VABS Manual for Developers[1]

(VABS 3.6)

1. Introduction

VABS can be used as either a callable library or a standalone application, depending on your specific need. The program flow of both applications is the same and is illustrated in Figure 1.

The yellow boxes are subroutines handling I/O needed for VABS. First VABS needs to read the cross-sectional model which is generated by a preprocessor (the first yellow box). The inputs should include problem control parameters, mesh control parameters, nodal coordinates, elemental connectivity, layup information, and material properties. If it is for recovery, VABS should also read the one-dimensional global behavior calculated by the global beam analysis and the warping functions and constitutive models calculated in the constitutive modeling. The outputs of the constitutive modeling include mass properties and stiffness properties and corresponding warping functions. The outputs of the recovery include the three-dimensional displacements, three-dimensional stresses, and three-dimensional strains. The recovered displacements are given for each node and expressed in the beam coordinate system. The recovered stresses and strains are given for each node and each Gaussian point, and are expressed in both the beam coordinate system and the material coordinate system.

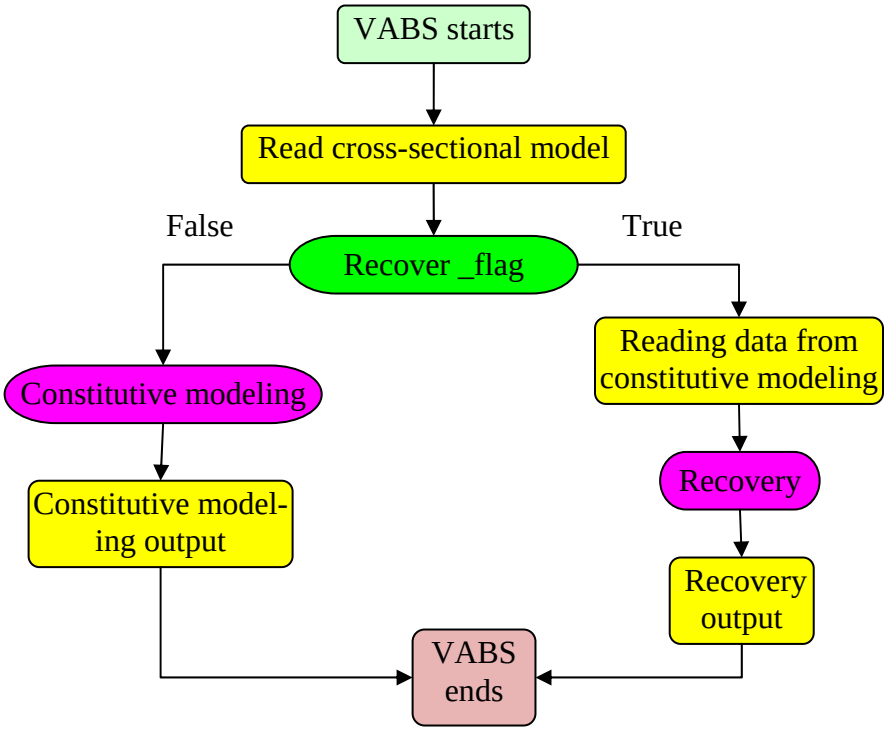

Figure 1. Modified VABS code structure

[1] You should read VABS users manual first before you read this manual.

The red boxes including the constitutive modeling and the recovery are handled by two separated Dynamic Link Libraries (DLLs) called Constitutive.dll and Recovery.dll respectively. These two DLLs contain all analysis capabilities of VABS. For design environment developments, these two DLLs like two plug-n-play black boxes. What the developers need to provide are Interfaces so that the two DLLS can be called and communicated with outside environment.

2. Global Variables needed for VABS

GlobalDataFun.f90 defines the global variables for VABS, although they are not passed to/from the two DLLs. They are necessary for defining the variables passing to/from the two DLLs. These variables are

- *allo_stat*: an integer variable to indicate status of allocating memory
- *in_stat*: an integer variable to indicate status of I/O process
- *DBL*: an integer constant to indicate how many digits real numbers should be used. For double precision DBL=8 for single precision DBL=4
- *TOLERANCE*: a real constant to simulate a small number
- *PI*: a real constant for π
- *DEG_2_RAD*: a real constant for $\pi/180$
- *NDIM*, an integer constant equal to 2
- *MAX_NODE_ELEM*, an integer constant equal to 9
- *NSTR_3D*, an integer constant equal to 6
- *NDOF_NODE*, an integer constant equal to 3
- *MAXDOF_EL*, an integer constant equal to 27
- *NE_1D*, an integer constant equal to 4

3. I/O Variables for Constitutive.dll

To take advantage of the constitutive modeling, one needs to call Constitutive.dll, which implies the right arguments should pass to and from this DLL. The DLL is invoked as follows (in the format of Fortran 90/95):

```
CALL ConstitutiveModeling(inp_name, format_I,mat_type_layer,layup_angle,LAY_CONST, &
                          nlayer,Timoshenko_I, curved_I, oblique_I, trapeze_I, Vlasov_I, kb, beta, &
                      nnode, nelem, nmate,  coord, element, layup, mat_type, material, orth, density,&
                        mass, xm2, xm3, mass_mc, I22, I33, mass_angle, Xg2, Xg3,Aee, Aee_F, Xe2, &
                         Xe3, Aee_k, Aee_k_F, Xe2_k, Xe3_k, ST, ST_F, Sc1, Sc2, stiff_val,    &
                       stiff_val_F, Ag1, Bk1, Ck2, Dk3, thermal_I,cte, temperature,NT,NT_F,error)
```

with the following variables passed to Constitutive.dll

- *inp_name* is a character of length 256 denoting the input file name for VABS.
- *format_I* is a integer indicating the input format with value equal to 1 or others.

