APPENDIX J

Subroutine Listing

CFL3D contains all the subroutines listed below:

CBSEM	RHS	AF3F	BC	LBCX	LBCX (continued)	DYNPTCH	TURBS
mgblk	resid	af3f	bc	tau	findmin_new	dynptch	blomax
setup	resp	amafj	bc1000	tau2x	bbdist	global2	barth3d
rp3d	fa	swafj	bc1001	colldat	bbdst1	patcher	spalart
setblk	i2x	amafk	bc1002	collx	calc_dist	loadgr	twoeqn
global	i2xs	swafk	bc1003	collv	collect_surf	collapse	triv
lead	fa2xj	amafi	bc1005	collq	distcc	rechk	
qinter	fa2xk	swafi	bc1008	coll2q	distcg	expand	
qout	fa2xi	diagj	bc1011	collqc0	initf	invert	
qface	gfluxr	diagk	bc1012	collxt	initi	shear	
plot3c	hfluxr	diagi	bc1013	collxtb	ifree	arc	
plot3d	ffluxr	tinvr	bc2002	addx	ffree	diagnos	
plot3t	xlim	tdq	bc2003	addxv	iialloc	direct	
update	fhat	dlutr	bc2004	add2x	ifalloc	project	
updateg	fill	dfbtr	bc2005	add2xv	makebb	extra	
resetg	fluxp	dlutrp	bc2006	init	calcbb	extrae	
xtbatb	fluxm	dfbtrp	bc2007	initvist	getvrt	topol	
grdmove	gfluxv	dfhat	bc2102	rrest	shells	topol2	
resetwk	hfluxv	dfluxpm	chksym	rrestg	spltbb	dsmin	
rpatch	ffluxv	gfluxl	chkrap	wrest	push	xe	
setqc0	dird	hfluxl	chkrot	wrestg	pop	xe2	
setdqc0	wmag	ffluxl	rie1d	metric	sort_x	newfit	
resadd	delv	abciz	rie1de	tmetric	move_real	trace	
readdat	prntcp	abcjz	blockk	cellvol	move_integer	transp	
getdhdr	trans	abckz	blockj	ctime	triang	rotatp	
	transmc	dabciz	blocki	vlutr	heap_sort		
	rotate	dabcjz	cblkk	vlutrp	cntsurf		
	rotatmc	dabckz	cblkj	bsub	celcen		
	hole		cblki	bsubp	yplusout		
	dthole		int2	q8sdot	calyplus		
	blkmax		int3	q8smax	forceout		
			rotateq	q8smin	histout		
			rotateq0	q8ssum			
			rotateqb	q8vrev			
			bcchk	l2norm			
			xupdt	l2norm2			
			getibk	force			
			intrbc	csurf			
			ccf	csout			
				rsmooth			
				xmukin			
				findmin			

J.1 CBSEM Routines

mgblk

Advances the solution in time using multigrid acceleration. At each level in the multigrid procedure, all the blocks are advanced before moving to the next level.

setup

Reads in the grid and restart data and calculates some preliminary information, such as the metrics, for subsequent use.

rp3d

Reads in the grids in PLOT3D format.

setblk

Initializes the `blank` array.

global

Reads in the case input file.

lead

Installs the attributes of a particular block into the common blocks.

qinter

Interpolates the solution from a coarser mesh to a finer mesh. The finer mesh can be either a global mesh or an embedded mesh. Also updates grid position of finer mesh if meshes are in motion.

qout

Outputs data for plotting or printing.

qface

Determines the primitive variables at the edges of the grid and installs them in the `qj0/qk0/qi0` arrays for use in output routines.

`plot3c`

Writes the output file for the cell-centered points in PLOT3D format. (Outputs the grid and/or solution in single precision for use with FAST and/or PLOT3D.) Also prints solution data to a printout file for a specified range of points.

`plot3d`

Writes the output file at the grid points in PLOT3D format. (Outputs the grid and/or solution in single precision for use with FAST and/or PLOT3D.) Also prints solution data to a printout file for a specified range of points.

`plot3t`

Writes turbulent information for the cell-centered points in PLOT3D format.

`update`

Updates the solution in time.

