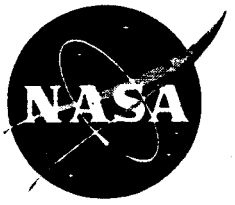
NASA/TM–1999-208781



User's Guide for ENSAERO_FE Parallel Finite Element Solver

Lloyd B. Eldred and Guru P. Guruswamy

April 1999

The NASA STI Program Office . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the Lead Center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA's counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized databases, organizing and publishing research results . . . even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at *http://www.sti.nasa.gov*

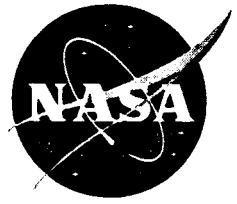
- E-mail your question via the Internet to help@sti.nasa.gov

- Fax your question to the NASA Access Help Desk at (301) 621-0134

- Telephone the NASA Access Help Desk at (301) 621-0390

- Write to:
  NASA Access Help Desk
  NASA Center for AeroSpace Information
  7121 Standard Drive
  Hanover, MD 21076-1320

NASA/TM–1999-208781



User's Guide for ENSAERO_FE Parallel Finite Element Solver

Lloyd B. Eldred and Guru P. Guruswamy
Ames Research Center, Moffett Field, California

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035-1000

April 1999

Available from:

NASA Center for AeroSpace Information
7121 Standard Drive
Hanover, MD 21076-1320
(301) 621-0390

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
(703) 487-4650

User's Guide for ENSAERO_FE Parallel Finite Element Solver

Lloyd B. Eldred and Guru P. Guruswamy
Ames Research Center

Summary

A high fidelity parallel static structural analysis capability is created and interfaced to the multi disciplinary analysis package ENSAERO_MPI of Ames Research Center. This new module replaces ENSAERO's lower fidelity simple finite element and modal modules. Full aircraft structures may be more accurately modeled using the new finite element capability. Parallel computation is performed by breaking the full structure into multiple substructures. This approach is conceptually similar to ENSAERO's multizonal fluid analysis capability. The new substructure code is used to solve the structural finite element equations for each substructure in parallel. NASTRAN/COSMIC is utilized as a front end for this code. Its full library of elements can be used to create an accurate and realistic aircraft model. It is used to create the stiffness matrices for each substructure. The new parallel code then uses an iterative preconditioned conjugate gradient method to solve the global structural equations for the substructure boundary nodes.

Introduction

Accurate structural modeling of a real aircraft by discretization has constraints that are completely independent from those faced by the aerodynamics discipline. A structural model focuses on the main internal features of the aircraft: the wing's spars and ribs and the fuselage's bulkheads and stringers. An aircraft aerodynamic model focuses on critical aerodynamic features: regions of separation, shocks, etc. These major features of interest are completely unrelated to each other.

An attempt at using common meshes is at best doomed to inefficiency, and more likely to failure. A much more efficient and powerful approach is to interface the highest fidelity single discipline technologies available. ENSAERO (refs. 1-3) implements the Reynolds averaged thin-layer Navier-Stokes equations. NASTRANs element library allows the accurate finite element modeling of the aircraft components as plates, bars, and beams and the substructure-based structural system solver allows for efficient parallel solution of the structural equations.

A fluid-structure interface calculation is performed on the aircraft skin. Fluid forces cause structural loading, causing deflection, which in turn changes the fluid field. Since the fluid surface grid does not, in general, correspond to the structural grid on the aircraft skin, an interpolation/extrapolation scheme is used to transfer the fluid loads to the structural nodes and the structural deflections to the fluid grid. This work builds on earlier domain decomposition work by Byun and Guruswamy(1-3). That work involved interfacing the Navier Stokes/Euler solver with structural models. The structural models included a modal model and a parallel finite element model, using

a partitioning approach.

Approach

A high fidelity parallel finite element capability, ENSAERO_FE has been developed. Parallel computations are performed on multiple substructures with an iterative scheme used to calculate the boundary values. A standard finite element package, in this case NASTRAN/COSMIC, is used as a preprocessor to generate the substructure stiffness matrices. The new parallel code solves the system of equations. Figure 1 shows how the structural solution is calculated in parallel.

The interactive parallel solution of the finite element system is strongly based on that proposed by Carter, et al(ref 5). It uses a preconditioned conjugate gradient method. That scheme has been adapted to run on the IBM SP-2 and the SGI Origin 2000 , Ames Research Center using the MPI for interprocess communication.

To use this scheme, the full structure is broken into substructures. The finite element stiffness matrices are assembled for each substructure, but never for the full structure. The use of connectivity information allows data to be exchanged about nodes that are shared between two or more substructures. This method has been shown to be scalable and efficient(ref. 5).

Dynamic structural analysis is performed using the Newmark (constant average accelleration) method. A linear accelleration method is also available. This method converts the dynamic system of equations into a pseudo-static system than can be solved by the same core parallel solver.

ENSAERO_.FE is dependent on an external code to generate the substructure stiffness matrices. As this is a one time operation for linear structures, there is little to be gained by parallelizing it. And there is no need to go to the effort and expense of duplicating standard codes that are readily available. In thiscase, NASTRAN/COSMIC(ref. 5)was used as a front end, although any similar code should be easily adaptable. This also allows access to the full range of standard preprocessing and CAD tools designed for NASTRAN, as well as use of the supply of existing data decks.

Once the user has used the front end program to create the substructure stiffness matrices, he or she builds input files describing the substructure connectivity and boundary conditions. Depending on the case being run, input load file may also be set up or the loads may be calculated by an attached aerodynamic code. In this case, ENSAERO_MPI(ref. 6) is used.

ENSAERO_MPI is an aeroelastic analysis package which couples the Reynolds averaged thin-layer Navier-Stokes equations with structural analysis. Its existing, limited, low fidelity simple finite element or modal structural capabilities were replaced by the new finite element code. ENSAERO's internal interpolation/extrapolation capability was adapted to exchange the aircraft aeroelastic data between the two disciplines.

The codes for the two disciplines are run in parallel. ENSAERO uses one processor per aerodynamic zone. ENSAERO_FE similarly uses one processor per substructure. The number of proces-

sors used varies substantially from problem to problem, depending on problem size and desired performance. For this work, the aerodynamic code has typically used around ten processors and the structural code around five. The MPI library is used for communication between like codes as well as across disciplines. NASA Ames' MPIRUN library is used to manage processor allocation and some communications set-up chores.

Interpolation/extrapolation of loads and deflections is performed using an intermediary interface grid. This interface grid is made of the structural skin elements (internal structural elements such as spars are ignored). These skin elements are converted to triangular interface elements for ease of interpolation. A search algorithm locates fluid grid points that fall within each triangular interface element and computes the appropriate bilinear interpolation coefficients. Figure 2 illustrates the matching of the two domain grids. Very dense fluid grids near the wing tip and at wing mid span (near a control surface) result in a large number of fluid points per interface triangle in these regions.

Validation

The code has been validated for a variety of problems ranging from simple to complex. This is discussed in reference 7. A typical converged aeroelastic result is illustrated in figure 3.

References

1. Byun, C.; and Guruswamy, G. P.: Wing-Body Aeroelasticity on Parallel Computers. *AIAA Journal*, vol. 33, no. 2, Mar.-Apr. 1996, pp. 421-428.

