1 Introduction

Finite element methods are frequently used in computational fluid dynamics to study unsteady and turbulent flows. The simulation of fluid dynamics problems are usually computationally expensive since the number of mesh elements is huge and three dimensional calculations are often necessary. So simplifications and assumptions on the studied flow can produce acceptable results for the finite amount of computational resources at hand.

When three-dimensional, time-dependent turbulent inflow data for simulations of complex spatially developing boundary layers is needed, Scaled Plane Extraction Boundary Condition (SPEBC) can be used to impose the inlet boundary condition on the flow variables, provided that the flow has homogeneity in one direction. SPEBC is a method of imposing boundary condition for the inflow variables by scaling the velocity field at a downstream location and then interpolating the obtained values to the inlet plane. By imposing homogeneity in one direction, the average velocity field in the boundary layer is essentially two-dimensional and various scaling laws can be used to characterize the flow field.

In this report first the finite element method used in the compressible 1.6 version of PHASTA (Parallel Hierarchic Adaptive Stabilized Transient Analysis) code. This method is SUPG (Streamline Upwind Petrov-Galerkin). The theory behind SPEBC will be presented in 2.2. Finally the structure of PHASTA will be described with a simplified explanation of each routine present. In particular the routines needed for SPEBC will be described and modifications presented.

2 Theory

2.1 Formulation

In PHASTA the compressible Navier-Stokes equations are used to solve fluid dynamic problems. Starting from these equations written in conservative form (see Jansen [8, 6] for details) the weak form will be derived. So the continuity, momentum and energy equations can be written in compact form as follows:

\[ U_{,t} + F_{i,;} = \mathcal{F} \]  \hspace{1cm} (1)

where

\[ U = \{ U_i \} = \rho \left\{ \begin{array}{c} 1 \\ \frac{u_i}{e_{tot}} \end{array} \right\}, \quad F_i = u_i U + p \left\{ \begin{array}{c} 0 \\ \delta_{ij} \end{array} \right\} + \left\{ \begin{array}{c} 0 \\ -\tau_{ij} \\ -\tau_{ik} u_k + q_i \end{array} \right\} \quad (2) \]

\[ F_{i,;} = \mathcal{F}_{i,;} \]

and

\[ \tau_{ij} = 2(\mu + \mu_T)(S_{ij}(u) - \frac{1}{3}S_{kk}(u)\delta_{ij}) \quad , \quad S_{ij}(u) = \frac{u_i,j + u_j,i}{2} \]  \hspace{1cm} (3)
\[ q_i = -(\kappa + \kappa_T) T_i, \quad \kappa_T = c_p \frac{\mu T}{P T}, \quad e_{tot} = e + \frac{u_i u_i}{2}, \quad e = c_v T \quad (4) \]

The variables are: the velocity \( u_i \), the pressure \( p \), the density \( \rho \) and the total energy \( e_{tot} \) and the temperature \( T \). Finally \( F \) is a body force (or source) vector. \( U \) is the vector of conservative variables, as discussed in Hauke and Hughes [4], it is often not the best choice of solution variables, particularly when the flow is nearly incompressible. Instead for the derivation of the SUPG stabilized method pressure-primitive variables are used:

\[ Y = \begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \\ Y_5 \end{bmatrix} = \begin{bmatrix} p \\ u_1 \\ u_2 \\ u_3 \\ T \end{bmatrix} \quad (5) \]

For the specification of the methods that follow, it is helpful to define the quasi-linear operator as

\[ \mathcal{L} = A_0 \frac{\partial}{\partial t} + A_i \frac{\partial}{\partial x_i} - \frac{\partial}{\partial x_i} (K_{ij} \frac{\partial}{\partial x_j}) \quad (6) \]

Here \( A_0 = U \cdot Y \) is the change of variables metric, \( A_i = F_{i, j} \cdot Y \) is the \( i \)th Euler Jacobian matrix, and \( K_{ij} \) is the diffusivity matrix, defined such that \(-K_{ij} Y_{, j} = F_{i, j}^{\text{diff}}\). For a complete description of \( A_0, A_i \) and \( K_{ij} \), the reader is referred to [3, 5]. Using this, we can write (1) as simply \( \mathcal{L} Y = F \).

To proceed with the finite element discretization of the Navier-Stokes equations (1), the finite element approximation spaces must be defined. First, let \( \Omega \subset \mathbb{R}^3 \) represent the closure of the physical spatial domain (i.e. \( \Omega \cup \Gamma \) where \( \Gamma \) is the boundary). In addition, \( H^1(\Omega) \) represents the usual Sobolev space of functions with square-integrable values and derivatives on \( \Omega \).

Next, \( \Omega \) is discretized into \( n_d \) finite elements, \( \Omega \). With this, the trial solution space for the semi-discrete formulations is

\[ \mathcal{V}_h = \{ Y | Y(\cdot, t) \in H^1(\Omega)^m, t \in [0,T], Y|_{x \in \Omega} \in P_k(\Omega)^m, Y(\cdot, t) = g \text{ on } \Gamma_g \}, \quad (7) \]

and the weight function space is

\[ \mathcal{W}_h = \{ W | W(\cdot, t) \in H^1(\Omega)^m, t \in [0,T], W|_{x \in \Omega} \in P_k(\Omega)^m, W(\cdot, t) = 0 \text{ on } \Gamma_g \}, \quad (8) \]

where \( P_k(\Omega) \), is the space of all polynomials defined on \( \Omega \), complete to order \( k \geq 1 \), and \( m \) is the number of degrees of freedom (\( m = 5 \)).

To derive the weak form of (1), the entire equation is dotted with a vector of weight functions, \( W \in \mathcal{W}_h \), and integrated over the spatial domain. Integration by parts is then performed on the integral with \( F_i \) to move the spatial derivatives onto the weight functions. This process leads to the integral equation (often referred to as the weak form): find \( Y \in \mathcal{V}_h \) such that

\[ 0 = \int_{\Omega} (W \cdot A_0 Y_{, t} - W_{, i} \cdot F_i - W \cdot F) \, d\Omega + \int_{\Gamma} W \cdot F_i \, n_i \, d\Gamma \quad (9) \]

The first line of (9) contains the Galerkin approximation (interior and boundary) and the second line contains the SUPG (Streamline Upwind Petrov Galerkin, see [1] for details) stabilization. The stabilization matrix \( \tau \) is an important ingredient in these methods and is well documented in Shakib [10] and in Franca and Frey [2].

