White Paper:
Customizing HyperSizer with User-Defined Plug-Ins

Phil Yarrington, Craig Collier, Mark Pickenheim
Collier Research Corporation
December 2001

Introduction

Companies often develop engineering analysis programs to solve particular problems, but many times these codes never get past the level of "research". These codes are often shelved due to their difficulty of use, cumbersome ASCII data input and output, and lack of expert users. HyperSizer now provides an engineering environment where user developed or proprietary analyses codes can be efficiently plugged directly into the HyperSizer analysis and optimization process. HyperSizer Plug-Ins breathe new life into legacy analysis programs, and engineers need only learn one interface that is all encompassing for automated structural sizing, detailed stress analysis, and report generation.

HyperSizer uses a Plug-In *standard* that defines categories of analysis methods such as composite strength, panel/beam buckling, local buckling, bolted joint analysis, etc. and provides an interface with typical data required for each analysis. For example, bolted joint analysis requires hole diameter, concentrated bolt load, loading angle, etc. In addition, the plug-in standard is flexible in permitting additional, unplanned unique data to be passed between the plug-in code and HyperSizer. These additional variables are entered into the HyperSizer GUI and stored in the HyperSizer database. Once plugged in, the user-defined analysis automatically becomes part of HyperSizer's optimization. Finally, margin of safety results generated by the plug-in are passed back to HyperSizer and displayed in the HyperSizer GUI with all other margins of safety.

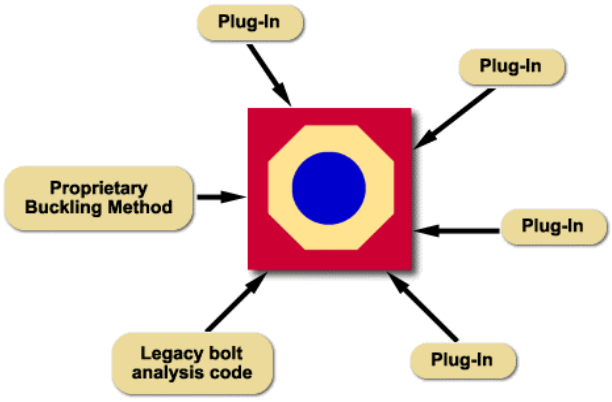

Figure 1: Plug-Ins enable customization of HyperSizer analyses by including legacy and proprietary methods

Two industry standard legacy codes were integrated with HyperSizer using the HyperSizer plug-in capability and are distributed as part of the software package. Those codes are BJSFM, a bolted joint analysis program, and SS8, a Ritz panel buckling analysis tool that analyzes curved panels and allows for general (free, fixed, or pinned) panel boundary conditions.

Overview

Test data validation and years of use make standalone legacy codes invaluable to industry. As such, they are necessary building blocks for design synthesis. For this reason, our objective is to include them as an intrinsic part of our core software whenever possible. One possibility is for companies to contract with Collier Research to incorporate these methods into the core of our software. It is understandable, however, that in many cases companies are unwilling to release proprietary source code.

Therefore, the Plug-In capability offers an alternative that allows each company's methods developers to plug-in their own user defined analysis methods directly into HyperSizer. By compiling Fortran, C, or C++ analysis routines into a dynamic link library (DLL), legacy and specialty analysis software are integrated directly. This coupling is possible with no performance slow-down to the original codes, and no intermediate ASCII files. The plugged-in software computes as efficiently as HyperSizer's built-in analyses. The capability offers users three benefits.

Benefit One
Plug-Ins provide a data handling framework, specifically for structural analysis methods, that incorporates legacy, standalone codes and brings them new life. This framework includes a user-friendly GUI that provides productivity and a relational database management system that provides data integrity. The legacy code automatically shares all material and composite data, is included in all FEA multiple design-to load cases, is executed for all structural assemblies, groups, and components, and becomes part of the optimization. Results from the analysis are maintained in the database with all other project data, can be plotted graphically, and are included in HTML stress reports.