2. Guruswamy, G. P.: ENSAERO - A Multidisciplinary Program for Fluid/Structural Interaction Studies of Aerospace Vehicles. Computing Systems Engineering, vol. 1, nos. 2-4, 1990, pp.237-256.

3. Byun, C.; and Guruswamy, G. P.: Aeroelastic Computations on Wing-Body-Control Configurations on Parallel Computers. AIAA Paper 96-1389, April 1996.

4. NASTRAN User's Manual. NASA SP-222(08), June 1986.

5. Carter, W. T.; Sham, T. L.; and Law, K. H.: A Parallel Finite Element Method and its Protoype Implementation on a Hypercube. *Computers & Structures*, vol. 31, no. 6., 1989, pp. 921-934.

6. Byun, C.; and Guruswamy, G. P.: ENSAERO-MPI Parallel Multi-Zonal Version User's Guide. 1996.

7. Eldred, L. B.; Byun, C.; and Guruswamy, G.P.: Integration of High Fidelity Structural Analysis into Parallel Multidisciplinary Aircraft Analysis. 39th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, Long Beach, CA, AIAA 98-2075, April, 1998.

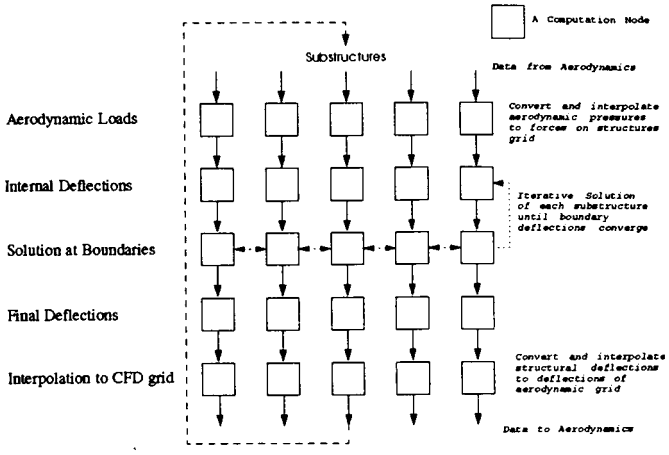


Figure 1 Solution flow chart.

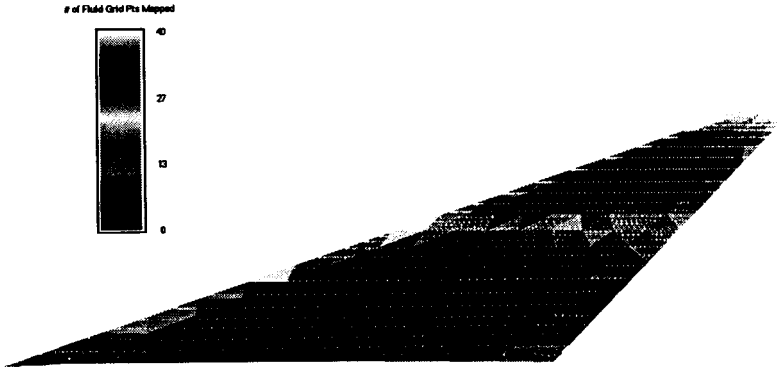


Figure 2 Fluid/structure grid mapping.

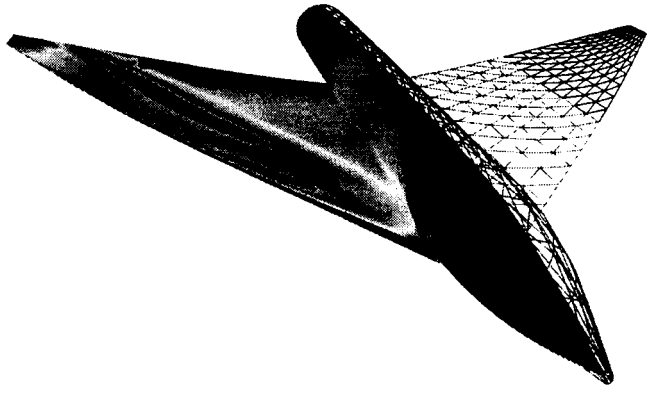
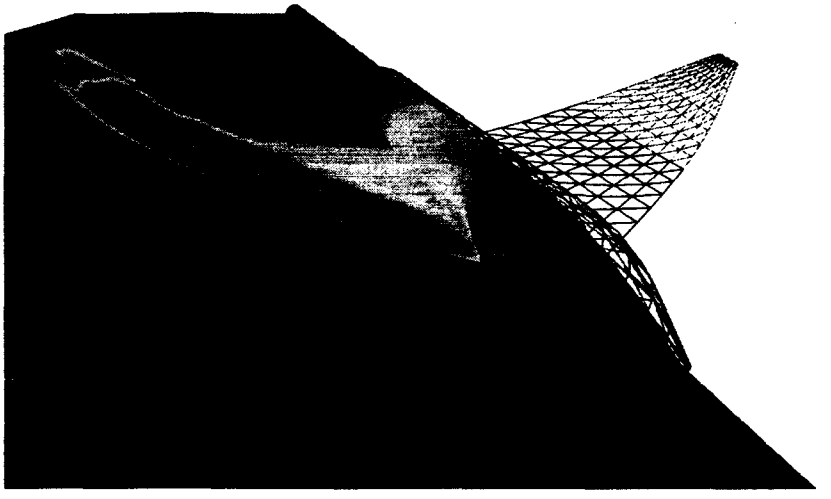


Figure 3 Typical aeroelastic solution.

User's Manual



1. Introduction

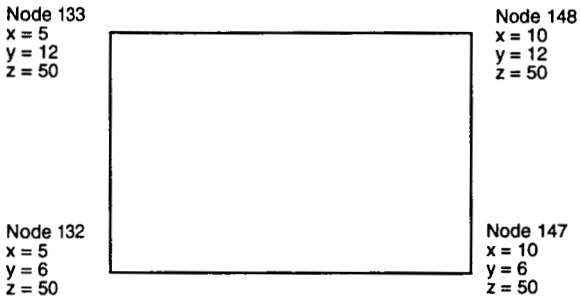
ENSAERO_FE is the parallel substructure-based finite element solution code. It can run as a standalone code or be used as a structural module for ENSAERO. It uses NASTRAN created stiffness matrices for each substructure, and iteratively solves for the structural deflections. Loads and deflections are exchanged with the ENSAERO_F fluids module via MPI.

The general approach used in the code is to break the aircraft structure into multiple substructures. This is very similar to breaking the fluid analysis domain into multiple zones. Each substrcture is assigned to a single computational node. That node solves for the deflections of its substructure and communicates with the other nodes to determine the deflections of the shared (also called boundary or external) nodes.

Each substructure is modeled in NASTRAN using its full library of elements. NASTRAN is instructed to generate the stiffness, mass, and node ordering matrices for each substructure. Additional data files describing how the substructures are connected are created by the user.

Additional input files are required to set up the fluid/structure interaction. The first set of files indicates which portions of the structural model are to exchange data with the fluids module (i.e. the aircraft skin). A file contains various problem scales: dynamic pressure, reference length, and freestream velocity. Another indicates how the structural grid matches up to the fluid zones.

2 NASTRAN Setup



NASTRAN is used to create the stiffness (and mass) matrices for each substructure. Its complete library of linear elements is available.

