ij} S_{ij} / (\rho q^2)$ when *name* is a turbulent viscosity. Which to use is based on the size of the array *name.*

Example: c: set_pkdk vcoakley

Needs: *idim4d, wherep, cltp, qturb,rho, U1, U2, U3, name*. Set: *pkdk.*

set_srate Calculate the mean velocity strain rate.
Input: None
Set *srate* = $\sqrt{2 S_{ij} S_{ij}}$ where $S_{ij} = 0.5(\partial U_i / \partial x_j + \partial U_j / \partial x_i)$.
Note: *srate* is also set by commands qbijles and viscles, and is a component of *ddtall* set by command ddtall.

Example: c: set_srate

Needs: *idim4d, wherep, cltp, U1, U2,* U3. Uses if available *itrange*. Sets: *srate.*

set_volcont Determine the volume of each continuity control volume.
Input: none

Example: c: set_volcont

Needs: *idim4d, xyz*. Set: *volcont.*

set_volmom Determine the volume of each momentum control volume.
Input: none

Example: c: set_volmom

Needs: *idim4d, noindfwpts, matchpc, wherefw, cltp, xyz, cvdc, cvdcdouble*. Use if available *xyzdouble*. Set: *volmom.*

set_wherefw Determine independent points which may need equations.
Input: *ninout*,
omit[j], for j=0,,j<$|ninout|$
Include, *ninout*>0, or exclude, *ninout*<0, specific point types, *omit*[j], where *omit*[j] are the 1 character point types for the grid in the array *clt*. Determine the number of independent points of the specified point types, *noindfwpts,* the equation number

for each point, *wherefw* and the primary point associated with each equation *whoisfw*.

Example: c: set_wherefw –1 s Omit points of type s

Needs: *idim4d, match, whoelse, clt*. Sets: *noindfwpts, wherefw, whoisfw*.

set_wherep Set information pertaining to pressure point equations.
Input: none
Determine the number of independent pressure points, *noindppts*[0]. Set *wherep*, the index of the p-point in the independent pressure points list. Set *wherep* = –1 if the p-point is sleeping. Set *whoisp* the index of the point in the *wherep* list. Also set a p-point type array, *cltp*, so the *cltp*=f for flow points, *cltp*=F for flow points next to a wall, *cltp*=s for internal solid points, and *cltp*=S for internal solid points which are next to a wall.

Example: c: set_wherep

Needs: *idim4d, matchpc, clt*. Sets: *noindppts, wherep, whoisp, cltp*.

set_xyzdouble Determine the double size grid (nominal half spacing)
Input: none
Determine the partitioned grid, *xyzdouble,* based on the control volume divide arrays, and then the pressure grid, *xyzp*.

Example: c: set_xyzdouble

Needs: *idim4d, xyz, cvdc, cvdcdouble*. Use if available: *itrange*. Sets *xyzdouble, xyzp*.

valueat Find the value at the point a,b,c in an on, mid, or p-point array.
Input: *nameout, namearray, a, b, c*
Creates *nameout* a double array of dimension 1. Set *nameout*[0]=*namearray* interpolated to *a,b,c* in array *abcd*. For mid or p-points arrays, interpolation assumes physical space interpolates linearly with *a, b,* and *c*. For p-points arrays *cvdc* is used if available, but the interpolation does not consider transverse variations.

Example: c: valueat ptredge pp 1. .5 0.

Needs: *idim4d, abcd, namearray*. Uses if available: *cvdc*. Sets: *nameout*.

varinit Initialize on the points variables.
Input: *name, iformat*
If *iformat*=0: *cf,cw,cs*
If *iformat*=1: *fac, na*[j], for j=0,,j<4
((*aa*[j][k], k=0,,k<*na*[j]), j=0,,j<4)
((*v*[j][k], k=0,,k<*na*[j]), j=0,,j<4)
If *iformat*=2 or 3: read from file *name* variables in arraydump format

For *iformat* = 0 create the on-the points array, *name* and initialize the values with *cw* for wall points, type w; *cs* for internal solid points, type s; and with *cf* for all other points.

For *iformat*=1, initialize with a product profile. The *abcd* values for the grid points are located (non-integer index) in the arrays *aa*[0], *aa*[1], *aa*[2], *aa*[3], respectively. The corresponding values for the factors using linear variation between the *v* values are then multiplied together to obtain the point value,

name[i,j,k,n] = *fac**(*v* of *a*[i])*(*v* of *b*[j])*(*v* of *c*[k])*(*v* of *d*[n]) .