- *mat_type_layer* is a integer array with dimension as *nlayer*, indicating the material type for each layer.
- *layup_angle* is a real array with dimension *nlayer*, indicating the layup orientation for each layer.
- *LAY_CONST* is an integer equal to 1 for new format or equal to 10 for old format.
- *nlayer* is an integer denoting the number of layers.
- *Timoshenko_I, curved_I, oblique_I, trapeze_I, Vlasov_I,* are five integer variables with values being 0 or 1.
- *kb* is a 1D real array with dimension as 3 holding three numbers for initial twist (k_1) and initial curvatures (k_2, k_3).
- *beta* is a 1D real array with dimension as 3 holding three numbers including the two oblique parameters plus a zero at the 3rd position.
- *nnode* is an integer number for total number of nodes.
- *nelem* is an integer number for total number of element.
- *nmate* is an integer number for total number of material types.
- *coord* is a 2D real array with dimension as (*nnode, NDIM*). The x_2 and x_3 coordinates of the *i*th node are stored in coord(i,1) and coord(i,2), respectively. Note these values are changed after execution due to optimizing the input mesh.
- *element* is a 2D integer array with dimension as (*nelem, MAX_NODE_ELEM*) holding the ten integers needed for elemental connectivity. Note these values are changed after execution due to optimizing the input mesh.
- *layup* is a 2D real array with dimension as (*nelem, LAY_CONST*) holding the ten real numbers including one number for θ_3 and nine numbers for θ_1.
- *mat_type* is a 1D integer array with dimension as *nelem* holding the material type (or layer type for new format) for each element.
- *material* is a 2D real array with dimension as (*nmate,21*) holding up to 21 real numbers for the elastic constants of each material.
- *orth* is a 1D integer array with dimension as *nmate* holding an integer to indicate the anisotropy of the material.
- *density* is a 1D real array with dimension as *nmate* holding density for each material.
- *thermal_ I* is an integer variable with values being 0 or 1.
- *cte* is a 2D real array with dimension as (*nmate,6*) holding the coefficients of thermal expansion.
- *temperature* is a 1D real array with dimension as *nnode* holding the nodal temperature.

The following variables are passed from Constitutive.dll

- *mass* is a 2D real array with dimension as (6,*6*) storing the mass matrix.
- *xm2,xm3* are two real numbers for the mass center location.
- *mass_mc* is a 2D real array with dimension as (6,*6*) storing the mass matrix at the mass center.
- *I22, I33* are two real numbers for mass moments of inertia about x2 and x3,

respectively.

- *mass_angle* is a real number for the angle of principal inertial axes in degrees
- *Xg2, Xg3* are two real numbers for the geometry center location.
- *Aee, Aee_F* are two real arrays with dimension as (*NE_1D,NE_1D*) storing the zeroth-order stiffness matrix and flexibility matrix, respectively.
- *Xe2, Xe3* are two real numbers for the tension center location.
- *Aee_k, Aee_F_k* are two real arrays with dimension as (*NE_1D,NE_1D*) storing the stiffness matrix and flexibility matrix due to correction of initial twist/curvature, respectively.
- *Xe2_k, Xe3_k* are two real numbers for the tension center location with corrections from initial twist/curvatures.
- *ST, ST_F* are two real arrays with dimension as (*6,6*) storing the stiffness matrix and flexibility matrix for the Timoshenko model, respectively.
- *Sc1, Sc2* are two real numbers for the shear center location.
- *stiff_val, stiff_val_F* are two real arrays with dimension as (*5,5*) storing the stiffness matrix and flexibility matrix for the Vlasov model, respectively.
- *Ag1, Bk1, Ck2, Dk3* are four real arrays with dimension as (4,*4*) storing the matrices for Trapeze effects.
- *NT* is a 1D real array with dimension as 4 storing the 1D nonmechanical stress resultants.
- *NT_F* is a 1D real array with dimension as 4 storing the 1D thermal strains.
- *error* is a character variable with length 300 to store the error message of the program.