In order for the code to keep track of the internal/ external node bookkeeping necessary for this analysis, NASTRAN must not rearrange the stiffness matrices or apply boundary conditions to the matrices. In other words, NASTRAN's "Bandit" routine must be turned off.

The following NASTRAN executive deck is used to save the unaltered stiffness and mass matrices, as well as some bookkeeping information:

```
NASTRAN BANDIT=-1
ID Parallel analysis
APP DISP
SOL MODES
TIME 1
ALTER 42
OUTPUT5  KGG,,,,//-1/15//1 $  Output stiffness matrix
OUTPUT5  MGG,,,,//-1/16//1 $  Output mass matrix
OUTPUT5  GPL,,,,//-1/17//1 $  Output grid ordering vector
JUMP     FINIS $              Stop
ENDALTER
CEND
```

To use NASTRAN as an ENSAERO_FE front end, create numbered NASTRAN input files "0.dat", "1.dat", etc. for each seperate substructure. NASTRAN's invocation script must be modified to assign file names to the unit numbers specified in the previous executive control deck. Modifying the script to contain the following definitions will accomplish this:

```
set probname = $1
set ft05=$probname.dat
set ft06=$probname.out
set ft15=k.$probname
set ft16=m.$probname
set ft17=o.$probname
```

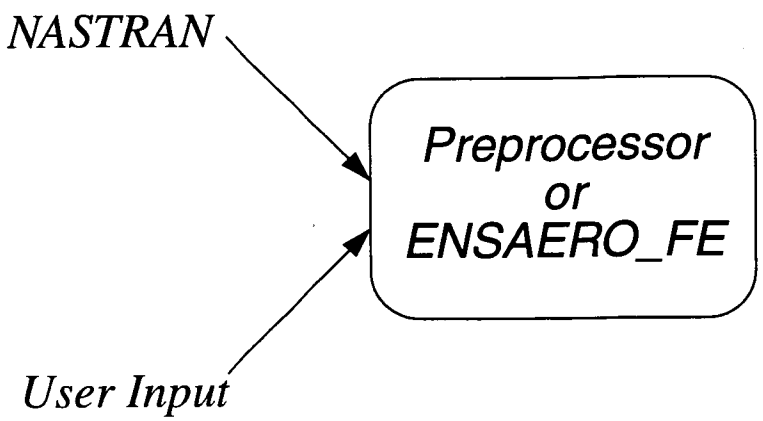
Use the ENSAERO_FE version of the NASTRAN invocation script(see section 11) by running it for each input file:

```
nastran 0
nastran 1
```
etc.

This will generate files containing the stiffness matrices, "k.0", etc., the mass matrices, "m.0", etc. and the ordering information, "o.0", etc.

The "o.N" file allows the user to use the same arbitrary node numbering scheme that is used in the NASTRAN input decks in the rest of the ENSAERO_FE input files. The code uses this file to connect a given node number to the rows and columns of the stiffness/mass matrix that contain the node's degrees of freedom.

3 Input Files



There are quite a number of input files necessary to run this code. An external preprocessor code is used to perform all of the once-per-configuration chores in advance of using expensive parallel compuation time. At this point, use of the preprocessor is optional. The main code looks first for the preprocessed data files. If they are not found, it will read or try to read the various user input and NASTRAN generated files itself. Use of the preprocessor is recommended, and described in the next section. Either way, the input data files are the same.

In most cases, there must be copies of the input files corresponding to each substructure in the problem. The letter "N" will be used in place of the substructure number for these types of files (i.e. the filename "bc.N" indicates there should be a "bc.0", "bc.1", etc.). There are a few input files that are for the entire problem. Such files end in ".dat".

Overview

The following data files are required to run this code:

```
k.N             - substructure stiffness matrix
o.N             - substructure ordering vector
bc.N            - boundary condition file
c.N             - skin connectivity file
```

```
s.N             - skin grid file
bnode.dat       - substructure attachement file
scales.dat      - aerodynamic scales
idsm.dat        - fluid/structure connectivity
1.N             - load file (standalone only,
                    optional)
```

Fluids Inputs

This code does not read any of the fluids module's input files. But, they have to be correctly setup to allow the combined system to work correctly. See the ENSAERO_MPI documentation for details.

Finite Element Data - *k.N, m.N, d.N, o.N*

See the previous section on NASTRAN use for instructions on creating these four files. "k.N" contains the substructure stiffness matrix, "m.N" contains the substructure mass matrix, "d.N" contains the substructure damping matrix and "o.N" contains the substructure node ordering vector. The mass and damping matrix files are required only for dynamic structural problems (and are set to zero if not supplied). The NASTRAN input decks "N.dat" are not required for this code.

Boundary Conditions - *bc.N*

The user must set up boundary condition (bc.N) files for each substructure. Most of the user-input files for ENSAERO_FE follow a standard format: the first line is the number of records, and each subsequent line is a record. A boundary condition file must exist for each substructure, even if there are no conditions to be enforced. In that case, the file should contain a single zero, "0", entry.

Each boundary condition record consists of a node number and a list of degrees of freedom to set to zero. These degrees of freedom are specified in NASTRAN format; i.e., 1-3 are the translational degrees of freedom, and 4-6 are the rotations. The node numbers are the ones specified in the corresponding NASTRAN input deck.

At this point, boundary conditions have to be configured manually. At some point, these could become at least partially automated by parsing the NASTRAN decks for the appropriate information.

Example bc file

```
2
1,123456
2,23456
```

In this case, node 1 is completely fixed and node 2 is only free to move in the x direction.

Aircraft Skin - *c.N, s.N*

In order for the loading and deflection data to be exchanged with the fluids code, the user must indicate what portions of the aircraft are on the "skin". Only nodes and elements that are specified as skin are used to exchange load/deflection data with the fluids. This specification of the aircraft skin is done by editing the input NASTRAN substructure data file into two additional files indicating the grid and grid connectivity of the aircraft skin.

The "c.N" file contains the skin grid connectivity information. This is in the form of NASTRAN element cards. Supported rectangular elements are CSHEAR, CTWIST, CQDMEM*, QDPCT, and CQUAD*. Supported triangular elements are CRTRPLT, CTRIA*, CTRIM6, CTRMEM, and CTRPLT. As with the other input files for this program, the first line contains the number of records in the file. This is followed by a comma, and a "1" or a "2" indicating if the cards use one or two lines.

Partial example c.N file

```
136,1
CQUAD4          1       3       1       2       11      10
CQUAD4          2       3       2       3       12      11
...
```

The "s.N" file file contains the skin grid information. This is in the form of NASTRAN "GRID" cards. Again, the first line of the file specifies the number of records, followed by a comma, and then a "1" or a "2" to indicate if the cards use single or double precision.

Partial example s.N file

```
162,2
GRID*                   1                0.00000000000   0.00000000000QGRD  1
*GRD  1   0.15648150444
...
```

Substructure Attachment - *bnode.dat*

The substructure attachment file "bnode.dat" is a single file for the entire structure. It indicates which nodes of each subtructure are attached to which of each other substructure. There is no requirement that corresponding nodes be numbered identically in each substructure; this file takes care of that detail.

This file starts with a line indicating the number of records. Each record is two lines long. The first line is a count of the number of substructures listed in the second line. The second line is a list of paired numbers indicating the substructure and its node number that is to be matched to the other nodes in the list.