Benefit Two
Plug-Ins provide the ability to customize HyperSizer while maintaining secure proprietary source code. Software is coupled at the user's sight by using the published interface protocols for each analysis type. Each company maintains its own codes, which protects the competitive edge they provide to the company.

Benefit Three
Finally, Plug-Ins provide an environment for developing, compiling, and testing/debugging of new analytical capability. By piggy-backing off of HyperSizer's infrastructure for controlling data flow, researchers can focus efforts on their methods development. In this way it becomes an ideal tool for university or government research.

In addition to numerical results, the HyperSizer interface protocols include graphics information. Analysis results from plugged-in software can be displayed in the HyperSizer GUI as the analysis is executed. The HyperSizer snapshots shown in Figure 2

show displacement fields predicted by the BJSFM bolted composite program including the effects of far field stress resultants on the loaded hole(left) and the buckling mode shape of a Ritz cylindrical panel buckling energy solution (right). In both cases these analyses were implemented as customer legacy codes and automatically became part of the design optimization process.

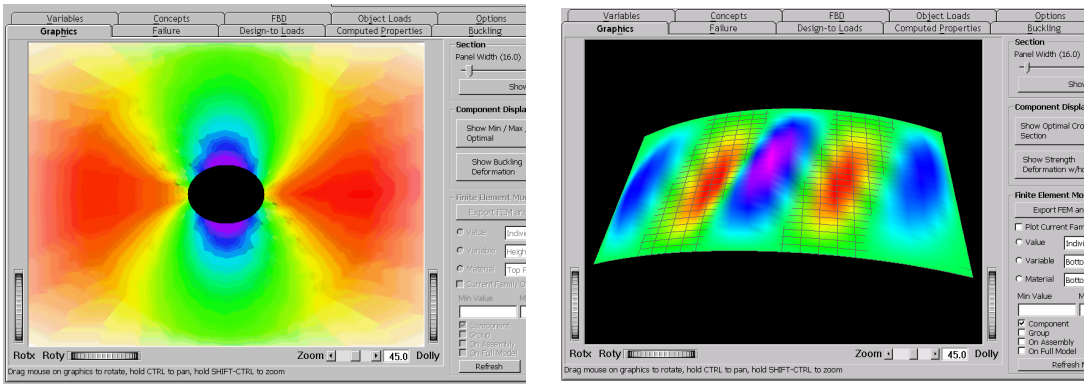

Figure 2: Graphical display of data generated by Plug-Ins and displayed in the HyperSizer interface

The Plug-In capability currently allows customization of the following analysis methods in HyperSizer:

Beam Buckling – Failure methods associated with overall beam buckling of concepts such as I-Beam, C-Beam, etc.

Bolted Hole – Failure methods associated with open holes and bolt loaded hole analysis. These analyses include holes with concentrated forces at arbitrary angles and far-field loading effects.

Composite Strength – Failure methods associated with composite failure theory. Examples include Tsai-Hill, Tsai-Wu, etc.

Crippling – Failure methods associated with panel and beam crippling. User-defined crippling failure methods can be used to enter proprietary crippling log-log curves.

Panel Buckling – Failure methods associated with overall panel buckling of concepts such as hat-stiffened, honeycomb core, isogrid stiffened, etc.

Sandwich Core – Failure methods associated with honeycomb or foam sandwich cores. Core crushing, shear crimping and shear strength are examples of sandwich core failure modes.

Sandwich Face – Failure methods associated with honeycomb or foam sandwich facesheets. Examples include wrinkling or intra-cell dimpling.

In addition to these failure methods, additional methods can be implemented as needed. For example, in the near future vibroacoustics, durability and damage tolerance (D&DT) and reliability methods will be implemented as part of the HyperSizer analysis and optimization. These methods may be perfect candidates as additional Plug-In analysis types.

Description

The process of implementing a new analysis method as a HyperSizer Plug-In consists of three steps:

Step 1: Determine the analysis category of the new analysis

The first step for implementing a new HyperSizer Plug-In is to categorize the method into one of the existing HyperSizer Plug-In analysis categories as described above. In most cases, this will be straightforward. To include a new code to be called from HyperSizer, the developer should study the interface protocols available for each analysis type and choose the one appropriate for the new method. If it turns out that the method does not fit into one of these categories, then the next step would be to contact Collier Research regarding the possibility of adding a new failure category to HyperSizer.