Note, if *aa* does not cover the entire grid, and the array did not previously exist, the end values specified are used to fill the grid.

Examples: c: varinit U1 0 5.2 0 0

```
c: varinit test 1 3. 4 2 1 1
      –10. 0 1. 100.    0 15.   0   0
        1.   1. 2. 2.       1. 2.   1.  1.
```

Iformat = 2 or 3. Retrieve and interpolate to current grid variables in file *name.* File *name* must include dumped array *idim4d* followed by *abcd* followed by on-the-points arrays of type double. Other arrays of type double or integer may be included in file *name*, but they will be ignored. The on-the-points arrays may include multiple variables such as *bij*. If *iformat*=2, only that part of the array covered by the read values *abcd* will be filled. If *iformat*=3, or the array did not previously exist, the entire array will be filled.

Example: c: arraydump saveme TIME idim4d abcd U1 U2 U3 pp pg ""

c: varinit saveme 3

Having dumped the variables to file saveme from one calculation, they be read in again interpolating them to the current grid for the next calculation. Note that *TIME* is ignored because it appears before *idim4d* and *abcd*, and that *pp* is ignored because it is not an on-the-point array.

Needs: *idim4d, clt, abcd*. *Iformat*=0 or 1, sets *name*. *Iformat*=2 or 3 sets or modifies on-the-points variables in file *name.*

varmatch Set matching points of on-the-points variables.

Input: *name* until *name*=""

Names which do not exist or are not on-the-point variables are ignored.

Example: c: varmatch U1 U2 ""

Needs: *nogpts, match*. Modifies each *name.*

varupdate Update a variable with its change using a 'safe' method.

Input: *name, change, method*

Update *name* with *change* using method *method. Name* and *change* must be on-the-points variables. Only the first letter of *method* is used. The default update is

name[j] = *name*[j]+*change*[j].

Methods log and inverse prevent negative changes from giving results <0.
If *change*[j]<0: *method* = "log", *name*[j]=*name*[j] exp(*change*[j]/*name*[j])
method= "inverse", *name*[j]=*name*[j] /(1–*change*[j]/*name*[j])
The *method* "b" is for turbulence anisotropy, after adding the change, realizability is used to limit the values.
An analysis array *updatew* is set. With respect to a reduction factor F, i.e. *name*[j] = *name*[j]+F**change*[j], the first value is F_{min}, the second the number of points with F<0.5 and the last the number of points with F<0.001. (For method "b" F represents an average factor.)

Examples: c: varupdate qturb dq log
c: varupdate omturb dom inverse
c: varupdate bij dbij b

Needs: *idim4d, name, change*. Modifies *name*. Sets *updatew*.

vint2eqpts Integrate between points array over momentum control volumes.
Input: *name, namevint, oldnew*
The purpose of this routine is to allow source term additions to equations based on the between-the-points array, *name*. The equation points array, *namevint* is ititiallized as zero if *oldnew* ='n', or it did not exist. Then the update is made,

$$namevint\mathrel{+}= \sum_{mom.c.v.sub-volumes} name\ dVol$$

Example: c: vint2eqpts hksourcem hksourcee n

Needs: *idim4dm noindfwpts, wherep, wherefw, clt, name*. Use if available *itrange*. Modifiies or sets *namevint*.

visccoakley Turbulent viscosity for the Coakley 2-eq. model.
Input: none
Set *vcoakley* between the points values of the turbulent viscosity as the smaller of the Coakley model, $0.09\rho\frac{q^2}{\omega}\left(1-\exp(-\frac{.02qy_{wall}}{\nu})\right)$, and a realizability limit, $\rho\frac{q^2}{1.732S}$.