To develop a numerical method, the weight functions \( W \), the solution variable \( Y \), and its time derivative \( Y_{, t} \) are expanded in terms of basis functions (typically piecewise polynomials). On
element level this is:

\[ W = \sum_{b=1}^{n_{en}} N_b(\xi) W_b^e \]  
(10)

\[ Y = \sum_{a=1}^{n_{en}} N_a(\xi) Y_a^e \]  
(11)

\[ Y_{,t} = \sum_{a=1}^{n_{en}} N_a(\xi) Y_{a,t}^e \]  
(12)

\[ Y_{,i} = \sum_{a=1}^{n_{en}} N_{a,i}(\xi) Y_{a,i}^e \]  
(13)

where \( n_{en} \) is the number of element nodes and \( \xi \) is the local coordinate system. With some manipulations and inserting (10) to (13) into (9), and by noting that \( W_b^e \) are arbitrary, equation (9) becomes:

\[ 0 = G_B(Y, Y_{,t}) = \bigwedge_{e=1}^{n_{el}} G_B^e(Y, Y_{,t}) \]  
(14)

where

\[ G_B^e = \int_{\Box} [N_b(A_b Y_{,t} - F) - N_{b,i} F_i] Dd\Box \]

\[ + \int_{\Box} \hat{L} N_b T \{LY - F\} Dd\Box + \int_{\Box'} N_b F_i n_i D1d\Box' \]  
(15)

The integrals (15) are then evaluated using Gauss quadrature resulting in a system of non-linear ordinary differential equations. Finally this system is discretized in time via a generalized-\( \alpha \) time integrator (see [7]) resulting in a non-linear system of algebraic equations. This system is in turn linearized with Newton's method which yields a linear algebraic system of equations to be solved at each Newton iteration.

\[ \frac{G_A}{R_A} \begin{pmatrix} Y_{,t}^{(l)} \\ Y^{(l)} \end{pmatrix} + \sum_{B=1}^{n_{el}} \frac{\partial G_A}{\partial Y_B^{(l)}} \begin{pmatrix} Y_{,t}^{(l)} \\ Y^{(l)} \end{pmatrix} = \frac{M_{AB}}{N_{AB}} \]  
(16)

In this equation \( \alpha_f \) and \( \alpha_m \) are parameters of the generalized-\( \alpha \) method. Newton iterations continue until the non-linear residual is satisfied at each time step, after which the method proceeds to the next time step, starting the process over again.

### 2.2 SPEBC

The scaled plane extraction boundary condition is used in boundary layers where initial perturbations are smoothed out using the fact that velocity profile in the boundary layer is self-similar. Analytically, a approximate solution for the flat plate boundary layer was found by Blasius using a similarity solution [3].

In SPEBC using the Blasius approximation, the solution at the inlet plate is extracted from the recycle plane by the following equation:

\[ u_{inl}(y) = u_{recy} \left( \frac{\delta_{inl}}{\delta_{recy}} y \right) \]  
(17)

where \( u_{inl} \) is the solution inside the boundary layer of inlet plane, \( u_{recy} \) is the known solution at some position inside the boundary layer at the recycle plane, \( \delta_{inl} \) and \( \delta_{recy} \) and are the boundary layer
thicknesses at the inlet and recycle positions respectively. For simplicity two dimensional flow is assumed in equation (17). This can be easily extended to three dimensional flow with homogeneity in the third direction by taking a spatial average of the velocity in the spanwise direction, then equation (17) holds for spanwise averaged velocity.

Blasius approximation is derived for laminar boundary layers. For turbulent boundary layers, the turbulent rescaling model from Lund, Wu and Squires can be used. This method does two rescaling depending where in the boundary layer the point lies. The law of the wall is used in the inner portion of the boundary layer and defect law in the outer portion the boundary layer, in between a combination of both [9]. The flow must be homogeneous in the spanwise direction so that the mean velocity $U_i$ is averaged in time and in the spanwise direction. Let define the velocity fluctuation as follows:

$$u'_i (x, t) = u_i (x, t) + U_i (x, y)$$  \hspace{1cm} (18)

For the law of the wall the friction velocity must be defined:

$$u_r = \sqrt{\nu \left( \frac{\partial u}{\partial y} \right)_{wall}}$$ \hspace{1cm} (19)

as well as the wall coordinate:

$$y^+ = \frac{u_r y}{\nu}$$ \hspace{1cm} (20)

For the defect law the outer coordinate used is:

$$\eta = \frac{y}{\delta}$$ \hspace{1cm} (21)

This is the similarity variable also used for Blasius rescaling. So the instantaneous velocity at the inlet plane is calculated as follows:

$$(u_i)_{inl} = \left[(U_i)_{inl}^{inner} + (u'_i)_{inl}^{inner}\right] \{1 - W(\eta_{inl})\} + \left[(U_i)_{inl}^{outer} + (u'_i)_{inl}^{outer}\right] W(\eta_{inl})$$ \hspace{1cm} (22)

where the mean velocities and the fluctuations at the inlet plane are related to those of the recycle plane as follows:

$$U_{inl}^{inner} = \gamma U_{recy} (y_{inl}^+ / \nu)$$ \hspace{1cm} (23)

$$U_{inl}^{outer} = \gamma U_{recy} (\eta_{inl}) + (1 - \gamma) U_{\infty}$$ \hspace{1cm} (24)

$$V_{inl}^{inner} = \gamma V_{recy} (y_{inl}^+)$$ \hspace{1cm} (25)

$$V_{inl}^{outer} = \gamma V_{recy} (\eta_{inl})$$ \hspace{1cm} (26)

$$(u'_i)_{inl}^{inner} = \gamma (u'_i)_{recy} (y_{inl}^+, z, t)$$ \hspace{1cm} (27)

$$(u'_i)_{inl}^{outer} = \gamma (u'_i)_{recy} (\eta_{inl}, z, t)$$ \hspace{1cm} (28)

and

$$\gamma = \frac{u_{r, inl}}{u_{r, recy}}$$ \hspace{1cm} (29)

Finally the weighting function $W(\eta)$ is defined as

$$W(\eta) = \frac{1}{2} \left\{ 1 + \frac{\tanh \left( \frac{\alpha (\eta - b)}{(1-2b)\eta + b} \right)}{\tanh(\alpha)} \right\}$$ \hspace{1cm} (30)

where $\alpha = 4$ and $b = 0.2$. For the scaling both $u_r$ and $\delta$ need to be known at the inflow and the recycle plane. At the recycle plane those quantities can be determined from the mean velocity profile, but for the inlet plane they need to be specified. In reality only $\delta_{inl}$ needs to be specified and $u_{r, inl}$ is obtained by:

$$u_{r, inl} = u_{r, recy} \left( \frac{\delta_{recy}}{\delta_{inl}} \right)^{1/8}$$ \hspace{1cm} (31)

where $\theta$ is the momentum thickness.
Code Description

3.1 General Description

To understand the code's design, first it will be explained what parameters and data the code needs, as well as how it is done. Then the preprocessing of data will be described. The resolution of Navier-Stokes equations is divided into five sections: the time integration, linear equation formulation, boundary condition prescription, SPEBC prescription and the linear equation solver. Finally it will be seen how the results are output. First appedned figure shows the general flow diagram of the compressible PHASTA (x, 14) code. The main() routine calls input.fform() and input which are explained in 3.2 and proces actually simulate the flow. This routine calls gendat which generates all the needed data (3.2), initTurb which initialize parameters for RANS turbulence modeling, initNABI which initialize parameters for the boundary fluxes projection, restar (3.3.6) and itrdrv which is the space-time predictor-corrector iterative driver (3.4).