Step 2: Develop an interface in HyperSizer's user-defined format

HyperSizer interfaces with Plug-Ins through a user-compiled Dynamically Linked Library (DLL) file called "Hs_UDef.DLL". The Hs_UDef library has a specific interface for each analysis type and these are called at specific points during HyperSizer's execution. The interface between the HyperSizer core software and each analysis type is fixed. The code within the DLL, however can be customized in any way by adding user code or calling other legacy analysis codes. As illustrated in Figure 3, HyperSizer passes data specific to each analysis type to the user-defined DLL which then executes its own analysis. When complete, the DLL passes margin of safety data back to the HyperSizer interface.

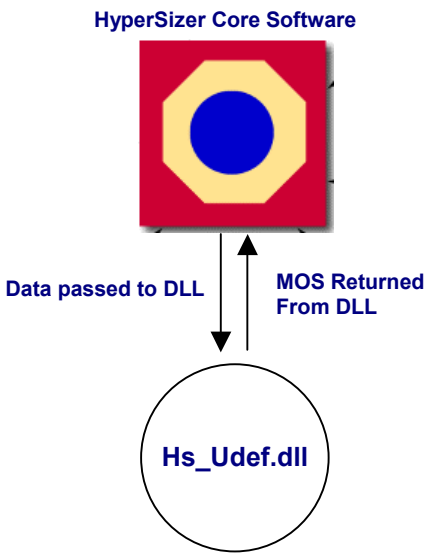

Figure 3: HyperSizer's interface to the user-defined dynamically linked library

Hs_UDef can be built and compiled using any programming language that can build a Windows DLL, including Fortran, C or C++. DLLs have been compiled and successfully integrated with HyperSizer using Microsoft Fortran PowerStation, Compaq Visual Fortran Version 6, and Microsoft Visual C++ Version 6. It is even possible to mix programming languages within the DLL such that some routines are built using Fortran and some using C or C++.

Rules-of-thumb for Plug-In development:

- Verify input data against validated ranges for the software
- Minimize file I/O of the analysis
- Trap potential errors and report them to HyperSizer

Several rules-of-thumb and utility routines provided as part of our distributed DLL code help the development of the user-defined analysis method go smoothly. First, the input data that comes through the HyperSizer interface should be verified against the bounds identified for the legacy code's data to ensure that the data is not outside the validated range of the software. Utility routines are provided to help the user check these ranges.

Second, ASCII file input and output should be limited as much as possible during the execution of the user-defined code. As part of the HyperSizer run-stream, the user-defined method can potentially be called hundreds or thousands of times during a typical sizing, therefore it is important to minimize file I/O, which is a very slow operation.

Third, potential errors should be trapped and passed back to HyperSizer and an error reporting utility function is provided. For example, if the input data passed to the Plug-In routine exceeds the software's validated range, an error should be reported to HyperSizer. If an un-trapped error occurs during the execution of the user-defined subroutine, such as "divide by zero", HyperSizer will report the analysis being performed, the structural component being analyzed, and other useful information to help the user track down the error.

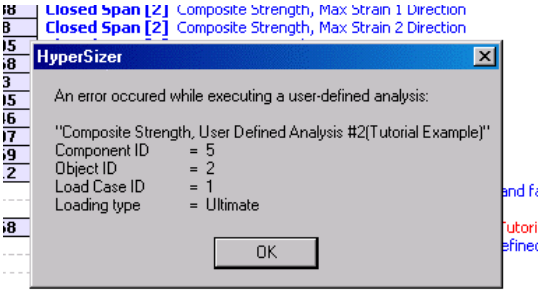

Figure 4: Un-trapped errors in user-defined methods are reported in the HyperSizer GUI