Example: c: visccoakley

Needs *idim4d, wherep, cltp, rho, qturb, omturb, U1, U2, U2, walldist, xyz*. Use if available: *itrange*. Sets *vcoakley*.

viscles Set a mixing length style LES turbulent viscosity.
Input: *namecl*
Set the length scale as the smaller of the between the points array, *namecl*, and a VanDriest correction,

$$Lles[\text{mmid}] = L = \min(\ namecl[\text{mmid}],\ 0.41y\left(1-\exp\left(\frac{-y\sqrt{(\nu_{lam}+\nu_{LES})S}}{26\nu_{lam}}\right)\right).$$

Set *vles* between the points values of the turbulent viscosity. $\nu_{LES} = \rho L^2 S$. (The strain rate, *srate*=*S* is also set.) After ν_{LES} is set a mock wall function resets ν_{LES} between the wall and near wall points, using $\nu_{LES} = \sqrt{(\nu_{lam} + \nu_{LES})\nu_{lam}} - \nu_{lam}$. This allows the use of liner profiles between the wall and near wall points beyond the laminar sublayer.

Note that if *namecl* is set to 0.08δ, the model reverts to an ordinary mixing length model.

Example: c: viscles cdyles

Needs *idim4d, wherep, cltp, rho, U1, U2, U2, vlam,walldist, xyz, namecl*. Use if available: *itrange*. Sets *Lles, vles, srate*.

viscmarv Turbulent viscosity for *bij* and *qturb* for the MARV model.
Input: none

Sets *vmarv* between the points values of $0.735\rho\frac{q}{\omega}\sqrt{\overline{u_i u_j}}$. The anisotropic turbulent viscosity has 6 independent components, $\mu_{t11}, \mu_{t22}, \mu_{t33}, \mu_{t12}, \mu_{t13}, \mu_{t23}$.

Example: c: viscmarv

Needs *idim4d, wherep, rho, qturb, omturb, bij, xyz*. Use if available: *itrange*. Sets *vmarv*.

viscmarvheat Isotropic turbulent viscosity for heat and mass, MARV compatible.
Input: none
Sets *vmarvheat* between the points values of
$0.1\rho\frac{q^2}{\omega}\sqrt{\left(1-\exp(-0.00018(yq/\nu)^2)\right)\left(1-\exp(-0.00018(yq/\nu)^2 P_r{}^{0.7})\right)}$ where P_r is the laminar Prandtl number, *prlam*[0].

Example: c: viscmarvheat

Needs *idim4d, wherep, cltp, rho, qturb, omturb, prlam, vlam, walldist, xyz*. Use if available: *itrange*. Sets *vmarvheat*.

viscqnoise A turbulent viscosity model for noise generated turbulence.
Input: none

Sets *vqnoise* between the points values of $\min\left(\frac{0.09}{1000}\rho q_{noise}\frac{q^3}{\varepsilon}, \rho\frac{(q_{noise})^2}{1.732S}\right)$ where q_{noise} is *qnoise*[0].

Example: c: viscqnoise

Needs *idim4d, wherep, cltp, rho, qturb, omturb, vlam, qnoise, U1, U2, U3, xyz*. Use if available: *itrange*. Sets *vqnoise*.

w2wlineflat Between the points wall to wall distance.

Input: none

A starter routine for considering corner wall reflection. This routine is for flat y,z walls only at j and k grid limits. Calculate the smallest wall-to-wall vector that passes through the midpoint of each continuity control volume. For each independent p-point set:
w2wline = this wall-to-wall vector. *w2wdist* = magnitude of the vector. *w2wnorn* = the vector normal to *w2wline* in the *y,z* plane.
If the model should be successful, a geometrically more general routine will be needed.

Example: c: w2wlineflat

Needs: *idim4d, whoisp, noindppts, csym, xyz, clt, cltp*. Sets: *w2wline,w2wdist, w2wnorm.*

walldist Distance to nearest wall from center of each continuity control volume.

Input: none

Determine the distance to the nearest wall from the mid-point of each valid continuity control volume. Dimensioned *noindppts*[0], *walldist* is negative for non-valid cont. c.v.'s Also determine the unit wall normal vector pointing to the mid-point, *walln2m*.

Example: c: walldist

Needs: *idim4d, whoisp, noindppts, csym, xyz, clt, cltp*. Sets: *walldist, walln2m.*

wallflux Add specified wall flux/area to the r.h.s. of an equation.

Input: *namerhs, namefluxda*

Used, e.g., to add a uniform heat flux boundary condition to an energy equation. Formally, adds to the right hand side of the equation, *namerhs,* for each wall grid point, the control volume wall area times the flux per area, *namefluxda*. Currently, a uniform flux/area is used (the first value in *namefluxda*). However, this is read as a name for forward compatibility, so the routine may be later upgraded to use a full array and variable flux per area.