3.2 Input of data

3.2.1 input.fform()

The input of different parameters and geometry data is done in different subroutines. First the subroutine input.fform() written in C++ is called by the main program. This routine takes as argument the name of the input file which by default is solver.inp, and reads in the modeling parameters for the current simulation from the files input.config and solver.inp. The first file sets default values for some problem parameters and lists all possible modeling parameter values. This file is never modified by the user. The user inputs in solver.inp the required modeling parameters that are different from those in input.config. In these files, the user specifies if the simulation is compressible or incompressible, the number of timesteps, the size of timesteps to be taken, some material properties like viscosity, thermal conductivity, body forces or surface tension, which solver to use depending if the compressible or incompressible case is calculated. The user also specifies the number of Krylov vectors for GMRES sweeps. The quadrature rules for interior and boundary elements are specified, as well as, the number of elements per block. In the SPEBC Modeling Parameters section of these files, the user needs to specify the location of the inlet and recycle planes (for now it is done by specifying the x coordinate of the two planes), the boundary layer thickness at the inlet and the velocity scale factor, when SPEBC is used. The scaling law is also specified in this section. In these files parameters for turbulence modeling or cardiovascular modeling are also specified. This is only a brief description of the input parameters set in input.config and solver.inp. The subroutine input.fform() checks for errors in the reading of input files, specifies if there is any to the user and returns an error code to the main program.

3.2.2 input

Once the general parameters for this simulation are known the mesh informations need to be gathered. This is done in the subroutine input which needs to know the number of processors used for this simulation and on which processor is the software currently executed. Figure 2 at the end of the report describes the structure of this routine. This routine first calls the subroutine readblk which reads in data, then writes the global information on this simulation into echo.dat, checks the validity of inputed control parameters, allocates memory for (Fessenberg and Krylov) vectors for GMRES linear solvers, calls the subroutine geninit which generates the spatial integration rules and computes the fluid thermodynamic properties for this simulation.

3.2.3 readblk

The subroutine readblk reads in or calculates the following parameters:
nump - total number of nodes;
nshg - global number of shape functions which can be more then \[ \text{nump} \] if using hierarchical basis functions where there is shape functions associated with edges, faces and even volumes in addition to those associated with vertices (for piecewise linear shape functions \( nshg = \text{nump} \));

\text{numel} - total number of internal elements on current processor;

\text{numelb} - total number of boundary elements in the mesh;

\text{nen} - maximum number of nodes per element;

\text{nelblk} - number of interior element blocks;

\text{nelblb} - number of boundary element blocks;

\text{numpbc} - total number of nodes with prescribed essential boundary conditions on current processor;

\text{ntopsh} - number of shape functions on current processor;

\text{nsd} - number of spatial dimensions;

\text{nflow} - total number of flow variables (for compressible Navier-Stokes the primitive variables more often used are: pressure, three components of velocity and temperature);

\text{ndof} - total number of variables which \text{nflow} plus number of scalar variables to solve for;

\text{nfath} - number of \underline{inhomogeneous nodes} called fathers over which certain quantities will be averaged;

\text{nsomax} - maximum number of nodes associated with a father-node.

It creates the following arrays:

\text{point2x(nump, nsd)} - array containing the node coordinates (also called \( x \));

\text{nBC(nshg)} - this array tells if a node (or generally the shape function) has a prescribed essential boundary condition;

\text{iBCtmp(numpbc)} - for each node with a boundary condition this array tells which of the following conditions it is: density, temperature, pressure, three components of velocity, up to four scalars, periodicity and SPEBC (it is a temporary iBC array);

\text{BCinp(numpbc, ndof+7)} - for each boundary condition this array contains the value specified by the user;

\text{point2iper(nshg)} - array containing the master node for each periodic slave, i.e. for periodic boundary condition two nodes are connected and one is set as a master and the other as the periodic slave;

\text{point2soms(nfath)} - array containing the number of sons or \underline{homogeneous points} for each father;

\text{point2ifath(nshg)} - array containing the element number of its father for a given son;

\text{qold(nshg, ndof)} and \text{aqold(nshg, ndof)} - arrays containing respectively the values of primitive variables and their time derivatives read from the restart files.

This subroutine calls the \text{genblk} and \text{genblk2d} subroutines which create respectively, internal element blocks and boundary element blocks. To initialize SPEBC set \text{SPEBC} and \text{renum} routines are called.
3.2.4 Blocks Generation

The routine genblk generates the lcb1k(10, nelblk) array for interior elements. This array contains the following parameters for each block:

lcb1k(1, .) is the global number of the first element of this block (iel);

lcb1k(2, .) is not used (icpen);

lcb1k(3, .) is the element coordinate system or element type: tetrahedron, hexahedron, wedge, pyramid (icsyst);

lcb1k(4, .) is not used (ipordl);

lcb1k(5, .) is the number of nodes per element (nenl);

lcb1k(6, .) is number of faces in an element (nfacel);

lcb1k(7, .) is the material type (fluid or solid) and is represented by mtyp;

lcb1k(8, .) is the number of degrees of freedom for this block (ndof1);

lcb1k(9, .) is the number of symmetric degrees of freedom (nsym1);

lcb1k(10, .) contains the number of shape functions per element (nsh1).

The subroutine genblb generates the lcb1kb(10, nelbelb) array. The difference in the parameters from lcb1k to lcb1kb are:

lcb1kb(6, .) is number of boundary nodes in an element (nenlb);

lcb1kb(9, .) contains the number of shape functions per element (nshb);

lcb1kb(10, .) contains the number of boundary shape functions per element (nshlb);

The subroutines gensav and gensvb which are called by genbl1k and genblb respectively generate the connectivity arrays ien(npro, nshl) and ienb(npro, nshlb) which contain the global node number for a node of the current element.