Step 3: Activate and execute the method from the HyperSizer GUI

Once the interface to the user code is built and the DLL is compiled, the final step is to activate the user-defined code in the HyperSizer GUI and then actually execute the analysis. HyperSizer treats user-defined analyses in the same way that it treats its built-in, intrinsic analyses. The user has complete control over what analyses are performed for each structural component. As shipped, user-defined analyses in the HyperSizer interface are all deactivated and have generic names such as "Composite Strength, User Defined Analysis #1". In order to give the analysis method a customized name, the user only needs to right

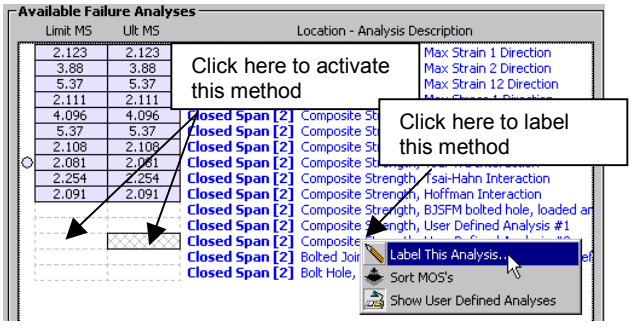

Figure 5: Labeling and activating user-defined methods in the HyperSizer GUI

click on the analysis method in the GUI to get a drop-down menu as shown in Figure 5. The assigned name then stays with this method both in the interface and in user-generated HTML stress reports. The analysis is then activated by clicking either the Limit MS or the Ultimate MS column.

Examples

Example One: User-Defined Tsai-Hill Failure Method

As an example of implementing a user defined analysis, we start with a simple implementation of the Tsai-Hill composite failure method. Tsai-Hill is also included as an intrinsic HyperSizer failure method. The failure criterion takes the form:

$$\frac{1}{\text{MOS}+1} = \sqrt{\left(\frac{\sigma_{11}}{\sigma_{11,all}}\right)^2 + \left(\frac{\sigma_{22,all}}{\sigma_{11,all}}\right)\left(\frac{\sigma_{11}}{\sigma_{11,all}}\right)\left(\frac{\sigma_{22}}{\sigma_{22,all}}\right) + \left(\frac{\sigma_{22}}{\sigma_{22,all}}\right)^2 + \left(\frac{\tau_{12}}{\tau_{12,all}}\right)^2}$$

The steps to implement this margin of safety calculation as a software Plug-In are outlined below. The examples shown below are strictly for illustrative purposes, and are not necessarily intended to be tutorials. More detailed instructions are found in the HyperSizer Programmers Guide.

Step One: Determine the analysis category of the new analysis

The Tsai-Hill margin of safety calculation uses only stresses (σ_{11}, σ_{22}, and τ_{12}) and stress allowables ($\sigma_{11,all}$ and $\sigma_{22,all}$) in its margin of safety. Therefore, this method is a perfect candidate to be plugged in as a ***Composite Strength*** method, which has inputs of stresses, strains, and stress/strain allowables.

Step Two: Develop an interface in HyperSizer's user-defined format

The name of the user-defined Composite Strength subroutine is fixed as Composite_Udef and must have the following form and arguments:

```
      REAL*8 FUNCTION Composite_Udef   (AnalysisIndex,
     2                    Strain, StrainAllow,
     3                    Stress, StressAllow,
     4                    nArrayString, ArrayString, ArrayValue)

C.. Dummy Arguments
      INTEGER AnalysisIndex,nArrayString
      REAL*8 Strain(3), StrainAllow(2,3)
      REAL*8 Stress(3), StressAllow(2,3)
      REAL*8 ArrayValue(nArrayString)
      CHARACTER*50 ArrayString(nArrayString)
```

The first argument is `AnalysisIndex`, an integer argument that links the current analysis to those defined in the GUI. For example, if "Composite Strength, User Defined Analysis #1" is turned on in the GUI, then the value of this variable will be 1. The real arrays `Stress` and `Strain` contain the real-time stress/strain state of the current analysis object. The arrays `StressAllow` and `StrainAllow` contain the stress and

strain allowable values. There are two sets of allowable stresses and strains corresponding to tension properties and compression properties respectively. The final arguments (`nArrayString, ArrayString`, and `ArrayValue`) are related to passing *user-defined constants* to the subroutine and are discussed in the second example.