4. I/O Variables for Recovery.dll

To use the recovery capability of VABS, one needs to call Recovery.dll, which implies the right arguments should pass to and from this DLL. The DLL is invoked as follows (in the format of Fortran 90/95):

```
CALL Recovery(inp_name, format_I,mat_type_layer,layup_angle,LAY_CONST,nlayer,&
              recover_I,Timoshenko_I, curved_I, oblique_I, Vlasov_I, kb, beta,          &
              nnode, nelem, nmate, coord, element, layup, mat_type,  material, orth,density,&
               disp_1D, dir_cos_1D, strain_CL, strain_CL_1, strain_CL_2,   &
              force_1D, load_1D, load1_1D, load2_1D, Aee_F, ST_F,      &
          disp_3D_F, k_F, nd_F, ss_F, ss_nd_F,ss_elem,thermal_I,cte,temperature,NT_F,error)
```

where *inp_name, format_I, mat_type_layer, layup_angle, LAY_CONST, nlayer, recover_I, Timoshenko_I, curved_I, oblique_I, Vlasov_I, kb, beta, nnode, nelem, nmate, coord, element, layup, mat_type, material, orth, density, disp_1D, dir_cos_1D, strain_CL, strain_CL_1, strain_CL_2, force_1D, load_1D, load1_1D, load2_1D, Aee_F, ST_F, thermal_I, cte, temperature, NT_F* are inputs to this DLL. In addition to those variables common to Constitutive.dll, the additional variables are defined as follows:

- *recover_I* is an integer flag whether it is for a linear or nonlinear theory.
- *disp_1D* is a 1D real array with dimension as 3 holding three numbers for global beam displacements.
- *dir_cos_1D* is a 2D real array with dimensions as (3, 3) holding the 1D direction cosine matrix.

- *strain_CL* is a 1D real array with dimension as 4 holding the four classical strain measures for Vlasov modeling. For other modeling, they are zeroes.
- *strain_CL_1* is a 1D real array with dimension as 4 with strain_CL_1(2) holding the first derivative of the twist rate and all others are zeros. For other modeling, they are zeroes.
- *strain_CL_2* is a 1D real array with dimension as 4 with strain_CL_2(2) holding the second derivative of the twist rate and all others are zeros. For other modeling, they are zeroes.
- *force_1D* is a 1D real array with dimension as 6, holding the 1D stress resultants with force_1D(1)=F1, force_1D(2)=F2, force_1D(3)=F3, force_1D(4)=M1, force_1D(5)=M2, force_1D(6)=M3.
- *load_1D* is a 1D real array with dimension as 6, holding the 1D distributed loads
- *load1_1D* is a 1D real array with dimension as 6, holding the first derivatives of 1D distributed loads
- *load2_1D* is a 1D real array with dimension as 6, holding the second derivatives of 1D distributed loads

The following variables are passed from Recover.dll

- *disp_3D_F* is a 2D real array with dimension as (*nnode,5*) storing the recovered 3D displacements.
- *k_F* is an integer number used to indicate the total number of Gaussian points we have recovered the 3D stresses and strains
- *nd_F* is an integer number used to indicate the total number of nodes we have recovered the 3D stresses and strains
- *ss_F* is a 2D real array with dimension as (*nelem*MAX_NODE_ELEM,26*) storing the recovered 3D stresses and strains at Gaussian points, where ss_F(:,1:2) denotes the position, ss_F(:,3:8) denotes the strains in beam coordinate system, ss_F(:,9:14) denotes the stresses in beam coordinate system, ss_F(:,15:20) denotes the strains in material coordinate system, ss_F(:,21:26) denotes the stresses in material coordinate system. Only the first *k_F* rows are meaningful.
- *ss_nd_F* is a 2D real array with dimension as (*nelem*MAX_NODE_ELEM,26*) storing the recovered 3D stresses and strains at nodal points, where ss_nd_F(:,1:2) denotes the position, ss_nd_F (:,3:8) denotes the strains in beam coordinate system, ss_nd_F (:,9:14) denotes the stresses in beam coordinate system, ss_nd_F(:,15:20) denotes the strains in material coordinate system, ss_nd_F(:,21:26) denotes the stresses in material coordinate system. Only the first *nd_F* rows are meaningful.
- *ss_elem* is a 2D real array with dimension as (*nelem,24*) storing the average of recovered 3D stresses and strains of all the Gaussian points within one element, where ss_elem(:,1:6) denotes the strains in beam coordinate system, ss_elem (:,7:12) denotes the stresses in beam coordinate system, ss_elem(:,13:18) denotes the strains in material coordinate system, ss_elem(:,19:24) denotes the stresses in material coordinate system. This array can be used to facilitate contour plot for visualization.

5. Standalone Application

The standard release includes the standalone application including Constitutive.dll and Recovery.dll and the files needed to compile the standalone application including CPUtime.f90, main.f90, VABSIO.f90, and GlobalDataFun.f90. Interfaces are provided in main.f90 so that the two DLLs can be called properly. VABSIO.f90 defines/inputs/outputs all the arguments needed to pass to/from the two DLLs. GlobalDataFun.f90 defines some global constants and functions needed for VABSIO.f90. If you are familiar with Fortran language, these files might be able to facilitate your development. The developers are also free to modify the source codes to add more capabilities which are design to take advantage of VABS incarnated in Constitutive.dll and Recovery.dll.

If the variables as explained previously are defined correctly, the remaining task for the developer to integrate VABS is to provide the interface necessary for calling the two DLLS. For example, for Fortran 90/95, the two needed interfaces are provided in main.f90.