3.3 Pre-processing of data

3.3.1 genint

This routine gets the quadrature data for interior and boundary elements depending on the element type and power of integration and creates Qpt(., 4, nint), Qptb(., 4, nintb), Qwt(., nint) and Qwtb(., nintb) arrays where nint and nintb are the number of integration points for the element and its faces, respectively. These arrays are the quadrature points location for the elements and their faces, and their respective weights. The first argument in these arrays represent the element type which are tetrahedron (1), hexahedron (2), wedges (3,4) and pyramids (5,6). Wedges and pyramids have two entries in Qwt and Qwtb because their faces have variable number of vertices, for example the pyramid has four faces with three vertices and one with four. The weights and location of quadrature points are calculated in the following routines: symhex(), sympyr(), symquad(), symquadw(), symtet(), symtri(), symtripyr() and symwdg().

3.3.2 gendat

Figure 3 of the annex shows the structure of gendat. This routine inputs the geometry and the boundary conditions. First the domain size is calculated by the xyzbound(x) subroutine. The subroutine gendat then calls gensh which generates the shape functions for the interior nodes. Afterwards the boundary condition codes and essential boundary conditions are generated by geniBC and genBC respectively. The shape functions for the boundary elements are generated by genshp subroutine. Finally the initial conditions are generated by gemini.
3.3.3 genshp and genshp2

The subroutine genshp generates and returns shp(., nshl, nint) and shgl(., nshl, nint) arrays for the hierarchical basis and Lagrangian basis respectively for each interior element block. The subroutine genshp2 generates and returns the corresponding arrays for each boundary element block (shpb(., nshl, nint) and shglb(., nshl, nint)). The first argument indicates the element type (see section 3.3.1). The shape functions are calculated in shpTet(., shphex(., shp6w) and shppyr(.) files depending on their element type.

3.3.4 genBC

This routine returns iBC(nshg) array calculated using nBC and iBCtmp arrays. This new array gives the types of essential boundary conditions for each unknown using the binary system where density is given by the first bit (2^0), temperature by the second (2^1), pressure by the third (2^2), next three bits are the three components of velocity (x-velocity: 2^3, y-velocity: 2^4 and z-velocity: 2^5), the next four represent scalars (2^6, 2^7, 2^8 and 2^9), then periodicity (2^10) and lastly SPEBC (2^11). For example if a node has a prescribed density, temperature, z-velocity and SPEBC it would have: 1 + 2^1 + 1 + 2^3 + 1 + 2^8 + 0 + 2^2 + 0 + 2^0 + 2^4 + 1 + 2^4 + 0 + 2^6 + 0 + 2^7 + 0 + 2^8 + 0 + 2^9 + 0 + 2^10 + 1 + 2^11 = 2083 (or in binary 110001000001)

3.3.5 genBC and genBC1

The routine genBC generates the essential boundary conditions by calling genBC1. It takes as inputs iBC and nBC and output BC(nshg, n dofBC). This routine first creates the temporary array BCtmp(nshg, n dof+7) from BCtmp array, then corrects it for wall normal as needed by calling genWall routine. It also calls genBC1 which determines the first point off the wall for each wall node. Nodes that are not on a wall point to themselves. They are stored in otwn(nshg) array. Then genBC1 is called to convert the input boundary conditions to condensed version. Finally the boundary condition input data is echoed.

The routine genBC1 adjusts the boundary conditions to accommodate for the velocity constraints in the non-axes directions and copies it to BC array. It copies first the prescribed scalar boundary conditions form BCtmp starting from the seventh position in BC. It then puts the density or pressure boundary conditions in the first place of BC and the temperature in the second. The next four contain the boundary conditions for velocities depending on how many components are set. Density and pressure cannot be set at the same time for a given node.

3.3.6 genini

This routine reads in the nodal coordinate array x and the boundary condition arrays iBC and BC and outputs the initial values of Y variables y(nshg, n dof) and their derivatives ac(nshg, n dof). It reads in y and ac from the restart file by calling restar. The routine restar puts the values from qold and acold arrays read in in routine readablk into y and ac arrays if the code is 'in' and does the inverse operation if the code is 'out', plus, then writes qold and acold in the restart file. The subroutine genini then gets, if SPEBC is used, the mean velocity array velbar(nfath, n dof) from the subroutine rwvelb. This subroutine reads in velbar if it exists when the code is 'in' and writes it out when the code is 'out'. If velbar does not exist yet the routine getvel is called to create it. If time varying boundary conditions are used genini calls initBC and BCint routines which read in these conditions from bct.dat into BC. Finally the boundary conditions are satisfied by calling itvBC (see section 3.6.1) and genscale (see section 3.7.3).

The routine getvel calculates the mean velocity array velbar. The velocity is averaged for homogeneous direction (i.e. over all sons) and for time. In solvei.inp when using SPEBC the user sets the velocity averaging steps which is used for time averaging. If it is not set, then the time averaging is done for all steps already calculated.
3.4 Non-linear Equation Marcher

The \texttt{itrdrv} routine (figure 4) is the iterative driver which uses the generalized-\alpha method to convert the system of nonlinear ODEs (14) into a system of nonlinear algebraic equations. First it generates the sparse data vectors (\texttt{genadj}) needed for sparse GMRES solver and allocates \texttt{EGmass} for the element by element GMRES solver. The \texttt{itrdrv} routine loops over the time sequences where first it calls \texttt{itrSetup}, then it predicts the solutions in \texttt{itrPredict}, calls \texttt{itrBC} to check the predicted \( Y_i^{(i)} \), then it goes into the multi-corrector phase. In the multi-corrector phase first the solver is called \( n+\alpha_f \) which calculates \( \Delta Y^{(i)} \), then \texttt{itrCorrect} is called when it is an update to correct the found solution which is then checked against the boundary conditions in \texttt{itrBC} and updated for SPEBC by calling \texttt{genscale}. At the end of one timestep, first the time integration parameters are reassigned to the initial values, then the solution is updated by calling \texttt{itrUpdate} and once more the solution is passed through \texttt{itrBC} and \texttt{genscale} to check for the boundary conditions. Finally it calls \texttt{Bflux} which outputs the data.

The \texttt{itrSetup} routine set up parameters needed for the time integration, i.e.:

\begin{align}
\alpha_m &= \frac{1}{2} \left( \frac{3 - \rho_\infty}{1 + \rho_\infty} \right) \\
\alpha_f &= \frac{1}{1 + \rho_\infty} \\
\gamma &= 0.5 + \alpha_m + \alpha_f
\end{align}

The \texttt{itrPredict} routine predicts the solution at the beginning of a timestep by setting it to \( n \) \( Y^{(i)} \) and setting the acceleration to \( n+\alpha_f \) \( Y^{(i)} \).