While the function declaration and arguments are fixed for interface to HyperSizer, the remainder of the subroutine is completely flexible and can perform any desired operation. The first step is to determine whether the analyzed object is in tension or compression, and return the appropriate allowable properties. This is done in a subroutine called, `GetAllowables`, which returns arrays called `AllowableStress` and `AllowableStrain` that contain appropriate allowable data for compression or tension. The source of this subroutine is not listed here.

```
      CALL GetAllowables(Stress, StressAllow,
     1                   Strain, StrainAllow,
     2                   AllowableStress, AllowableStrain)
```

In the following IF block, if the value of `AnalysisIndex` is 1, then the `TsaiHill` function is called. During execution of the `TsaiHill` routine, an integer error flag is set to a non-zero value if an error occurs and this flag, along with an error string is returned to HyperSizer using the `HsReturnError` utility function. Also, if `AnalysisIndex` takes any value other than 1, an error message is returned to HyperSizer because HyperSizer is apparently trying to access an undefined analysis method (i.e. an analysis method has been activated in the GUI which has not been defined in code).

```
C.. Perform user defined analysis #1: TSAI-HILL Failure Theory
      IF (AnalysisIndex.EQ.1) THEN

C.. replace this failure criteria with one of your own
         MOS = TsaiHill (Stress, AllowableStress, iErr)

         IF (iErr.NE.0)
     1       CALL HsReturnError(iErr,'Error in User Defined '//
     2                          'Composite Failure Analysis 1')

      ELSE

C.. If the index for this analysis is out of range, return error
         CALL HsReturnError(-1, 'Undefined User Defined Analysis')

      ENDIF
```

Finally, the calculated margin of safety is returned to HyperSizer as the return value of the Composite_Udef function.

```
C.. Return margin of safety calculation
      Composite_Udef = MOS

      RETURN
      END
```

The definition of the TsaiHill function is fairly straightforward. The arguments are the `Stress` and `AllowableStress` arrays and the integer error flag, `iErr`.

```
      REAL*8 FUNCTION TsaiHill (Stress, AllowableStress, iErr)

C.. Dummy Arguments
      REAL*8 Stress(3),AllowableStress(3)
```

The rest of the subroutine performs the margin of safety calculation. Note that if the sum of stresses is equal to zero, then the margin of safety would be infinite, and a "divide by zero" error would occur. To avoid this, a margin of safety of 1000 is returned.

```
      SUM= 0.0
      DO i=1,3
         SUM= SUM + DABS(Stress(I))
      ENDDO

      IF (SUM.EQ.0.0) THEN
          TsaiHill = 1000.0
      ELSE
          X= Stress(1) / AllowableStress(1)
          Y= Stress(2) / AllowableStress(2)
          S= Stress(3) / AllowableStress(3)
          R= AllowableStress(2) / AllowableStress(1)

          if (R .le. 1.0 .and. R .ge. -1.0) then
C.. the normal 1 direction stronger than the 2 direction
              DNOM = DSQRT( (X*X) - (R*X*Y) + (Y*Y) + (S*S) )
          else
C.. the normal 2 direction stronger than the 1 direction
              DNOM = DSQRT( (X*X) - ((1.0/R)*X*Y) + (Y*Y) + (S*S) )
          endif
          TsaiHill = 1.0 / DNOM -1.0E+00
      ENDIF

C.. Set Error flag
      iErr = 0
      RETURN
      END
```

This code is then compiled as part of the Hs_UDef.DLL library, as described in the HyperSizer Programmers Guide, and is ready to be called from HyperSizer.