`updateg`

Updates the grid to a new position and obtains corresponding grid-boundary velocities for use in the boundary conditions. Also collocates new grid position to coarser levels and obtains grid-boundary velocities on coarser levels. Also updates moment center.

`resetg`

Checks to see if any blocks in the grid have been translated and/or rotated out of the bounds set in the input deck. If so, resets these blocks so that they are at or near the initial positions. If the rotational displacement of a block is reset, the solution must also be rotated to correspond to the reset position. (Resetting is allowed only for constant translational speed, **itrans** = 1, or constant rotational speed, **irotat** = 1.)

`xtbatb`

Stores grid speeds and accelerations on the boundaries for use in setting no-slip and wall pressure boundary conditions for dynamic meshes.

`grdmove`

Moves the grid from one position to another.

`resetwk`

Replaces all locations of the work array which were filled with integer values in subroutine `plot3d` with real values. (Otherwise, problems may arise later when a real array attempts to access the integer value located in memory.)

`rpatch`

Reads in the generalized-coordinate interpolation data for grid patching from a file.

`setqc0`

Stores conservative variables for use in second-order temporal differencing and sub-iteration.

`setdqc0`

Stores conservative variables ($\mathbf{Q}_n - \mathbf{Q}_{n-1}$) for use in second-order temporal differencing.

`resadd`

Adds additional terms to the right-hand side for sub-iteration and second-order temporal accuracy.

`readdat`

Reads in auxiliary data arrays for the "2000 series" of boundary conditions.

`getdhdr`

Sets the character data for the main output file headers when the "2000" series of boundary conditions are used.

J.2 RHS Routines

`resid`

Computes the residual contributions to the right-hand side.

`resp`

Computes and prints residuals and sums the forces.

`fa`

Accumulates fluxes to insure conservation with grid embedding.

`i2x`

Interpolates the primitive variables from coarser meshes onto twice finer meshes for grid embedding.

`i2xs`

Interpolates the primitive variables from coarser meshes onto twice finer meshes for grid embedding.

`fa2xj`

Accumulates fluxes in the *j* direction for use on a twice coarser mesh to insure conservation with grid embedding.

`fa2xk`

Accumulates fluxes in the *k* direction for use on a twice coarser mesh to insure conservation with grid embedding.

`fa2xi`

Accumulates fluxes in the *i* direction for use on a twice coarser mesh to insure conservation with grid embedding.

`gfluxr`

Computes residual contributions for the right-hand side in the *j* direction from the inviscid terms.

`hfluxr`

Computes residual contributions for the right-hand side in the *k* direction from the inviscid terms.

`ffluxr`

Computes residual contributions for the right-hand side in the *i* direction from the inviscid terms.

`xlim`

Performs monotone interpolations to the interfaces of the cells.

`fhat`

Computes Roe's[31] generalized flux at the interface given the left and right states at the interface.

`fill`

Fills the edges of the `q` array for safety using a multi-plane vectorization technique.

`fluxp`

Computes the "positive" parts of the fluxes using the flux-vector-splitting method of van Leer.[39]

`fluxm`

Computes the "negative" parts of the fluxes using the flux-vector-splitting method of van Leer.[39]

`gfluxv`

Calculates the right-hand-side residual contributions in the j direction due to the viscous terms.

`hfluxv`

Calculates the right-hand-side residual contributions in the k direction due to the viscous terms when `idf` = 0. Calculates the implicit matrix terms when `idf` > 0.

`ffluxv`

Calculates the right-hand-side residual contributions in the i direction due to the viscous terms.

`dird`

Evaluates directed distance from the $k = 0$ wall and the $i = 0/j = 0$ wall for use in evaluating the Baldwin-Lomax[10] turbulence model.

`wmag`

Evaluates the vorticity magnitude for use in determining the turbulent eddy viscosity.

`delv`

Evaluates the velocity derivatives at cell centers.

`prntcp`

Writes the pressures on the body (actually the cell centers closest to the body) to an output file for unsteady flow.

`trans`

Determines the increment to grid position due to a grid translation.

`transmc`

Determines the increment to moment center due to a grid translation.

`rotate`

Determines the increment to grid position due to a grid rotation.

`rotatmc`

Determines the increment to moment center due to a grid rotation.