The \texttt{itrCorrect} routine corrects the solution for the update part of the step sequence. It adds \( n+\alpha_f \) \( \Delta Y^{(i)} \) to \( Y^{(i)} \), then it corrects the acceleration array:

\begin{equation}
\frac{n+\alpha_m}{\gamma \Delta t \alpha_f} Y^{(i)} = Y^{(i)} \left( 1 - \frac{\alpha_m}{\gamma} \right) + \left( Y^{(i)} - \hat{Y} \right) \frac{\alpha_m}{\gamma \Delta t \alpha_f}
\end{equation}

where \( Y^{(i)} \) and \( \hat{Y} \) are the solution and the acceleration vectors respectively.

The \texttt{itrUpdate} routine updates the solution and acceleration vectors and puts them in \texttt{yold} and \texttt{acold} as follows:

\begin{align}
n^{n+1} Y &= Y^{(i)} + \frac{1}{\alpha_f} \left( Y^{(i)} - \hat{Y} \right) \\
n^{n+1} Y_{,t} &= Y_{,t}^{(i)} + \frac{1}{\alpha_m} \left( Y_{,t}^{(i)} - Y_{,t} \right)
\end{align}

3.5 Linear Equation Formation

3.5.1 \texttt{ElmXXX}

The subroutines \texttt{ElmGMR}, \texttt{ElmGMRs} and \texttt{ElmMGF} return the global residuals, \texttt{res(nshg, nflow)} and \texttt{rres(nshg, nflow)}, and the preconditioning matrix \texttt{Bdiag(nshg, nflow, nflow)} to the GMRES solver routines, \texttt{SolGMR}, \texttt{SolGMRs} and \texttt{SolMGF} respectively. In addition the first two routines also return \texttt{EGmass(numel, ndof, ndof)}. The new arrays are:

- \texttt{res(nshg, nflow)} - is the global nonlinear residual of the algebraic equations. It is a vector containing \texttt{nshg} (number of shape functions) vectors \( R_A \) of length \texttt{nflow} (5). The residual is
the following expression:

\[
G_A \left( Y^{(i)}_{,t}, Y^{(i)} \right) = \int_{\Omega} \left[ N_A \left\{ A_0 Y^{(i)}_{,t} - F \right\} - N_{A,i} F_i \left( Y^{(i)} \right) \right] d\Omega \\
+ \sum_{\varepsilon=1}^{n_\varepsilon} \int_{\Omega^\varepsilon} L_{\text{adv}} N_A \tau \left\{ A_0 Y^{(i)}_{,t} + A_i Y^{(i)} - F \right\} d\Omega^\varepsilon \\
+ \int_{\Gamma} N_{A,i} F_i n_i d\Gamma
\]

\[r_{\text{res}}(nshg, nflow)\] is the global nonlinear modified residual used when using the Matrix-Free GMRES solver. Instead of evaluating the residual at \( Y \) it is evaluated at \( Y \) plus a perturbation times the Krylov vector.

\[B_{\text{diag}}(nshg, nflow, nflow)\] is the preconditioning matrix which is used for the block diagonal scaling of the tangent matrix, of the unknowns and of the residual. It is not used when Matrix-Free GMRES solver is used. This matrix contains on the diagonal only \( M_{AA} \) tangent matrices.

\[E_{\text{mass}}(n\text{elem}, n\text{dof}, n\text{dof})\] is the tangent matrix. For each element it has \( n\text{dof} = n\text{en} \times n\text{dof} \) entries, i.e. for each nodal point there is \( n\text{dof} \) degrees of freedom. The tangent matrix is:

\[
M_{ab}^{\varepsilon} = \sum_{b=1}^{n_m} \frac{4_{LHS}}{Y_{\beta}^{(i)}} \frac{\partial G_a}{\partial Y_{\beta}^{(i)}}
\]

where

\[
4_{LHS} G_A \left( Y^{(i)}_{,t}, Y^{(i)} \right) = \int_{\Omega} \left[ N_A \left\{ A_0 Y^{(i)}_{,t} - F + A_i Y^{(i)} - N_{A,i} K_{ij} Y^{(i)} Y^{(i)} \right\} \right] d\Omega \\
+ \sum_{\varepsilon=1}^{n_\varepsilon} \int_{\Omega^\varepsilon} L_{\text{adv}} N_A \tau \left\{ A_0 Y^{(i)}_{,t} + A_i Y^{(i)} - F \right\} d\Omega^\varepsilon
\]

These routines first loop over element blocks to calculate the diffusion correction for stabilization by calling AsIq. Then the residuals and tangent matrix are calculated by looping over interior element blocks in AsIQR and AsIQRs, and AsIMFG for ElmMFG. For the first two routines the boundary condition for LHS need to be satisfied before exiting the loop which is done in bc3LHS. Then the contribution to the residuals by the boundary integral is calculated in As2MF inside the loop over the boundary element blocks. Before passing the residuals to solver routines, the boundary conditions are satisfied for \( \text{res} \) and \( r_{\text{res}} \) by calling bc3Res and for \( B_{\text{diag}} \) by calling bc3Bdg. The satisfaction of boundary condition for the modified residual \( r_{\text{res}} \) is only done in ElmMFG.

3.5.2 AsIxxx

The subroutines AsIQR and AsIMFG compute local residuals, rl(npro, nshl, nflow), local mass matrix E3mass and block diagonal preconditioner, through the e3 routine and assemble them to form residual and Bdiag which are returned to the ElmXXX subroutines. Here E3mass spans only over elements of the current block. AsIMFG also assembles the modified residual rml(npro, nshl, nflow) calculated by e3 into rres needed for the Matrix-Free GMRES solver. Both routines have as inputs primitive variables and their derivatives arrays, nodal coordinates array, element shape functions and their derivatives, element nodal connectivity array for current element block and global diffusive flux vector calculated form AsIq.

The subroutines AsIq, e3q and e3qvar are used to calculate the diffusion correction if the user asks for it in solver.inp.
3.5.3 Global Assembly Operators

There are six routines that perform global to local operations as well as the local to global. They are: \texttt{local}, \texttt{localx}, \texttt{localy}, \texttt{localM}, \texttt{localSum} and \texttt{localb}.

The \texttt{local} routine needs as input the nodal connectivity array, the number of degrees of freedom to copy, a code which tells if it is from global to local, from local to global or if local is to be added to global and of course the global or local vector and the other is then outputed. local arrays have one more argument then global ones which span over all shape functions. Local arrays span over all elements in the current block and over all local shape functions.

The \texttt{localx} routine performs the same operation as above but specifically for the nodal coordinates array. Here the local array spans over all vertices for each element of the current block.

The \texttt{localy} routine is similar to \texttt{local}, but it put on the local level the pressure in the first slot, then the velocity components and temperature stays in the last slot.