Example bnode.dat file

```
2
2
0,37,1,37
4
0,21,1,21,4,21,5,25
```

This file indicates that there are two nodes that are shared between processors. Node 37 on processor 0 is the same as node 37 on processor 1 and Node 21 is the same on processors 0, 1, and 4 and corresponds also to node 25 on processor 5. Again, there is no requirement that corresponding nodes be called the same thing; between this file and the "o.N" file, the code can figure out where things go.

Aerodynamic Scales - *scales.dat*

The "scales.dat" file contains problem scaling constants used by the fluids code. The three values are a length scale "phylen", the dynamic pressure "dynpre" and the free stream velocity "frevel". These values are used to convert the pressure coefficients supplied by the aerodynamic code into nodal forces. This file replaces the "modals.dat" file specified in the ENSAERO documentation for the modal structural code.

Example scales.dat file

```
31.2963                                    phylen
0.5                                        dynpre
10000.0                                    frevel
```

Fluid/Structure Grid Connectivity - *idsm.dat or idsm.N*

The "idsm.dat" or "idsm.N" file(s) specifies the correpondences between the structural grid and the fluids grid. The values specified in this file for the structural skin are matched to the fluid grid sections with the same value. The fluid grid is marked by the "idph" value in "fsintf.dat".

Two different versions of this file are supported. The program automatically detects which one the user has chosen (by attempting to open first idsm.0, then idsm.dat).

The first type of file allows one idsm value per substructure. Thus, it contains one line for each substructure. Since the number of substructures are known to the code by the time it reads the idsm file, no linecount line is necessary.

Example idsm.dat (type 1) file

```
1
0
0
```

The second type of file allows a different idsm value for each skin element, but requires a file for each substructure, rather than a global file. Its format is a line count header followed by pairs of element numbers and idsm values.

Important: For the moment, the element number is ignored by the code. Instead the ordering is assumed to be the same as that of the c.N file.

Partial Example idsm.N (type 2) file

```
168
        1       1
        2       1
        3       1
        4       1
...
```

Load files - *l.N*

Load files are read only by the preprocessor code and are used only when the parallel code is run in stand-alone mode (i.e. no fluids) A record contains a node number followed by six values which are the loads applied to

each of the six degrees of freedom at that node.

Note that even in stand-alone mode, the use of this file is optional. If it is not present, the subroutine "applyload" is called instead to generate the problem force vector.

Example l file

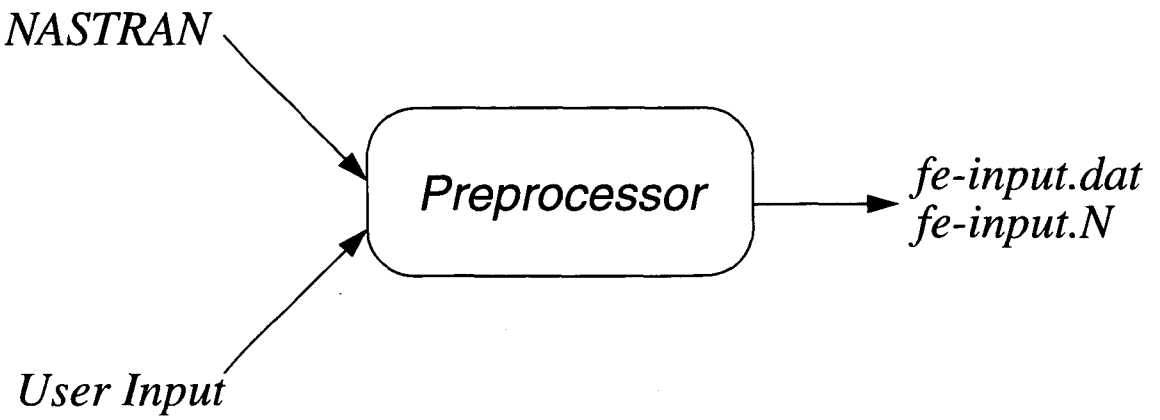
```
1
11,100.0,0.0,0.0,0.0,0.0,0.0
```

In this case, a load of 100.0 in the x direction is applied at node 11.

Restart files - *fe-output (.N and .dat)*

The "fe-output.N" and "fe-output.dat" files are used to restart the code from a previously finished run. They are created by the code as part of its final output. They are exactly the same format as the preprocessed input files, and thus to continue from a previous run they should be renamed to "fe-input" with the appropriate extensions. The code will read them just the same as normally preprocessed data.

4 Preproc.exe - Preprocessor



The use of the preprocesor code "preproc.exe" is strongly recommended. It performs all of the once-per-configuration setup computations such as generating the fluid/structure interface and compressing the stiffness matrix. It is also the only way to access many recent additions to the code, such as dynamic structural analysis.

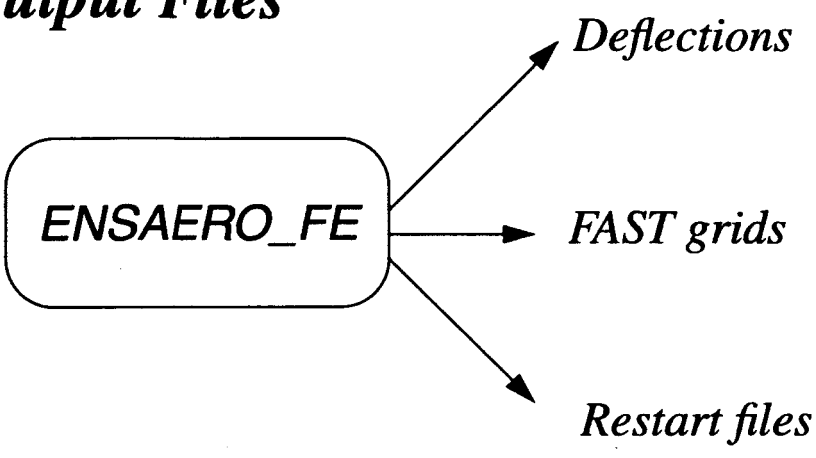
The code should be run once in a directory containing all the required (and desired optional) input files. It will generate a global input file "fe-input.dat" and substructure input files "fe-input.N" for each set of substructure inputs found.

The preprocessed data files are stored in the "StIFF" data format(see section 12), described in this documentation. They are architecture and operating system independent.

Only the preprocessor output files (fe-input.*) need be present when the main ENSAERO_FE code is run. It will automatically detect their presence and use them. In this case, any old-format (k.N, etc.) input files are ignored.

Remember to rerun the preprocessor whenever the input data are changed.

5 Output Files



A wide variety of output files are created by the program including plain text and FAST plottable files for both the full structure and the component substructures.

The "deflections.N" files contain the plain text six degrees of freedom deflections for each substructure and the corresponding input node number.

The "deflgrid.N" files contain a FAST format grid file for the deflected substructure skin. To load them in FAST, select "Grid", "Formatted" and "Unstructured". The "SurferU" module must be used to plot the unstructured grid. This grid is made up of the interpolation triangles used to map loads and deflections between fluids and structures.

"deflgrid.dat" contains a FAST format grid for the entire deflected structural skin.

The "deflgrid2.N" and "deflgrid2.dat" files contain a different version of the same grid that is in the files without the "2". This version of the file allows the user to map a scalar onto the grid, a feature not normally available in a FAST unstructured grid. It does this by tripling the grid and connectivity information. Once this is done, a color can be specified for each

skin triangle.