`hole`

Zeroes out the right-hand-side residuals for the blanked points.

`dthole`

Updates the Δt values for the hole and fringe cells. The values will be replaced with $\Delta t_{\min}$.

`blkmax`

Determines the location of the maximum residual.

J.3 AF3F Routines

`af3f`

Advances the solution in time using a 3-factor spatially-split approximate factorization algorithm.

`amafj`

Formulates the implicit matrices in the j direction for the 3-factor algorithm.

`swafj`

Solves the block 5×5 tridiagonal equations for the 3-factor spatially-split algorithm in the j direction.

`amafk`

Formulates the implicit matrices in the k direction for the 3-factor algorithm.

`swafk`

Solves the block 5×5 tridiagonal equations for the 3-factor spatially-split algorithm in the k direction.

`amafi`

Formulates the implicit matrices in the i direction for the 3-factor algorithm.

`swafi`

Solves the block 5×5 tridiagonal equations for the 3-factor spatially-split algorithm in the i direction.

`diagj`

Solves the scalar tridiagonal equations to approximate the spatially-split factor in the j direction of the 3-D spatially-split algorithm.

`diagk`

Solves the scalar tridiagonal equations to approximate the spatially-split factor in the k direction of the 3-D spatially-split algorithm.

`diagi`

Solves the scalar tridiagonal equations to approximate the spatially-split factor in the i direction of the 3-D spatially-split algorithm.

`tinvr`

Multiplies the inverse of the diagonalizing matrix **T** times the residual contribution ($\mathbf{T}^{-1}R$).

`tdq`

Multiplies the inverse of the diagonalizing matrix **T** times the change in characteristic combination of variables ($\mathbf{T}^{-1}\Delta\mathbf{q}$).

`dlutr`

Performs the scalar tridiagonal (LU) decomposition.

`dfbtr`

Performs the back substitution for a scalar tridiagonal system of equation.

`dlutrp`

Performs the LU decomposition for a periodic scalar tridiagonal system of equations.

`dfbtrp`

Performs the back substitution for a periodic scalar tridiagonal system of equations.

`dfhat`

Computes a Jacobian matrix with respect to the primitive variables at the cell interface. The Jacobian evaluation is approximate, being taken as either A^+ or $A^-(T\Lambda\mathbf{T}^{-1})$, and is computed with metric terms from the interface and dependent variables fro the cell centers.

`dfluxpm`

Computes “positive” or “negative” parts of the flux Jacobians using the flux-vector-splitting method of van Leer.[39]

`gfluxl`

Computes the left-hand flux contributions due to the inviscid terms for the j direction.

hfluxl

Computes the left-hand flux contributions due to the inviscid terms for the *k* directions.

ffluxl

Computes the left-hand flux contributions due to the inviscid terms for the *i* direction.

abciz

Zeroes out the left-hand-side matrix element with the help of the blank array. For a point with `blank` = 0, all elements of matrices `a` and `c` become zero. Only diagonal elements matrix `b` is changed to 1.0 for *i* implicit; *j* sweep.

abcjz

Zeroes out the left-hand-side matrix element with the help of the blank array. For a point with `blank` = 0, all elements of matrices `a` and `c` become zero. Only diagonal elements matrix `b` is changed to 1.0 for *j* implicit; *k* sweep.

abckz

Zeroes out the left-hand-side matrix element with the help of the blank array. For a point with `blank` = 0, all elements of matrices `a` and `c` become zero. Only diagonal elements matrix `b` is changed to 1.0 for *k* implicit; *j* sweep.

dabciz

Uses the blank values to modify the coefficient matrices, `a`, `b`, `c`, for the diagonal inversion in the *i* direction.

dabcjz

Uses the blank values to modify the coefficient matrices, `a`, `b`, `c`, for the diagonal inversion in the *j* direction.

dabckz

Uses the blank values to modify the coefficient matrices, `a`, `b`, `c`, for the diagonal inversion in the *k* direction.

J.4 BC Routines

`bc`

Determines boundary data/conditions at edges of grids.

`bc1000`

Sets free-stream boundary conditions.

`bc1001`

Sets symmetry plane boundary conditions.