The \texttt{localM} routine assembles the global tangent matrix from element matrices. It only does local to global assembly.

The \texttt{localSum} routine also assembles only from local to global, but for vectors and keeps track of the number of local contributions for each node in \texttt{nHits}.

The \texttt{localb} routine is similar to \texttt{local}, but it is used only for boundary elements and local arrays span over local boundary shape functions.

3.5.4 e3 and e3xxx

The subroutine \texttt{e3} calculates the local residual, the modified residual, the consistent mass matrix and the block-diagonal preconditioner by calculating first the integration variables in \texttt{e3ivar} and the needed matrices in \texttt{e3matrx}. Then, the convection, diffusion, body force, least-squares and time derivative contributions are calculated in \texttt{e3conv}, \texttt{e3visc}, \texttt{e3source}, \texttt{e3LS} and \texttt{e3massr} respectively. The time contribution to the mass matrix is done in \texttt{e3mass1}. All these contributions are multiplied by corresponding shape functions or their derivatives in \texttt{e3wmlt}. The preconditioner is calculated in \texttt{e3bgd}.

The subroutine \texttt{e3ivar} returns the following variables at integration points: density, pressure, temperature, all three components of velocity (i.e. \(Y(x)\)), internal energy, enthalpy, expansivity, isothermal compressibility, specific heat at constant pressure, kinetic energy, gradient of all primitive variables in all three directions (i.e. \(Y_i(x)\)), \(\Delta U\) variables at current step, primitive acceleration variables, inverse of deformation gradient, weighted Jacobian, element global gradient shape functions (i.e. \(N_{0,i}(x)\)), divergence of diffusive flux and resolved Leonard stresses.

The subroutine \texttt{e3matrx} calculates and returns to \texttt{e3} routine \(A_0\) and \(A_4\) matrices defined in (2.1).

The subroutine \texttt{e3conv} calculates \( -P_i^{adv} \) in the 1 to 15 entries of the second argument of the partial residual \(r_i\) (\(npro, nflow\* (nsd+1)\)) matrix which is later multiplied by the appropriate shape functions and their derivatives to form \(r_i\). It also computes \(A_4 Y_i\), which are stored in \(rlyi(npro, nflow)\) and also in the 16 to 20 entries of \(rmi(npro, nflow* (nsd+1))\). For the modified residual convection was not integrated by parts, i.e. it uses the residual of 40. The matrix \(rmi\) is the modified counterpart of \(r_i\). The matrix \(rlyi\) is the partial residual for the stabilization part which needs to be multiplied by \(\tau\)-matrix. Finally this routine calculates \(N_i A_i N_{k,i}\) and stores them in \texttt{e3mass}.

The subroutine \texttt{e3visc} calculates \(K_{ij}\) which are stored in \texttt{stiff} and \(P_i^{diff}\) stored in 1 to 15 of \(r_i\) and \(rmi\).

The subroutine \texttt{e3source} calculates the source contribution to the 5 last entries of \(r_i\) and \(rmi\), and to \(rlyi\).

The subroutine \texttt{e3LS} calculates the stabilization contributions to the residuals. First it adds \(A_0 Y_i\) to \(rlyi\). Then the tau matrix \(\tau\) is calculated by \texttt{e3tau}. In this subroutine \(rlyi\) is multiplied by \(\tau\) and stored into itself, also the modified counterpart is created \(rLyni\). They are then multiplied by \(A_j\) and added to \(r_i\) and \(rmi\) respectively in entries 1 to 15. It adds \(A_i \tau A_j\) into \texttt{stiff}. Finally it adds \(N_{0,i} A_i \tau A_0 N_{j}\) multiplied by the Jacobian and the factor of the generalized \(\alpha\) method into.
The subroutines e3dc and e3cool add the contributions to the residuals due to the discontinuity capturing operator and cooling factor respectively.

The subroutine e3masr adds to ri and rmi the contribution due to the time derivative $A_0 Y_t$ into the entries 16 to 20. For the modified partial residual the contribution is multiplied by the factor $c$ defined in (41).

The subroutine e3mass1 adds to EGmass the contribution due to the time derivative $N_e A_0 N_0 c$.

The subroutine e3bdg calculates the preconditioner all at once.

The subroutine e3wmt multiplies the corresponding entries of ri and rmi with the shape functions and their derivatives as well as with the weighted Jacobian to form ri and rmi. The first five entries correspond to the $x$ derivative of the shape functions, the next five to $y$ derivative, then to $z$ derivative and the last five entries correspond to the shape functions themselves. It also multiplies stiff with the pair of corresponding shape functions and their derivatives as well as with the weighted Jacobian and adds it to EGmass.

3.6 Boundary Condition Prescription

3.6.1 Essential Boundary Condition

The subroutine itrBC adjusts $Y$ array for essential boundary conditions as follows:

$$Y = \begin{bmatrix}
Y_1 \\
Y_2 \\
Y_3 \\
Y_4 \\
Y_5 \\
Y_6
\end{bmatrix} = \begin{bmatrix}
u_1 - C_1 Y_1 - C_2 Y_2 \\
u_1 - C_1 Y_1 - C_2 Y_2 \\
u_1 - C_1 Y_1 - C_2 Y_2 \\
u_1 - C_1 Y_1 - C_2 Y_2 \\
p^a \\
T^a
\end{bmatrix} \text{ on } \Gamma_g^n$$

Depending on how many components of velocity are set BC(., 3-5) have combinations of $u_1^a$, $u_2^a$, $u_3^a$, $C_1^a$, $C_2^a$ and $C_3^a$ which were performed in genBC1. BC(., 1) contains pressure or density (but not both) and BC(., 2) the temperature.

Note: there is an inconsistency in this routine because the density boundary condition is put in the first slot of the $Y$ vector where we have the first component of velocity instead of in the forth place as it is done for the pressure.

The bc3LHS routine adjusts the tangent mass matrix for essential boundary condition on element level as follows: $S^m_t M_{ab} S_t$ where $S_t$ is the linear transformation of the weight space to account for the boundary conditions. For example if $u_2^a$ is set:

$$S_{uv} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -\frac{C_2}{C_1} & -\frac{C_2}{C_1} & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}$$

When doing the multiplication zeros on the diagonal (here $S_{22}$) is set to one so that there isn't a division by zero later.

The subroutine bc3per adjusts the global residuals for periodic boundary conditions, by adding the residual of the master to the periodic slave and zeroing the master's.

The subroutine bc3Res adjusts the global residuals for essential boundary conditions doing $S^m_t G_t$. It adjusts it also for periodic boundary conditions as in bc3per and for SPEBC by zeroing the residuals on the inlet plane.