Step Three: Activate and execute the method from the HyperSizer GUI
The final step is to give the method a name ("User Tsai-Hill") and then activate the failure method in the HyperSizer GUI. This operation is shown in Figure 5. After executing HyperSizer, we can examine the results on the **Failure** tab of the Sizing form and compare the results from this user-defined method to the HyperSizer intrinsic Tsai-Hill method. As shown in Figure 6, the user-defined method and the intrinsic method return identical results.

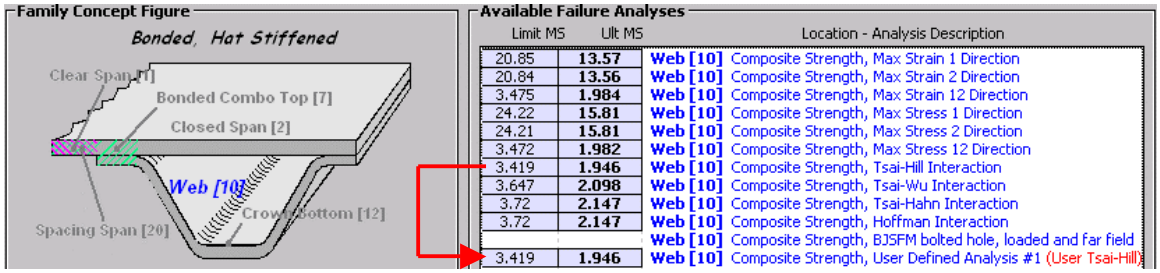

Figure 6: Comparison of user-defined Tsai-Hill margin of safety with the HyperSizer intrinsic method. As expected, the results are identical.

Example Two: User Defined Bolt Hole Analysis: BJSFM

The second example illustrates how an industry standard analysis tool, called BJSFM, is integrated with HyperSizer as a user-defined Plug-In. When implementing BJSFM as a HyperSizer Plug-In, the analysis portion of the established legacy source code is unmodified, but the input and output was modified using the above defined rules-of-thumb. The analysis type that is used by the BJSFM Plug-In is "Bolted Hole".

The standalone BJSFM code runs from the command line and uses ASCII files for input and output. The only major modification made to the BJSFM source code was in the way it handles I/O. The first change that was made was to change the *program* into a *subroutine* or *function* that accepts input values through its argument list.

```
      REAL*8 FUNCTION Hs_UDef_Bjsfm (Alength,   Blength,
     2                     Dia, HoleLoad, HoleAngle, Forces,
     3                     nArrayString, ArrayString, ArrayValue,
     3                     nObjLayup, nObjMaterial,
     3                     rObjLayup, iObjLayup, rObjMaterial)
```

Next, in order to allow the code to be run either as a plug-in or as a standalone code, logical variables were defined (`LInputFile` and `LInputFile`) that dictate whether the input and output files were generated. Then each call that would either read or write data from an ASCII file is wrapped in an IF block:

```
      IF (LInputFile) THEN
        READ  (5,  * ) ( MATID(J), ANG(J), PLYTHK(J), J = 1, NUMPLYRD )
        IF ( MATID(J) .GT. NUMMAT ) MATID(J) = NUMMAT
      ENDIF
```

The objective is to minimize file I/O by replacing READ and WRITE statements in the standalone code with I/O directly through the subroutine call if possible. Any single fastener composite bolt analysis program requires at least the following data, which is provided as standard input by the HyperSizer "Bolted Hole" analysis type.

- Plate width
- Length
- Layup
- Material properties
- Far field loading
- Hole diameter
- Bolt load
- Load direction

The data specific to bolted hole analyses is entered into the HyperSizer GUI as shown in Figure 7.