`bc1002`

Sets extrapolation boundary conditions.

`bc1003`

Sets characteristic inflow/outflow boundary conditions.

`bc1005`

Sets inviscid surface boundary conditions.

`bc1008`

Sets tunnel inflow boundary conditions.

`bc1011`

Sets singular axis (half plane) boundary conditions.

`bc1012`

Sets singular axis (full plane) boundary conditions.

`bc1013`

Sets extrapolation boundary conditions for a singular axis.

`bc2002`

Sets pressure ratio; extrapolates other quantities.

bc2003

Sets characteristic inlet boundary conditions at engine inlet given (estimated) inlet Mach number, total pressure ratio, total temperature ratio, and flow angle.

bc2004

Sets solid wall (viscous wall) boundary conditions.

bc2005

Sets periodic boundary conditions given angular rotation angle to the periodic face and its block number.

bc2006

Sets pressure via radial equilibrium condition; extrapolates other quantities.

bc2007

Sets all the primitive variables with standard CFL3D normalization; $\tilde{\rho}/\tilde{\rho}_\infty$, $\tilde{u}/\tilde{a}_\infty$, $\tilde{v}/\tilde{a}_\infty$, $\tilde{w}/\tilde{a}_\infty$, $\tilde{p}/(\tilde{\rho}_\infty\tilde{a}_\infty^2)$.

bc2102

Sets the pressure ratio as a function of time; extrapolates the other flow-field quantities.

chksym

Checks for symmetry boundary conditions in j, k, or i directions in order to apply boundary condition type 1011 (singular axis with half-plane symmetry).

chkrap

Checks for wrap-around in j, k, or i directions in order to apply boundary condition type 1012 (singular axis with full plane).

chkrot

Checks to make sure that the proper rotation angle for periodic boundary conditions has been set.

rie1d

Determines far-field boundary data using quasi-1-D characteristic relations for boundary condition type 1003.

rie1de

Determines far-field boundary data using quasi-1-D characteristic relations for boundary condition type 2003.

blockk

Transfers information from the block designated `ir` to the `qk0` array of the block designated `it`.

blockj

Transfers information from the block designated `ir` to the `qj0` array of the block designated `it`.

blocki

Transfers information from the block designated `ir` to the `qi0` array of the block designated `it`.

cblkk

Checks information transferred from the block designated `ir` to the `qk0` array of the block designated `it`.

cblkj

Checks information transferred from the block designated `ir` to the `qj0` array of the block designated `it`.

cblki

Checks information transferred from the block designated `ir` to the `qi0` array of the block designated `it`.

int2

Linearly interpolates `q` from one grid to ghost cells of another grid using generalized coordinates without using a limiter on the gradients.

int3

Linearly interpolates q from one grid to ghost cells of another grid using generalized coordinates using a limiter on the gradients.

rotateq

Rotates the solution contained in array q through a specified angle and stores the rotated solution in qrot.

rotateq0

Rotates the solution at ghost points contained in array q0 (either qi0, qj0, or qk0 through the angle $\Delta\theta_x/\Delta\theta_y/\Delta\theta_z$ and stores the rotated solution in q0rot.

rotateqb

Rotates the solution in the qb array through the angle $\Delta\theta_x/\Delta\theta_y/\Delta\theta_z$ for the chimera scheme with rotating grids.

bcchk

Determines if the boundary conditions have been set and fills the end-points for safety.

xupdt

Updates the fringe points of overlapped grids with boundary values which have been interpolated from other grids to provide the mechanism for coupling the various grids.

getibk

Reads the output from MaGGiE (but not the grids).

intrbc

Interpolates the corrections for boundary values for all grids overlapped in the current mesh.

ccf

Modifies u_f, w_f, a_f (velocities and speed of sound at the far field) based on point vortex correction (used when **i2d** = -1 and the far-field boundary condition type is 1003).