The subroutine bc3Bdg adjusts the block diagonal preconditioner for essential boundary conditions similarly to bc3LHS.
3.6.2 Boundary Element Integrals

The subroutine AsBMFG computes local residuals, r1 and rm1, through the routine e3b and assembles them to form res and rmea which are returned to the subroutine EImMFG. It has as inputs primitive variable array, nodal coordinates array, boundary element shape functions and their derivatives, boundary element nodal connectivity array for current boundary element block and iBCB and BCB arrays.

The subroutine e3b calculates the element residual r1 and the element modified residual rm1 and outputs them to AsBMFG. This routine receives local Y variables, y1 and yc1, local nodal coordinates x1, boundary element shape functions and their derivatives and iBCB and BCB arrays containing the boundary condition codes and values respectively. It calls e3bvar to calculate all necessary integration variables. In this subroutine the boundary integral of equation (15) is evaluated:

\[
\int_{\Gamma} N_b F \, n_i \, D \, d\Gamma = \int_{\Gamma} N_b \left( u_n U + p \begin{bmatrix} 0 \\ \delta_{jn} \\ u_m \\ \tau_{jn} \\ \tau_m u_i \\ \tau_n \\ q_n \end{bmatrix} \right) + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \, D \, d\Gamma \tag{44}
\]

The expression in the parentheses is calculated and stored in F1, F2, F3, F4 and F5 for the five equations. If a natural boundary condition was prescribed at some node its prescribed value at integration points is used already calculated by e3bvar, if not the expressions are calculated by Y variables. The residual which is the integral in (44) is evaluated by looping over integration points and adding the value already stored of r1 the corresponding Fi multiplied by the shape function and the determinant of the Jacobian, both, evaluated at that integration point. For the modified local residuals only the viscous part of Fi is used. In this routine the aerodynamic forces acting on the body are also evaluated.

The subroutine e3bvar computes all necessary variables at integration points for e3b. These variables are: density, pressure, temperature, all three components of velocity, internal energy, kinetic energy, velocity gradients in all three directions, the specific energy at constant pressure, the weighted Jacobian for boundary integrals and the components of the outward normal vector. First the primitive variables are calculated at the integration points, \( \xi^i \):

\[
Y = \sum_{a=1}^{n_{in}} N_a(\xi^i) Y^a \tag{45}
\]

The thermodynamic properties are then calculated by getthm which are used to calculate other variables stated above. It also outputs the natural boundary conditions at integration points calculated from BCB array: pressure, mass flux, viscous fluxes and heat flux.

3.7 SPEBC

3.7.1 setSPEBC

This subroutine allocates the following arrays used in renum and genscale:

- yin(nfath) - the y coordinates for the fathers of the inlet plane
- yint(nfath) - the y coordinates for the fathers of the recycle plane
- ienson1(nfath, nsomax+1) - sort the nodes on inlet plane by decreasing z coordinate for each y coordinate (i.e. father)
- ienson2(nfath, nsomax+1) - same as above but for the recycle plane
- newmap(nump) - rearrange the ordering such that first are nodes from the inlet plane, then nodes from the recycle plane and then all other nodes
3.7.2 renum

This routine makes a map which arranges the nodes by having the nodes on the inlet plane first, then those on the recycle plane and finally all other nodes. The newmap array keeps a map between the global node ordering and this new ordering. This is accomplished by first finding the nodes that are on the inlet plane and placing them in nen1 and those of the interior plane which are placed in nen2. There are num1 and num2 nodes on inlet and recycle planes respectively. At this point the global number of all nodes that are not on inlet or recycle planes are placed in newmap starting at num1 + num2 + 1. For both lists of nodes, new lists are made to find unique y values and put them in y1un and y2un depending on the plane. These values are then reordered from smallest to largest y value. Finally all the nodes of the inlet and internal planes are stored in ienon1 and ienon2 respectively by first decreasing z coordinate for each y coordinate which is increasing.

3.7.3 genscale

This subroutine generates the inflow velocity from a recycle plane for a boundary layer. It needs as inputs the nodal coordinates array, x, the boundary condition code array, iBC, the boundary condition data array, BC and the Y variables array, y which enters preteening to the inlet plane are then modified.

The SPEBC is used in the code if in solver.inp irscale is set to be positive. This parameter also indicates if the laminar (or Blasius)- or turbulent rescaling is used, i.e. if irscale is less then 10 then it is the turbulent scaling that is used. Only these two possibilities are implemented in the code.

First, the newmap array is used to create ymapped and ifatz for easier use. It also rearrange the numbering in ymapped such that the pressure is in the first slot (as in the incompressible code) for the compressible code.

Then the boundary layer thickness at the interior plane needs to be found. It is done by finding the father that has a velbar of 0.99Uinf and taking its y coordinate to be δrcy. This thickness is smoothen for early transients and bounded by half of the boundary layer thickness at the inlet plane, δint as lower bound and twice of δint as the upper bound. The boundary layer thickness at the inflow is set by the user in solver.inp and is rbltin in the code. Both rescaling possibilities use η as similarity variables. So at this point in the routine etain and etaint are calculated for each father of the inlet and interior planes respectively (21).

For the turbulent rescaling, first the momentum thicknesses at the inlet and recycle planes are calculated to compute urinl by (31). The friction velocity, urc, is given by (19). With urc's known γ can be computed and is stored in gamt. The fluctuations for the interior station are calculated and stored in fluc. The wall coordinates are stored in yplusi and ypluso for inlet and recycle planes respectively. Urin is stored in ubarint. Then first the inner layer scaling is performed where ubar1 and fluc1 are calculated, they are Uinouter and (u')inouter respectively. The outer layer scaling gives ubar2 and fluc2 (Uouter and (u')outer respectively). Some smoothing option exists for early transients. The weighting function is then calculated by (30). Lastly ymapped is updated for the inflow calculations using equation (22).

Similar procedure is done if Blasius scaling is used. Only ubar2 and fluc2 are computed which then gives updated ymapped for the inlet plane. Finally for both scalings y is updated by ymapped only for the inlet plane.

Note that for now it is assumed that there is the same number of points in y at the recycle station as at the inflow which restrains the meshes that can be used for SPEBC. This is one of the improvements that will be done in this project.

3.8 Linear Equation Solvers

3.8.1 Element By Element GMRES

Figure 5 shows the structure of this solver. When element by element (EBE) GMRES solver is used itrdriv calls SolGMRE. This routine outputs the preconditioned residual, the block-diagonal precon-
dلونer and the computed solution $\Delta Y^{(i)}$ which is stored in $Dy(nshg, nflow)$. The important arrays in this routine are:

- $uBrg(nshg, nflow, Kspace+1)$ - is the Krylov vector;
- $HBrg(Kspace+1, Kspace)$ - is the upper Hessenberg matrix;
- $eBrg(Kspace+1)$ - is the error of the Hessenberg minimization problem;
- $yBrg(Kspace)$ - is the solution of the Hessenberg minimization problem which is related to the solution sought, $\Delta Y^{(0)}$.