Example: c: wallflux rhstt fluxda

Needs: *idim4d, noindwpts, whoisw, whoelse, wherefw, clt, xyzdouble*. Modifies: *namerhsin.*

wallnorm Determine wall normal vectors and near wall points.

Input: none

For each wall grid point, determines the wall normal vector, *wnorm* and the grid index of the closest (in grid space) near wall point, *wnear*. If available, the wall normal is determined from a cross of circular arcs put through the wall point and neighboring wall grid points.

Example: c: wallnorm

Needs: *idim4d, match, csym, clt, xyz*. Sets: *noindwpts, whoisw, wnorm, wnear*.

Description of M4d Plot Package Commands

abcmask Set specified surfaces to a specified value in an on-the-points array.

Input: *name, value, aborc,*

abcvalue until abcvalue less than previous one

Set points in array *name* to *value* for specified a, b, or c locations, *abcvalue*….
name - an on-the-points array of type double. *aborc* = 'a', 'b', or 'c'.
The purpose of this routine is to enable the setting of a mask to, for example, select velocity vectors to be plotted by pc_picture.

Example:
```
c: varinit maskc 0 0 0 0
c: abcmask maskc 1. c 0 .01 .02 .03 .04 .05 .07 .1 .15 .2 .25 .3 .35 .4 .45 .5 0
c: algebra U1mc maskc U1 "" 1. 0. 1.
```
Initialize *maskc* as 0.
Set *maskc* to 1 for grid planes c=0, .01,,,,,,,, .5.
Set *U1mc=maskc*U1* so that *U1mc* is 0, except on the selected c-planes.

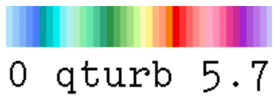
Needs: *idim4d, abcd, name*. Modifies: *name*.

bar Create a labeled colorbar image.

Input: *imagename, ipix, jpix, label*

imagename - the name for the pixel image (integer array).
ipix, jpix - the minimum horizontal, vertical size of the image in pixels. If necessary, the size will be increased to accommodate the label.
label - either 'b' or 't' for the label to be below or on top of the horizontal colorbar.
Command bar uses *p.fnames* and *p.fparms* to determine the bar colors and labels. For the image to be labeled, it will also require the file font.list and the corresponding (my jgm format) font maps.

Example:
```
c: constant p.defaultdir s 1 /Users/Shared/a/jgm
c: constant p.fnames s 3 qturb fi yout
c: constant p.fparms d 6 0. 0. 5.7 11 -1 50
c: bar imagebar 50 50 b
```
Set location of file font.list and font map files.
Set p.fnames and p.fparms.
Create imagebar.
Results at right after also using
```
c: image outgif imagebar  qbar   (to give qbar.gif)
```



Needs: *p.fnames, p.fparms*. Sets: *imagename*. Text will not be added unless the program can find the file font.list (and the files listed in file.list) in the current directory, or in the directory specified by *p.defaultdir*.

giflist Write to a file, a numbered list of .gif names.

Input: *prename, is, ie*

Write to the file *prename*.list, a list of .gif names prefixed with *prename* and the numbers, *is* thru *ie*.

Example: c: giflist qte 1 100
Will write to the file qte.list : qte1.gif qte2.gif …….. thru qte100.gif (each on a separate line). This file may be used as input to an external program which turns the .gif files into a video.

gridinfo Create information files for the double grid.
Input: none
Set arrays *idim4ddouble, abcddouble* and *cltdouble* to go with *xyzdouble* so that the double-grid may be plotted. Between the grid points values for *abcd* are simple averages. For *cltdouble*, between the points values are consistent with *clt,* but then the grid points themselves are set to 'x', so that on-the-grid surfaces will show the outlines of the control volumes.