The "skingrid.N", "skingrid.dat", "skingrid2.N" and "skingrid2.dat" follow the same naming scheme as the "deflgrid" files. In this case, they contain the undeflected skin grids in a FAST format for each substructure and for the full structure.

The "fgrid.N" files contain the merged fluid dynamics grids.

The files "fe-output.dat" and "fe-output.N" contain the data necessary to restart the code and continue from the last timestep of a completed run. They are identical in format to the preprocessed "fe-input" files and only need to be renamed in order to be used.

"fe-converge.dat" contains the history of the deflection of a single degree of freedom. At this point, that degree of freedom must be configured inside the source code.

6 Running stand-alone

ENSAERO_FE can be run as a stand-alone structural analysis code as well as a module of ENSAERO_F.

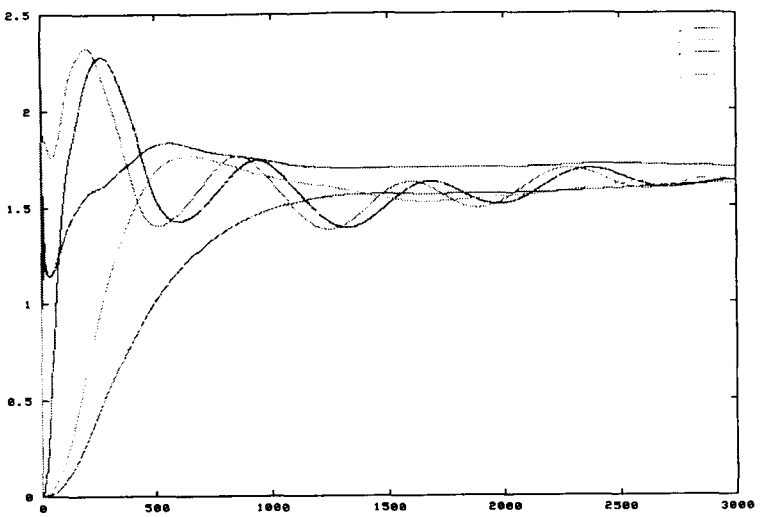
The finite element code will run in stand-alone mode if either of the following conditions is met:

1) The file "input/multid.dat" (an ENSAERO_F input file) does not exist.
2) The file "input/multid.dat" specifies no fluids codes are to be run.

In stand-alone mode the user has two choices for specifying the structural loads. "`l.N`" (the letter ell, not the number one) files can be used to specify static loading. The load input files must be present during preprocessing. Load input is not supported directly by the main code.

The second option for loading is to modify the "applyload" subroutine. Since the subroutine is called every time step, dynamic or nonlinear loading is possible. Note that this subroutine is called only if no static load files were found during preprocessing.

7 Dynamic Structural Analysis



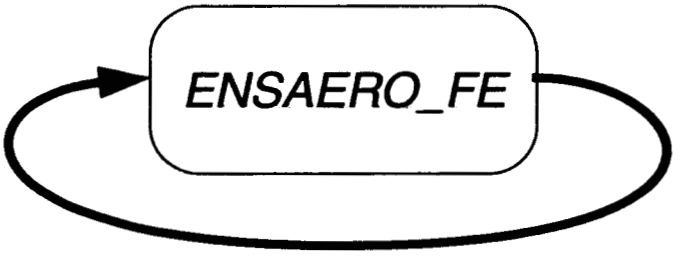
ENSAERO_FE can perform dynamic structural analysis using the Newmark (constant average accelleration) method. A linear accelleration method is also supported. The method used may be changed by editing the parameters in the "newmarkinit" subroutine found in *struct/dynamic.f*.

When run with the fluids code, ENSAERO_FE performs a dynamic analysis when "itask" is set to 4 or 5. This analysis is identical for both itask settings. A time step matching that used by the fluids is used.

When run standalone, the dynamic options are controlled by two lines in the main routine, *parstruct.f* , seen below. The logical variable "dynamicrun" should be set to ".true." or ".false." and the time step size "deltime" should be set as desired. Note that the Newmark method is unconditionally stable.

```
c Control for standalone dynamic structural analysis
c   when run with fluids code, fluids inputs will
c   overide these settings
      dynamicrun=.false.
      deltime=0.0
```

8 Restarting



The program supports restarting if the user wishes to continue an analysis. The previous run must have completed normally for this to work.

See the ENSAERO documentation for full details, but a very short summary appears below. Note that steps 1 and 2 are not required when running in stand alone mode.

1) Edit "ensaero_f.dat":
   Change the "restart" variable to 1 or 2.
   Change the "start" and "stop" timesteps to the new values
   (i.e., to restart a run that ran from time steps 1-50, change these to 51 and 100).

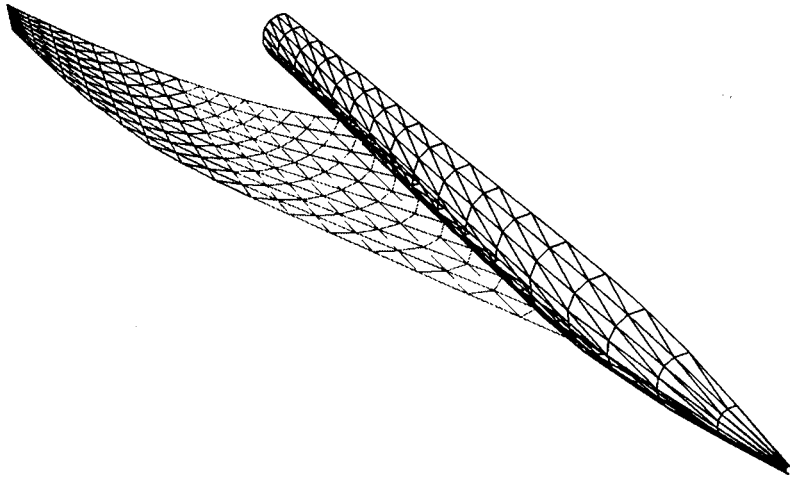
2) Rename or copy all " restart.out_X_Y" files as "restart.dat_X_Y". These files contain the fluid solution.

And for the structures code:

3) Rename or copy "fe-output.dat" as "fe-input.dat".

4) Rename or copy all "fe-output.N" files as "fe-input.N".

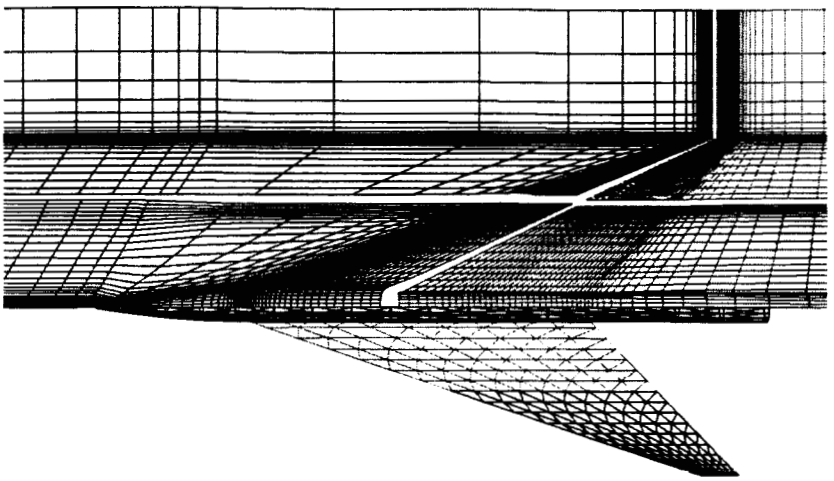
9 Other Issues



This program is configured for medium sized problems. The program will complain and stop if a user tries to run a problem too large for the PARAMETERs it was compiled with. These are contained in "parsub.h" and are well documented within that file if they need to be increased.