J.5 LBCX Routines

tau

Computes a residual correction and stores the values of q for later use in determining corrections to finer grids in the multigrid iteration scheme.

tau2x

Puts the restricted residual from a finer embedded mesh into a coarser mesh.

colldat

Restricts auxiliary boundary condition data arrays to coarser meshes.

collx

Restrict x, y, and z values to coarser meshes.

collv

Restricts volumes to coarser meshes.

collq

Restricts q (the primitive variables) with a volume-weighted interpolation and residuals to coarser meshes. Also restricts turbulent eddy viscosity in the case of turbulent flows to coarser meshes.

coll2q

Restricts q (the primitive variables) with a volume-weighted interpolation and residuals from finer embedded meshes to coarser meshes.

collqc0

Restricts conservative variables $\mathbf{Q}_n$ and $\mathbf{Q}_n - \mathbf{Q}_{n-1}$ to coarser meshes via summation over fine-grid cells for use in time-accurate multigrid.

collxt

Restricts xt (array containing grid speeds) to coarser meshes.

collxtb

Restricts xtb and atb (arrays containing grid boundary velocity and acceleration, respectively) to coarser meshes.

addx

Interpolates the solution or the correction from a coarser mesh to a finer mesh.

addxv

Interpolates the turbulence quantities from a coarser mesh to a finer mesh during mesh sequencing.

add2x

Interpolates the solution or the correction from a coarser mesh to a finer embedded mesh.

add2xv

Interpolates the turbulence quantities from a global mesh to an embedded mesh during mesh sequencing.

init

Sets the initial conditions on a mesh to be free stream.

initvist

Sets the turbulent initial conditions on a mesh.

rrest

Reads the restart file for a block.

rrestg

Reads the restart file to get the latest position for a dynamic mesh, along with corresponding metrics and grid-boundary speeds. Also reads qc0 for a second-order accurate in time restart.

wrest

Writes the restart file for a block.

wrestg

Appends the latest position of a dynamic mesh to the end of the restart file. Also writes qc0 for a second-order accurate in time restart.

metric

Calculates the cell-interface directed areas.

tmetric

Calculates the time-metric terms for a grid in motion.

cellvol

Calculates the cell volumes.

ctime

Calculates the time step for an input fixed Courant number or calculates the Courant number based on an input value of Δt.

vlutr

Performs the LU decomposition for a block 5×5 tridiagonal system of equations.

vlutrp

Performs the LU decomposition for a block 5×5 tridiagonal system of equations which is periodic.

bsub

Performs the back substitution for a block 5×5 tridiagonal matrix equation solution.

bsubp

Performs the back substitution for a block 5×5 tridiagonal matrix equation solution which is periodic.

q8sdot (function)

Computes the dot product between two vectors.

`q8smax (function)`

Finds the maximum value in an array.

`q8smin (function)`

Finds the minimum value in an array.

`q8ssum (function)`

Sums the elements of a vector

`q8vrev`

Reverses the elements in an array.

`l2norm`

Computes the L2-norm of the residuals or the change in primitive variables from one time to the next.

`l2norm2`

Computes the L2-norm of the residuals, after subtracting out the contribution of the unsteady terms that were added in subroutine `resadd`.

`force`

Integrates the forces on the body.

`csurf`

Integrates control surface mass flow and momentum/forces.

`csout`

Writes control surface mass flow and momentum/forces to an output file.

`rsmooth`

Performs implicit residual smoothing (constant coefficient).

`xmukin`

Computes Sutherland's formula with linear law at low temperatures.

`findmin`

Finds minimum distances from field points to viscous wall(s).

`findmin_new`

Serves as a driver routine for computing distances to the closest viscous surface.

`bbdist`

Serves as a driver routine for determining the nearest bounding box for each field point.

`bbdst1`

Identifies the nearest bounding box for each field point.

`calc_dist`

Finds the minimum distance to field points using the recursive-box algorithm.

`collect_surf`

Stores coordinates of surface points and identifies the neighbors of each surface point.

`distcc`

Averages the distance function based at grid points to one based at cell centers.

`distcg`

Collocates the fine-grid minimum distance function on a fine grid to a coarser grid.

`initf`

Initializes pointers to floating-point variables in recursive-box algorithm.

`initi`

Initializes pointers to integer variables in recursive-box algorithm.

`ifree`

“Frees up” pointers to integer variables in recursive-box algorithm.

`ffree`

"Frees up" pointers to floating-point variables in recursive-box algorithm.

`iialloc (function)`

Increments pointers to integer variables in recursive-box algorithm.