Example: c: gridinfo

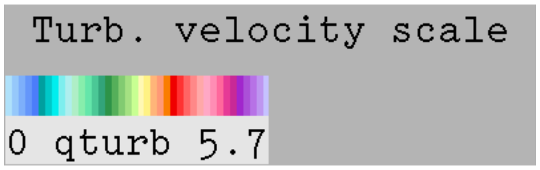
Needs: *idim4d, abcd, clt*. Sets: *idim4ddouble, abcddouble, cltdouble.*

image Create, combine, label, recolor, and/or output pixel image arrays.
Input: *action* then input needed for that action,
Repeat reading *action* and the needed input until *action*=="end"
Actions are : new, combine, label, recolor, outgif, and outjgm.
new Input: *imagename, ix, iy.* Create image *imagename* with pixel width *ix,* and height *iy*, and initialize the image with color 0. Sets: *imagename*.
combine Input: *imagebase, imageadd, ixll,iyll.* Add *imageadd* to *imagebase* with the lower left hand corner of *imageadd* placed at *ixll* pixels to the right and *iyll* pixels up from the lower left hand corner of *imagebase*. 'Add' means that color number 0 in *imageadd* is taken as transparent. Needs: *imagebase, imageadd.* Modifies: *imagebase*.
label Input: *imagename, text, ixc, iyc*. Add *text* to *imagename* centered at *ixc, iyc* (measured from the left hand cower corner) in *imagename.* If the text contains white-space, enclose in " ". Needs and modifies *imagename*. Text will not be added unless the program can find the file font.list (and the files listed in file.list) in the current directory, or in the directory specified by *p.defaultdir*.
recolor Input: *imagename, icfrom, icto*. Change color *icfrom* in *imagename* to color *icto*. Needs and modifies *imagename.*
outgif Input: *imagename, prename.* Output *imagename* as a .gif file to *prename*.gif. Note, *prename* is read as a file name so that aliasing and imbedded integers may be used. The '.gif' is added by the program. Needs *imagename* and the file color.map which may be either in the current directory or in the directory specified by *p.defaultdir*.
outjgm Input: *imagename, filename.* Output *imagename* in a jgm format to file *filename*. Needs *imagename.*

Example: c: image recolor imagebar 0 6

```
  new  imageL 400 120
recolor imageL 0 7
label imageL "Turb. velocity scale"  200 100
combine imageL imagebar  1 1
outgif imageL labelbar2
end
```

Recolor background of imagebar (created using command bar above).
Create new image, imageL, of width 400 and height 120.
Recolor background of ImageL.
Add label to image L.
Add imagebar to imageL.
Output imageL as a .gif.
Results at right.



iplaneint Create a lineplot file with area and mass averages.

<u>Input</u>: *filename, densityname,*
propname until *propname* = "". (Max of 20 prop names.)

Create file, *filename*, for command lineplot with area and mass averages of on-the-points properties on each i plane. The names for the grid and velocity vectors are obtained from *p.gnames* and *p.vnames*. Also create array *aveiflow*[(No. of prop names +1)*(No. of planes)] containing the flow-rate, then mass-average of each prop, for each i-plane.

<u>Example</u>: c: constant p.gnames s 4 idim4d xyz abcd clt
c: constant p.vnames s 3 U1 U2 U3
c: iplaneint lineipl rho cpt qqt ""

The resulting file, lineipl, contains:

```
iplaneint
14
 i a area flowp flowm flow
 cpta cptfp cptfm cptf
 qqta qqtfp qqtfm qqtf
```

plus values for these for each i-plane.

Needs: *p.gnames, p.vnames* and the arrays listed in these, *densityname*, and each *propname*. Sets: *aveiflow*. Writes file: *filename*.

keyword Analyize an arraydump file to give a lineplot file.

<u>Input</u>: *filedumpin, filelineout, nokey,*
For j=0,,,,j<*nokey* read:
keyin[j], *nokeyout*[j],
keyout[j][k] for k=0,,,,k< *nokeyout*[j]

Used, for example, to gather for plotting, information about convergence of a steady flow calculation or properties of a time dependent calculation.

<u>Example</u>: c: constant TIME d 1 0.
c: constant data d 10 0 0 0 0 0 0 0 0 0 0

```
        c: arraydump converg TIME data ""
        c: constant TIME d 1 1.
        c: constant data d 10 1 2 3 4 5 6 7 8 9 10
        c: arraydumpmore converg TIME data ""
        c: keyword converg convline 2
                TIME 1 t data 4 A1 B2 C3 D4
```

The resulting file, convline, contains:

```
 keyword from converg
 5
 t A1 B2 C3 D4
 0 0 0 0 0

 1 1 2 3 4
```

Keyword looks in the file converg for "TIME", the first value is the value for t. It then looks for "data" and takes the first four values as A1, B2, C3, D4. This is repeated until the end of the file is found.