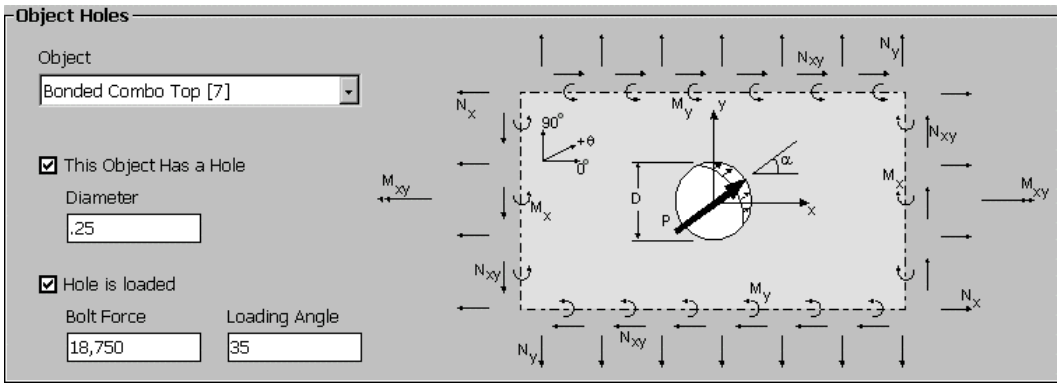

Figure 7: Bolted Hole data entry in the HyperSizer GUI

In addition to the general bolted hole data, the BJSFM program requires the following specific additional data to dictate how its analysis is executed and how its own graphical output is displayed. These additional parameters are:

- Angle Increment
- Starting Angle
- Ending Angle
- Radial Increment
- Radial Steps

These additional parameters are not applicable for all Bolted Hole analysis programs, therefore it would not be appropriate to include this data as part of the general GUI. HyperSizer allows "unplanned" data such as this to be entered into the GUI and saved in the database as ***user-defined constants***. Figure 8 shows how these constants are entered into the GUI by assigning a name (e.g. "Angle Increment") and a

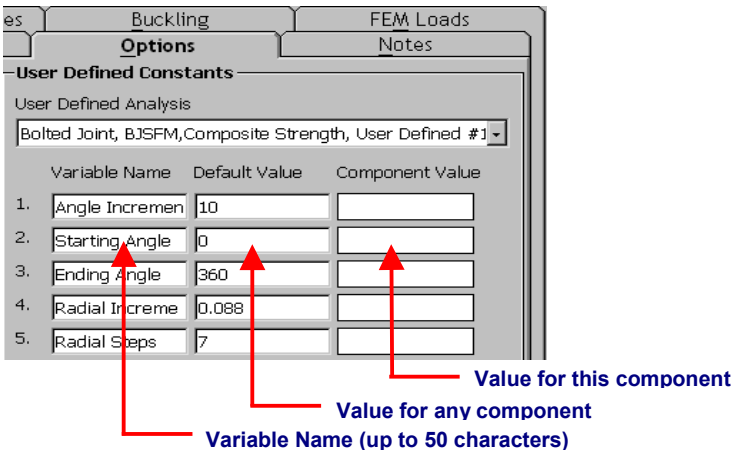

Figure 8: User-defined constants for the BJSFM program

default value to each constant. HyperSizer passes these constants to the subroutine whenever the user-defined BJSFM analysis method is executed. By default, the values passed to the subroutine come from the "Default Value" column, however, this value can be overridden for any structural component by entering a "Component Value".

Extracting a constant within the user-defined subroutine is accomplished with a set of utility functions called `GetOptionValue`. The calls to retrieve the data referred to in Figure 8 are:

```
      CALL GetOptionValue4('Angle Increment',nArrayString,
     1                     ArrayString,ArrayValue,IANG)
      CALL GetOptionValue4('Starting Angle',nArrayString,
     1                     ArrayString,ArrayValue,ILOW)
      CALL GetOptionValue4('Ending Angle',nArrayString,
     1                     ArrayString,ArrayValue,IHIGH)
      CALL GetOptionValue4('Radial Increment',nArrayString,
     1                     ArrayString,ArrayValue,STPINK)
      CALL GetOptionValueI('Radial Steps',nArrayString,
     1                     ArrayString,ArrayValue,NUMSTP)
```