Relaxation of the structural response is supported in the "relaxscale" subroutine in *struct/parsubsub.f*. That routine returns a value used to scale the computed structural perturbation based on the current loop value. This support is intended to damp out oscillations in the iteration history of static aeroelastic solution of very flexible structures. It may be useful for other purposes as well.

Example Problems



A few example problems are included with the package for learning and testing purposes. They are found in the *examples* subdirectory.

Flat Plate Problem: *examples/plate*

The flat plate problem can be used to test the stand alone capability of the code, as well as scalability. A simple FORTRAN program is used to generate a flat plate composed of as many QUAD4 elements as desired and broken into as many substructures as desired.

The first step in setting up a problem is to edit the *control.dat* file.

Sample *control.dat* file

```
40                    # elements in x direction
50                    # elements in y direction
2                   # substructure sections in x dir
2                   # substructure sections in y dir
1.0                   x dimension, length
1.0                   y dimension, length
0.1                   z dimenzion, thickness
3.0e6                 e, Young's modulus
0.30                  nu, Poisson's ratio
```

```
1.0                     rho, density
123456                  bottom edge bound. cond.
6                       Left edge BC
6                       top edge BC
6                       right edge BC
6                       interior BC (zero if none)
```

The comments on the right should make the file fairly self explanatory. The boundary condition specifications are NASTRAN style and list which degrees of freedom, 1-6, are constrained to be zero on each edge and in the interior.

The next step is to edit the files *head.dat* and *tail.dat*. These two files will be copied to the top and bottom of the generated bulk data decks. The *head.dat* file should contain the desired NASTRAN executive and case control cards. The *tail.dat* may just contain an "ENDDATA" card.

After editing these files as desired, run *plate*. This will generate the desired number of NASTRAN bulk data decks as well as boundary condition and connectivity files. Next, run either the script *cat-em* or *cat-em2* (use *cat-em2* if you have 11 or more substructures) to concatenate the header and footer files with the bulk data files.

You will now have numbered NASTRAN input decks for each plate substructure, e.g., *0.dat*. Run the ALTERed NASTRAN on each of these input files to generate the stiffness and ordering files.

Create loading files, *l.N*, as desired, then run the preprocessor and finally the main code.

Arrow Wing - Body problem: *examples/awb*

Complete input decks for both stand alone and aeroelastic analysis of a simple arrow wing/body aircraft are supplied. This model is the basis for all of the aircraft figures in this documentation and uses eight fluids zones and three substructures.

10 File Naming Conventions

"N" is the substructure/parallel processor number
(NAST) indicates a file generated by a NASTRAN run
(USER) indicates a user generated file
(ENSAERO_FE) indicates a file generated by a previous run

Input files (for preprocessor)

```
N.dat      - NASTRAN input deck                (USER)
N.out      - NASTRAN run output               (NAST)
bc.N       - Boundary condition file          (USER)
c.N        - Skin connectivity file           (USER)
k.N        - Stiffness matrix                 (NAST)
m.N        - Mass matrix                      (NAST)
d.N        - Damping matrix                   (NAST)
o.N        - Node ordering file               (NAST)
s.N        - Skin grid file                   (NAST)
bnode.dat  - Substructure attachment file     (USER)
scales.dat- Aerodynamic scales                (USER)
pick one:
idsm.dat   - Fluid/structure grid connect     (USER)
idsm.N     - Fluid/structure grid connect     (USER)
```

Preprocessor Output Files

```
fe-input.dat - global (skin grid, etc.)
fe-input.N   - local (stiffness matrix, etc.)
```

Output Files

```
deflections.N - text deflections
deflgrid.N    - FAST substucture deflected skin grid
deflgrid.dat  - FAST full structure deflected skin grid
deflgrid2.N   - tripled FAST substructure deflected skin grid
deflgrid2.dat - tripled FAST full structure deflected skin grid
skingrid.N    - FAST substructure undeflected skin grid
skingrid.dat  - FAST full structure undeflected skin grid
skingrid2.N   - tripled FAST substructure undeflected skin grid
skingrid2.dat - tripled FAST full structure undeflected skin grid
fgrid.N       - merged fluids grids
fe-output.dat - restart file (global data)
fe-output.N  - restart file (individual substructure data)
fe-converge.dat   - convergence history of a single DOF
(standard output) - diagnostics, warnings, errors, and summary'
```

11 NASTRAN invocation script

The following script is used to invoke NASTRAN. The script specifies the file names for all the input and output files:

```
#!/bin/csh
#   streamlined, non-interactive NASTRAN invoker
unalias rm
#clear
set   rfdir=$HOME/nastran/rf
set nasexec=$HOME/bin/nastrn.exe
set naschk=$HOME/bin/chkfil.exe
set probname = $1
echo ' '
  if ( $probname == '' ) then
  echo '                                    NASTRAN'
  echo ' '
  echo -n 'Please give problem id for designation of files ===> '
  set probname = $<
  endif
# set ft01=$probname.pun
set dbmem=12000000
set ocmem=2000000
set ft01=none
set ft04=none
set ft03=$probname.log
set ft05=$probname
if (! -e $ft05) then
      set ft05=$probname.inp
endif
if (! -e $ft05) then
      set ft05=$probname.dat
endif
# if we can't find the input file, reset to what was given
if (! -e $ft05) then
      set ft05=$probname
endif
set ft06=$probname.out
set ft08=none
# set ft11=$probname.out11
set ft11=none
set plt2=none
set script=$probname.cmd
set nasscr=$cwd/temp$$
set ft12=none
set ft15=k.$probname
set ft16=m.$probname
set ft17=o.$probname
```

```
#set ft18=$probname.coord
set ft18=none
set sof1=none
# set sof1=$probname.sof
set sof2=none
set sft12=
set nptp=$probname.nptp
set optp=none
   if ( ! -e  $ft05 ) then
   else
   if ( -e nogood1 ) then
   rm nogood1
   endif
   if ( -e nogood2 ) then
   rm nogood2
   endif
   if ( -e nogood3 ) then
   rm nogood3
   endif
   $naschk < $ft05
        if ( -e nogood1 ) then
        set ft04=$probname.dic
        rm nogood1
        endif
        if ( -e nogood2 ) then
        set plt2=$probname.plt
        rm nogood2
        endif
        if ( -e nogood3) then
        set ft04=$probname.dic
        set plt2=$probname.plt
        rm nogood3
        endif
   endif
if ( -e $script ) then
rm $script
endif
touch $script
echo '#/bin/csh' >> $script
echo ' unalias rm ' >> $script
echo 'if ( -d ' $nasscr' ) then' >> $script
echo 'rm -r '$nasscr >> $script
echo 'endif' >> $script
echo 'mkdir '$nasscr >> $script
echo 'if ( -e '$nptp ' ) then'>> $script
echo 'rm '$nptp >> $script
echo 'endif' >> $script
echo 'if ( -e '$ft03 ' ) then'>> $script
echo 'rm '$ft03 >> $script
echo 'endif' >> $script
echo 'if ( -e '$ft01 ' ) then'>> $script
echo 'rm '$ft01 >> $script
echo 'endif' >> $script
echo 'if ( -e '$ft04 ' ) then'>> $script
```

```
echo 'rm '$ft04 >> $script
echo 'endif' >> $script
echo 'if ( -e '$ft06 ' ) then'>> $script
echo 'rm '$ft06 >> $script
echo 'endif' >> $script
echo 'if ( -e '$plt2 ' ) then'>> $script
echo 'rm '$plt2 >> $script
echo 'rm '$ft04 >> $script
echo 'endif' >> $script
echo 'if ( -e '$ft06 ' ) then'>> $script
echo 'rm '$ft06 >> $script
echo 'endif' >> $script
echo 'if ( -e '$plt2 ' ) then'>> $script
echo 'rm '$plt2 >> $script
echo 'endif' >> $script
echo ' env  NPTPNM='$nptp '\' >> $script
echo '  PLTNM='$plt2 ' DICTNM='$ft04 ' PUNCHNM='$ft01 '\' >> $script
echo '  FTN11='$ft11 ' FTN12='$ft12 ' DIRCTY='$nasscr '\' >> $script
echo '  LOGNM='$ft03 ' OPTPNM='$optp ' RFDIR='$rfdir '\' >> $script
echo '  SOF1='$sof1 ' SOF2='$sof2  '\' >> $script
echo '  FTN14=none FTN17=none FTN18=none FTN19=none FTN20=none \' >> $script
echo '  FTN15='$ft15 ' FTN16='$ft16  '\' >> $script
echo '  FTN17='$ft17 ' FTN18='$ft18  '\' >> $script
echo '  FTN21=none FTN13=none \' >> $script
echo '  DBMEM='$dbmem ' OCMEM='$ocmem '\' >> $script
echo    $nasexec' < '$ft05' >'$ft06 >> $script
echo 'rm -r '$nasscr >> $script
echo 'if ( -e none  ) then'>> $script
echo 'rm none' >> $script
echo 'endif' >> $script
echo 'echo == NASTRAN - "'$probname'"  done ==' >> $script
chmod +x $script
$cwd/$script &
```