Needs file *filedumpin*. Writes file *filelineout*.

lineplot Create a lineplot pixel image

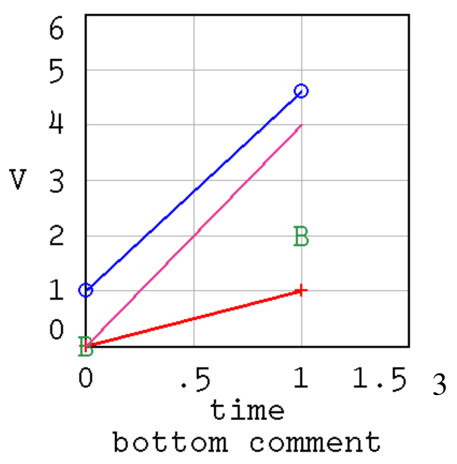
Input: *imagename, lineplotfile, comment, x1, x2, y1, y2, icbox, icgrid, labelx, axmin, axmax, dax, labely, aymin, aymax, day, namex, facx, addx, namey, facy, addy, icolor, linew, symbol,*
Repeat last line until *namex* or *namey* equals "end"

Create *imagename* of pixel width *x2*+20 and height *y2*+20. The lineplot box extending from *x1* to *x2* pixels and *y1* to *y2* pixels is drawn in color *icbox*. The x-axis is labeled *labelx*, and the variable values are *axmin* at *x1*, to *axmax* to *x2*. If *dax*=0, the x-axis is taken as a log scale and *axmin* and *axmax* should be factors of 10 (e.g. 0.1 100). Otherwise *axmin* and *axmax* should be even multiples of *dax*. If *icgrid*>0 grid lines are drawn each *dax* (or each factor of 10 for a log plot). If *icgrid*=0 tick marks are used instead. Similarly for the y-axis. The text *comment* is written under the x-axis label. Put in "" if it includes imbedded white-space.

The last line of input refers to the variables in file *lineplotfile*. Plot *namex*facx+addx*, versus *namey*facy+addy* in color *icolor* with linewidth *linew* pixels, and putting *symbol* at each data point in the file.

Example: c: lineplot imageconv convline "bottom comment"

```
        100 400 140 450 1 7
        time 0 1.5 .5   V 0 6 1
        t 1 0 A1 1 0 2 3 '+'
        t 1 0 B2 1 0 26 0 'B'
        t 1 0 C3 1.2 1 5 3 'o'
        t 1 0 D4 1 0 44 3 ' '
        end
        c: image outgif imageconv conv end
```



m4d.commands. Input colors: double, integer, 1-char name or string, multip

This example, using file convline created with the keyword example above, gives the results at the right.

Need file *lineplotfile*.Sets:*imagename*. Text and symbols will not be added unless the program can find the file font.list (and the files listed in file.list) in the current directory, or in the directory specified by *p.defaultdir*.

picture Draw grid lines, velocity vectors and fill contours for grid surfaces.
Input: *imagename, ix, iy, pixelperscale, action*
Create the image, *imagename,* of width *ix* pixels and height *iy* pixels. *pixelperscale* is the number of pixels to be used to represent one unit of geometric distance. For each letter in *action* take the corresponding action: 'g' draw grid lines; 'v' draw velocity vectors; 'f' contour fill a property. Note actions are added to the image in the order listed.