First the tangent matrix, the global residual and the block-diagonal preconditioner are formed in ElmGMR, it then performs the LU decomposition of the block-diagonal preconditioner and precondition the global residual array $\text{res}$, both by calling $i3LU$. The $Dy$ array is initialized to zero. The $i3per$ routine preconditions the mass matrix, $ECmass$. This finishes the preconditioning step of the EBE GMRES solver. To start the solver:

- The first Krylov vector is defined as the global residual and is normalized by the second norm of $\text{res}$. If the norm is down to the machine precision no GMRES loops are necessary.
- The tolerance for this linear equations system is set by multiplying the tolerance set by the user with the norm of the global residual.
- Loop over GMRES cycles
  - Calculate $M_{AB} \Delta Y^{(i)}$ by calling $Au1GMR$ and adjust for periodicity ($bc3per$). Here the mass matrix is preconditioned.
  - Subtract the performed product from the preconditioned global residual to create the Krylov vector.
  - Calculate its norm and place it in $eBrg(1)$.
  - Normalize the Krylov vector by its norm which creates the first Krylov vector for this GMRES cycle.
  - Loop over GMRES iterations
    - Form next Krylov vector by performing $Ap$ product on $ECmass$ and $uBrg(., iKs+1)$ ($Au1GMR$) and adjust for periodicity ($bc3per$).
    - Orthogonalize this Krylov vector to all previous Krylov vectors and normalize it.
    - Construct the upper Hessenberg matrix $HBrg(iKs+1, iKs)$.
    - Reduce it to $eBrg(iKs+1)$.
    - Exit the loop when $eBrg(iKs+1)$ is within the tolerance calculated at the begining of this routine.
  - Solve for $yBrg(i)$ which is $eBrg(i)/HBrg(i, i)$ where $i$ is going form $iKs$ to 1.
  - Update $Dy$ by adding to it the dot product of all Krylov vectors with the corresponding $yBrg$ solution.
  - Check once more for convergence of $eBrg(iKs+1)$
- Do the back preconditioning on the obtained solution, $Dy$.
- And output the statistics by calling $rstat$.
The subroutine i3LU performs the LU decomposition of a diagonal matrix Diag if the code is
'LU_Fact'. The matrices L and U overwrite the original matrix. If the code is 'forward' the vector \( x \) is forward multiplied by the inverse of \( L \). If the code is 'backward' the vector \( x \) is forward multiplied by the inverse of \( U \). It also multiplies \( x \) by \( Diag \) and stores it in \( x \) when the code is 'product'.

The subroutine i3per preconditions EGmass by first left multiplying it by the inverse of \( L \), then right multiplying the result by the inverse of \( U \) and stores the results in EGmass.

The Au1GMR routine performs the matrix vector product for the EBE GMRES slower loop over element blocks and compute the matrix vector product at the local level and assemble the results. This part is done in asAuGMR. In this new routine first the Krylov vector is localized, then the matrix vector product is performed between this local Krylov vector and EGmass. Finally the result is globalized and passed back to Au1GMR. So in this routine the following operation is done:

\[ V_{i+1} = \bigwedge_{e=1}^{n_{el}} M_{ab}^e V_i \]  

(46)

3.8.2 Matrix-Free GMRES

The SolMFG routine is called in itrdrv if Matrix-Free GMRES solver is used (see figure 6). This routine also outputs the preconditioned residual, the block-diagonal preconditioner and the computed solution \( \Delta Y^{(i)} \) as is done in SolGMR. This solver does not need EGmass, but instead has two global residuals, res, as for EBE GMRES, and the modified residual, mres which is the residual for the perturbed \( Y \) vector. This vector is perturbed by adding to it \( \epsilon V_i \) where \( V_i \) is the Krylov vector and \( \epsilon \) is the small perturbation. For this solver the following assumption is made as \( \epsilon \) approaches zero:

\[ \frac{\partial G_A}{\partial Y_B} = \frac{G_A(Y + \epsilon V_i) - G_A(Y)}{\epsilon V_i} \]  

(47)

The algorithm is similar to what was presented for EBE GMRES solver. Instead of preconditioning EGmass, here mres is also preconditioning by calling i3LU. The \( Y^{(i)} \) is also preconditioned by multiplying it by the block diagonal matrix using the code 'product' in i3LU. to calculate the first Krylov vector of the GMRES cycle instead of calling Au1GMR, Au2MFG is called. For the second call of Au1GMR in SolGMR, i.e. inside a GMRES iteration Au1MFG is called instead of Au2MFG. The rest of SolMFG is the same as SolGMR.

The subroutine Au2MFG performs the matrix vector product for Matrix-Free GMRES and subtracts it form the residual. So it outputs the first Krylov vector of the GMRES cycle. It does that by:

- uBrg is back preconditioned.
- It is then adjusted for essential boundary conditions and SPEBC by calling itrBC and genscale.
- Then itrRes is called which calculates the modified residual, ubtmp1 by looping over interior element blocks where AsIRes is called, then looping over boundary element blocks where AsBRes is called. Finally the calculated modified residual is adjusted for the essential boundary conditions by calling bc3Res. The AsIRes routine localize necessary variables, calls e3 routine to calculate the modified residual for the current block and globalize it to return it to itrRes. The subroutine AsBRes performs the same operations but for boundary elements.

Finally this described procedure is repeated for ubtmp1 to form ubtmp2 form which the Krylov vector is deduced.

The Au1MFG routine does the same procedure as Au2MFG but only one pass is performed to compute the Krylov vector.

3.8.3 Sparse GMRES

The Sparse GMRES solves are done in SolGMRs routine. In this solver the full EGmass is stored in a sparse way by I don’t have time to go in detail how this solver work.
3.9 Output

The output is done in the routine Bflux which calls restart (3.3.6). It is called at the end of the simulation, but also during the simulation at predetermined timesteps. The user set the number of timesteps between restarts when the solution array and its derivative array are outputted in restart.###. The first number is the timestep number and the second one the processor number for parallel processing simulations. Some parameters are outputted to the screen and in histor.dat file for each timestep which is done in rstat. This routine is called at the end of SolXXX routines. The parameters that are outputted help to check convergence during a simulation.

4 Conclusion

In this report, the functioning of PHASTA was tried to be explained. Not all the routines were described and some were described in more detail than others. Anything that pertains to higher order polynomial basis functions was not looked at as well as anything that wasn’t directly connected to SPECT.

References