12 StIFF *format specification*

StIFF
"STructural Interchange File Format"
File Specification

Introduction

"StIFF" is a format for exchange of structural data (stiffness matrices, mass matrices, etc.). It is intended to be platform independent and extendable to include new data types without breaking existing codes. Any type or combination of types of data may be stored in a StIFF file.

Inspiration

The StIFF format is conceptually similar to the TIFF and IFF image file formats. Both achieve platform independence through the use of data blocks.

Approach

The formatting approach used is block oriented. Each block starts with a header consisting of the block name and the block length. The allows a reading program to skip over unneeded or unfamiliar blocks. In fact, a reading program must do this for things to work well.

While the contents of each block type will be different, in general, each block will have a secondary header specifying the format of the actual data in the block, so that the data may be read regardless of the bit format of the writing and reading machines.

This standard is also defined to be friendly to the limited I/O capability of the FORTRAN programming language. Most of the structural codes of interest are written in FORTRAN rather than a language with more powerful I/O capabilities. Thus, data suchas block lengths are given in units of text lines rather than bytes so that a FORTRAN reader can loop over an empty 'read()' statement to skip past a block.

Structure

A StIFF format file consists of three parts: header, any number of blocks, and a tail. While the blocks can appear in any order, for efficiency sake, the writer program should write more general blocks first and more specific

blocks last. For instance, a stiffness matrix block should probably go before a mass matrix block. This is because the stiffness matrix is required for both static and dynamic analyses, but the mass matrix is needed only for dynamics. Thus, in some cases, the reader code can find all it needs early and stop reading.

The header and tail blocks can be viewed as special types of block that (at least currently) contain no data sections. Since there are other reasons for no-datablocks, such as option flags, and since the header and tail blocks definition may change in some future version of this document, all blocks will have a line length value, including blocks of length zero. Thus, any StIFF block will have the structure:

```
BLOCKNAME <number of data lines or zero>
      .
      .
      .
<specified number of data lines of data>
      .
      .
      .
```

In general, for ease of reading, block names will be all uppercase (the StIFF header is an exception). Future block type definitions may be mixed or lowercase if needed for clarity. Writing codes should take care to match the case of the block name in the documentation to avoid reader code parsing problems. Block names are limited to 10 characters in length.

Block Definitions

I> Header/Tail blocks

Block: StIFF n

The StIFF header block consists of the five-character block named "StIFF". (upper and lower) case is important, allowing the header to be used as a "magic cookie" to determine the type of file.
This block contains no data, but reader code should still check the line count variable and skip lines if required. Thus, a writer code could put some sort of comment in the header, although the use of a text block would be better for this purpose.

Block: END n

The END tail block serves as the last block in a StIFF file. Its inclusion allows the reader code to stop before getting an end-of-file error. As with the header block it should contain no data.

II> Text blocks

Blocks: COMMENT, TEXT, TITLE
These block types are all designed to allow the inclusion of lines of raw text. These may be ignored or read and printed as desired by the reader code. The different types of block names are provided so that different behaviors can be used.

It is sugested that COMMENT blocks be used for data documentation and ignored by reader codes and TITLE blocks be read and printed in program output.

III> Generic Data blocks

There is an unlimited number of possible data blocks. A few generic ones are specified below. Others may be added as desired; reader codes should be designed to safely skip over any blocks that they do not understand. To reduce conflicts, however, application specific data should use a unique prefix. For instance, a block specifying options for ENSAERO_FE might be named "ENSFE-OPT".

Blocks: STIFF, MASS, DAMP, SQ-MATRIX
There are three parts to a square matrix block. The first two lines are valid FORTRAN FORMAT specifications for reading first the matrix numerical data, then the pivot or index data. Thus, the first will be for real numbers and the second for integers. They should have parentheses around them. Example: "(F12.6)". For formats without pivot or index data, the second FORMAT specification is still required, but is ignored.

The second part specifies the storage format and the number of elements to be read. Depending on the storage format used, the number of fields on this line may vary.

The third part of the block is the actual matrix data that are read according to the instructions in the first two parts.

Storage Format: RAW n
The RAW storage format stores the entire matrix explicitly. The single data field on the format line indicates the dimension of the square matrix. The actual data are stored across by row, i.e., a(1,1), a(1,2),..., a(1,n), a(2,1),...

Storage Format: TRI n

The TRI storage format stores the upper triangular part of a symmetric matrix. The single data field is the dimension of the full matrix. The actual data are stored across by row, i.e., a(1,1), a(1,2),..., a(1,n), a(2,2),...

Storage Format: NAST
The NAST storage format uses a NASTRAN style compression scheme to store a full matrix.

Storage Format: INV n
The inverse of the matrix is stored in RAW format. The data field indicates the dimension of the matrix.

Storage Format: FACTN (FACT1, FACT2,...) n
A factored version of the matrix is stored. The data field indicates the dimension of the matrix.