`ifalloc (function)`

Increments pointers to floating-point variables in recursive-box algorithm.

`makebb`

Serves as a driver routine for generating the bounding boxes for the recursive-box algorithm.

`calcbb`

Calculates bounding boxes.

`getvrt`

Searches for all points that fall within a bounding box.

`shells`

Performs a shell sort.

`spltbb`

Subdivides bounding boxes.

`push`

Places an item into the stack and adjusts the pointer accordingly.

`pop`

Removes an item from the stack and adjusts the pointer accordingly.

`sort_x`

Sorts surface points with respect to x-coordinate.

`move_real`

Rearranges items in the real array `x` based on the pointer `iperm`.

`move_integer`

Rearranges items in the integer array `x` based on the pointer `iperm`.

`triang`

Finds the closest distance from a field point to the actual surface (i.e. not simply the closest discrete surface point), using local triangulation of the surface.

`heap_sort`

Sorts a list of points.

`cntsurf`

Counts the number of viscous surface points.

`celcen`

Finds cell centers of field points.

`yplusout`

Calls the routines necessary for calculating yplus at the first grid point above solid walls.

`calyplus`

Calculates y^+ in turbulent flows at solid surfaces in a block and calculates statistics for the y^+ distribution (average y^+, maximum y^+ and its location, standard deviation).

`forceout`

Writes the forces and moments on individual blocks, as well as a global force/moment summary, to an output file.

`histout`

Writes the convergence history for mean-flow equations and turbulence equations to an output file.

J.6 DYNPTCH Routines

dynptch

Establishes zone-to-zone communication for block interfaces that move relative to one another, using a patched-grid technique (nonconservative).

global2

Reads the dynamic patch input parameters.

patcher

Calculates patched-grid interpolation coefficients.

loadgr

Loads the proper grid from a 1-D storage array to a 2-D work array.

collapse

Checks for collapsed points in the grid and expands any collapsed lines detected.

rechk

Checks for branch cuts.

expand

Expands the grid at boundaries.

invert

Determines generalized coordinates of cell centers of the “to” grid in terms of the generalized coordinate system(s) defined on the “from” grid(s).

shear

Determines the generalized coordinates of cell edge midpoints on $\xi = 0$ and $\eta = 0$ boundaries and determines the requisite shearing correction to the generalized coordinates near $\xi = 0$ and/or $\eta = 0$ boundaries.

arc

Performs arc-length correction to the generalized coordinates near a boundary if required when shearing correction has failed.

`diagnos`

Performs diagnostic checks on interpolation coefficients (generalized coordinates) found via search and inversion routines.

`direct`

Computes normalized directed area components, or equivalently, components of the unit normal to a cell face.

`project`

Projects a point onto a plane.

`extra`

Computes extra mid-cell points in ξ direction.

`extrae`

Computes extra mid-cell points in η direction.

`topol`

Searches appropriate “from” blocks for current “to” cell center. Driver routine for determining ξ and η of the cell center.

`topol2`

Searches appropriate “from” blocks in one direction only for current “to” cell center

`dsmin`

Finds the closest point in a grid to a specified point.

`xe`

Selects proper coordinates to use for inversion.

`xe2`

Sets up the coefficients for locally fitting a polynomial variation in the ξ and η directions.

`newfit`

Determines a new polynomial fit for cells with stubborn convergence.

`trace`

Writes the search routine history for the current “to” cell to unit 7.

`transp`

Translates the “from” block to provide complete coverage for interpolation for cases in which the complete physical domain is not modeled.

`rotatp`

Rotates “from” block to provide complete coverage for interpolation for cases in which the complete physical domain is not modeled.

J.7 TURBS Routines

`blomax`

Computes the turbulent viscosity distributions using the Baldwin-Lomax[10] two-layer eddy-viscosity model.

`barth3d`

Computes the turbulent viscosity distributions using the one-equation Baldwin-Barth[9] turbulence model.

`spalart`

Computes the turbulent viscosity distributions using the one-equation Spalart[35] turbulence model.

`twoeqn`

Computes the turbulent viscosity distributions using the two-equation turbulence models.

`triv`

Solves a scalar tridiagonal system of equation