After retrieving the user defined constants, they are checked against the bounds of the BJSFM program using the `VerifyValue` utility function as show below. If any of the values are outside of the program's bounds, an error is returned to HyperSizer using the `HsReturnError` function.

```
      DIAMin   =  0.01 ; DIAMax   =   1.0
      IANGMin  =  1.0  ; IANGMax  =  30.0
      ILOWMin  =  0.0  ; ILOWMax  = 359.0
      IHIGHMin =  1.0  ; IHIGHMax = 360.0

      IF (.NOT.VerifyValueI(NUMSTP, 1,    50  ) .OR.
     1    .NOT.VerifyValue4(DIA,    DIAMin,   DIAMax  ) .OR.
     2    .NOT.VerifyValue4(IANG,   IANGMin,  IANGMax ) .OR.
     3    .NOT.VerifyValue4(ILOW,   ILOWMin,  ILOWMax ) .OR.
     3    .NOT.VerifyValue4(IHIGH,  IHIGHMin, IHIGHMax ) )GOTO 1101

...

1101 CONTINUE
      CALL HsReturnError(-10, ' One or more variable values are' //
     1                        ' out of range')
```

Finally, after the BJSFM analysis is executed, the margin of safety is reported back to HyperSizer and a graphical plot of the displacement in the vicinity of the hole is displayed in the GUI as shown in the left image of Figure 2.

Example Three: User Defined Ritz Panel Buckling

As a final example of a user-defined analysis method, a Ritz energy buckling program, based on the legacy SS8 buckling code, was implemented both as a new HyperSizer intrinsic method and as a user-defined Plug-In. The Ritz solution provides additional functionality to HyperSizer's original intrinsic buckling method by allowing for curved panels and general boundary conditions (free, pinned, fixed or percent fixity) on each edge of a panel.

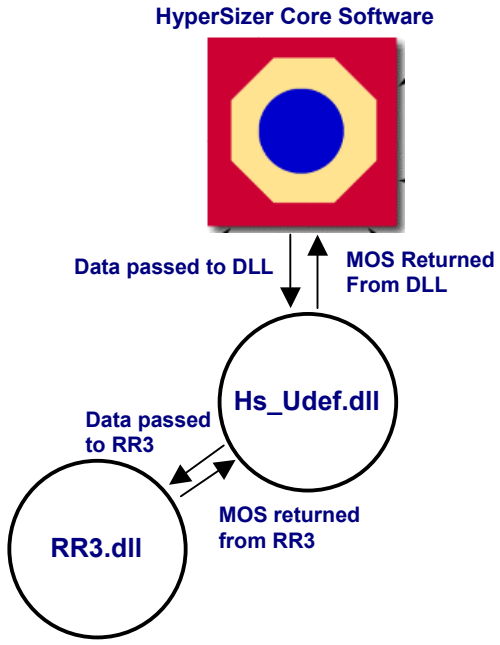

Figure 9: Data flow between HyperSizer, the user defined analysis DLL and a separate DLL for the Ritz energy buckling analysis

The Ritz buckling program further demonstrates the flexibility of HyperSizer Plug-Ins because instead of compiling with the user-defined library, Hs_UDef.dll, the code (called RR3) is actually compiled into a separate dynamically linked library file which is then called by Hs_UDef. The data flow between HyperSizer and the two DLLs is shown in Figure 9. This further illustrates that other than the interface to HyperSizer, which is rigid, the code within the user-defined subroutines can perform any desired operation, include calling other codes or libraries. In Fortran, the code for calling the Ritz code is:

```
      EXTERNAL RR3DLL
      !MS$ATTRIBUTES DLLIMPORT :: RR3DLL
      !MS$ATTRIBUTES ALIAS: 'RR3DLL' :: RR3DLL
...
      CALL RR3DLL (iErr, Forces, BoundCond, Spans, Radius,
     +                    ABDMatrix, MOS, ModeShape, UserPath)
```

The RR3DLL subroutine is declared as `EXTERNAL` which tells the compiler that the subroutine is defined outside the project, and the compiler directives `DLLIMPORT` and `ALIAS` tell the compiler to import the definition for the RR3DLL subroutine from a DLL.

Just as with the Bolted Hole program, the Ritz buckling program produces graphical output that is passed back to HyperSizer through the "ModeShape" array and is displayed in the GUI as shown in the right side image of Figure 2.