Example: c: picture imagete 500 500 1000 fgv

Picture uses several named arrays to describe how to draw the picture. These can be set with the command constant as illustrated below.
p.gnames Names of the grid arrays.
c: constant **p.gnames** s 4 idim4d xyz abcd clt
p.gxx Cartesian across and up vectors for plot.
c: constant **p.gxx** d 6 1 0 0 0 1 0
p.grange Limit a,b,c range, (as,ae,bs,be,cs,ce). Set anchor point (a,b,c) and its fractional location on plot, (facross, fup).
c: constant **p.grange** d 11 -2 1. 0 2 0 0 -2 0 0 .02 .02
Action g - grid lines - uses *p.gparms*. This contains, lwpixel if iw is io jf jw js jo kf kw ks ko which are the line-width, color numbers for i, j, k grid lines allowing different colors for flow (type 'f'), wall (type 'w'), solid (type 's') and other types of grid points. Lines connecting to grid points of type 'x' are omitted.
c: constant **p.gparms** i 13 1 0 1 0 0 0 1 0 0 0 1 0 0
Action v - velocity vectors - uses *p.vnames,* the names of the velocity components,
c: constant **p.vnames** s 3 U1 U2 U3
and *p.vparms* to describe how the vectors are plotted. p.*vparns* contains 11 items, dt headmax headf nybet lwpixel colotz colors colore dUz Uns Une.
c: constant **p.vparms** d 11 uscale .04 .2 0 1 5 11 20 0.2 -3. 3.
The component of the velocity vector in the viewing plane is shown with length dt times its magnitude (the result in geometry units). The head for the velocity vector is the smaller of headmax and headf times the length of the vector. If nybet = 0, on the points vectors are shown, otherwise between the points vectors are shown (in directions where more than 1 plane is included in the plot range). The linewidth for the vectors are lwpixel, in pixels. The out of plane component is shown by color. For the above example, colors 11 to 20 span out-of-plane U values from -3 to 3, but color 5 is used when the magnitude of out of plane component is < 0.2.
Action f - fill contours - uses *p.fnames* and *p.fparms*

c: constant **p.fnames** s 3 pg fi noout
c: constant **p.fparms** d 6 -1200. 0 -600. 50 -1 11
p.fnames contains the name of the property to be contoured, the point clt-types between which the contours should be filled, and a nyparm. If nyparm starts with 'n' contours outside the range given in *p.fparms* are not filled, otherwide they are filled with the end colors. If the contoured property is an on-the-points array, linear interpolation between the grid lines and a 4-point average in the center is used for filling. Mid-point and p-point arrays are contoured stepwise, with the between the points value.
p.fparms contains 3 values of the property, propstart propfix propend, and 3 color numbers, colorstart colorfix coloreend. If propstart<=propfix<=propend and colorfix >=0, a segmented color scheme is used with propstart to propfix contoured over the color range colorstart to colorfix, then propfix to propend contoured over colorfix to colorend. Otherwise a single color scheme is used based on the end values.

xytocgrid Create a c-grid or a flat "parallel"-normal grid around a body.
Input: *append, cs, ce, numasbe,*
as[j], *ae*[j], *bs*[j], *be*[j] for j=0,,,,,j<*numasbe*
distance, nptsn, expand (neg for flat + for c grid),
propname until *propname* = ""
Intended for 3-d grids of 2-d geometries where the third (k or c) direction is in the z direction. *append* - the appendix to be added to the name of all arrays converted to the new grid. *cs, ce* - the reange in c for the new grid. *numasbe* - the number of *as, ae, bs, be* starting and ending a,b values describing the wall, going around the object in a clockwise direction. The new grid extends a maximum distance, *distance* from the wall, and covers *nptsn* points in the wall normal direction. If *expand* is negative, the wall is shown flat in the new grid with the velocity vectors appropriately converted. Each *propname* that is for an on-the-points array is interpolated to the new grid.

Example: c: xytocgrid cg .3 .5 5 0 0 .5 1 0 1 1 1 1 1 1 0 1 0 0 0 0 0 0 .5
.5 100 1. qturb pg ""

This example does a c-grid around an object which extends from a= 0 to 1 and b= 0 to 1. With append="cg" the sample input creates arrays *idim4dcg, xyzcg, abcdcg, cltcg, U1cg, U2cg, U3cg, qturbcg, pgcg.*

Needs: *idim4d, abcd, xyz, clt, csym, U1, U2, U3.*

xyzicut Cut the grid in the i-grid direction to specified parallel planes.
Input: *append, nopl, A, B, C,*
S[j], j=0,,,,j<*nopl*
bs, be, cs, ce,
propname until *propname* = ""
Cut the grid in the i-grid direction to form planes of constant *Ax+By+Cz=S*[j] covering b from *bs* thru *be*, and c from *cs* thru *ce*. Choose *append* name (e.g. cut)

then xyzicut creates *idim4dcut xyzcut abcdcut cltcut* and *propcut* for all on-the-points *propname*s listed.

Example c: xyzicut cut 5   1. 0. 0.   -.143 -.111 -.079 -.048 -.016
                1. 2. 0. 1.   U1 U2 U3 cps cpt qturb u1u1 ""

Needs: *idim4d, abcd, xyz, clt, csym.*