Storage Format: SKY na
The SKY storage format uses a Skyline storage scheme to store the upper triangular portion of a symmetric matrix. "na", the length (number of entries, not number of lines!) of the skyline data vector is specified on the format line. The integer pointer vector "maxa" is provided after the skyline stored matrix. "maxa" is preceded by its length "nma". Thus, a skyline store matrix would look like:

```
SKY na
 .
(lines of matrix data)
 .
nma
 .
(lines of maxa data)
 .
```

Blocks: LOAD, DEFLECT, ORDER, VECTOR
The vector storage blocks consist of three parts. The first is a storage format block, like the first part of the square matrix block type. The first line should contain a valid FORTRAN FORMAT specification for reading the vector data. It should be contained in parentheses. The second part is a storage scheme specifier. At this point only "RAW" is specified, but others may be added as needed. The third part is the actual data.

Storage Format: RAW n
The vector is stored explicitly and is of length "n".

Blocks: VECTOR2, VECTOR3, GRID

These multidimensional vector blocks work exactly like a one dimensional vector block, except with double or triple the data. The length "n" in the RAW specifier is the length of the vector. This is followed by 2 or 3 sets of data of length "n". For instance, 10 grid points would be stored as:

```
RAW 10
x1 x2 x3 ... x10
y1 y2 y3 ... y10
z1 z2 z3 ... z10
```

Of course, the number of values on a single line depends on the FORTRAN FORMAT used. Generally, however, the transition between components will start on a new line.

Blocks: IVECTOR, ENSFE-IBCS, ENSFE-IBNO, ENSFE-MAPG

Identical to the VECTOR format, but stores integer values.

Blocks: MATRIX, ENSFE-KIE

This block is for storing a nonsquare matrix. It is very similar to the other matrix and vector storage blocks. Use the SQ-MATRIX/RAW definition but with two data fields on the storage scheme line:

Storage Format: RAW n m

The matrix is stored explicitly and is of dimension n x m

--- Example StIFF file ---

```
StIFF 0
COMMENT 2
This is comment line 1
This is comment line 2
VECTOR 4
(F7.2)
RAW 2
1234.56
9876.54
END 0
```

13 Utility Programs

A few small, useful utility programs are included in the *util* subdirectory.

Autobnode

"Autobnode" is a utility for use with manually substructured data files (such as when Patran is used to chop up an aircraft). Its main purpose is to generate the "*bnode.dat*" file, but it also performs a variety of validity checks on the substructured data.

To work, the program needs access to the full structure NASTRAN file as well as to all of the substructure NASTRAN files. A file called "*files*" is used to control the code and tell it which files are to be used. The first line gives the file name for the full structure file. The second line gives the number of substructure files. The file names for the substructure data files are given one per line after that.

Example "*files*" file

```
full.dat
3
wing.dat
fuselage.dat
tail.dat
```

The *bnode.dat* file is generated by assuming that consistant node numbering is used between the various substructure files (this is **not** a requirement of the main code). Thus, any nodes with the same ID number are assumed to be the same. And any that appear in more than one substructure are then listed in the *bnode.dat* file.

In addition, the following validity checks are performed:

- All grid points in full file must be in at least one substructure
- All substructure grid points must appear in full file
- All elements must appear in full file and in exactly one substructure
- All grid points specified by elements in a substructure appear in that substructure
- No unused grid points appear in a substructure

Should any of these checks fail, a warning will be issued, and details will be written to a file called *autobnode.error*.

The size problem allowed by the program can be controlled by editing the *autobnode.h* header file.

Autosub

"Autosub" is an automatic substructuring code. It reads a NASTRAN input deck and generates somewhat equally sized substructures from it. It should be used with care, however, as it isn't very bright. Basically, it sorts all the input grid points by a single one of their coordinates (x, y, or z), assigning the first N points to the first substucture, etc. Thus, it simply slices the structure up along one axis.

Autosub is written in C. Before using it, the user will need to configure it for the problem at hand. The `#define` statements at the top of the code will need to be modified to reflect the size of the problem. Also, the desired slicing direction needs to be set in the "indexr" call.

For aircraft applications, this code is best used after some manual substructuring. For a full aircraft, the engineer might manually cut the structure into wing, fuselage, and eppenage substructures. Then, autosub could be used to slice the wing in the "y" direction, the fuselage in the "x" direction, and the eppenage in the "z" direction. In such a case, the `#define` for "ibase" can be used to generate appropriate file numbers.

Autosub also writes some of the preprocessor input files such as "bnode.dat". Such files may need to be manually edited if the code is only slicing part of a larger structure (as in the example above).

El2Nas

"El2Nas" is a program to convert Elfini structural data to NASTRAN data.

Usage

"el2nas" is simple to use. It is invoked by typing:

```
el2nas [elfini-file nastran-file]
```

The default input file name is "elfini.dat". Similarly, the default output file name is "nastran.dat". If other file names are desired, they may be specified on the command line, but both must be given.

Limitations

There are a number of limitations in the code due to differences in the way the two codes work and limitations in the available test decks. The second is obvious: if a certain Elfini input type or option in the demo deck(s)

used to develop the code are not seen, that option is not supported.

Shear and Bending of Composite Plates

The test Elfini desks used to develop the code include composite plates with either no bending stiffness or stiffness only in shear. There are no NASTRAN elements that support this combination. (The CSHEAR element, for instance, requires an isotropic material.) The converter models these plates as full composite QUAD4 plates.

Solid Orthotropic Elements

The test Elfini decks contain a variety of solid elements with orthotropic material properties. Only the hexahedral NASTRAN solid element allows this type of material. The solid wedge and tetrahedral elements require isotropic materials. For these cases, the converter models the elements using an approximate isotropic equivalent.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Month 1999	Technical Memorandum

4. TITLE AND SUBTITLE

User's Guide for ENSAERO_FE Parallel Finite Element Solver

5. FUNDING NUMBERS

509-10-11

6. AUTHOR(S)

Lloyd B. Eldred and Guru P. Guruswamy

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Ames Research Center
Moffett Field, CA 94035-1000

8. PERFORMING ORGANIZATION REPORT NUMBER

A-99V0021

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NASA/TM–1999-208781

11. SUPPLEMENTARY NOTES

Point of Contact: Guru P. Guruswamy, Ames Research Center, MS 258-1, Moffett Field, CA 94035-1000
(650) 604-6329

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified — Unlimited
Subject Category 08 Distribution: Standard
Availability: NASA CASI (301) 621-0390

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

A high fidelity parallel static structural analysis capability is created and interfaced to the multidisciplinary analysis package ENSAERO-MPI of Ames Research Center. This new module replaces ENSAERO's lower fidelity simple finite element and modal modules. Full aircraft structures may be more accurately modeled using the new finite element capability. Parallel computation is performed by breaking the full structure into multiple substructures. This approach is conceptually similar to ENSAERO's multizonal fluid analysis capability. The new substructure code is used to solve the structural finite element equations for each substructure in parallel. NASTRAN/COSMIC is utilized as a front end for this code. Its full library of elements can be used to create an accurate and realistic aircraft model. It is used to create the stiffness matrices for each substructure. The new parallel code then uses an iterative preconditioned conjugate gradient method to solve the global structural equations for the substructure boundary nodes.

14. SUBJECT TERMS

ENSAERO, NASTRAN, Parallel computers

15. NUMBER OF PAGES

39

16. PRICE CODE

A03

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified		