NASA-TM-87586 19860007142

NASA Technical Memorandum 87586

EXPERIENCES WITH A PRELIMINARY NICE/SPAR STRUCTURAL ANALYSIS SYSTEM

CHRISTINE G. LOTTS AND WILLIAM H. GREENE

OCTOBER 1985

NASA
National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

TABLE OF CONTENTS

Summary

Construction of a new structural analysis system based on the original SPAR finite element code and the NICE System is described. The system is denoted NICE/SPAR. NICE was developed at Lockheed Palo Alto Research Laboratory and contains data management utilities, a command language interpreter, and a command language definition for integrating engineering computational modules. SPAR is a system of programs used for finite element structural analysis developed for NASA by Engineering Information Systems, Inc.. It includes many complementary structural analysis and utility functions which communicate through a common database. The work on NICE/SPAR was motivated by requirements for a highly modular and flexible structural analysis system to use as a tool in carrying out research in computational methods and exploring new computer hardware. Analysis examples are presented which demonstrate the benefits gained from a combination of the NICE command language with the SPAR computational modules.

Introduction

Research in computational methods for structural analysis has been severely hampered by the complexity and cost of the software development process. Even though the researcher is usually interested only in a small aspect of the overall analysis problem, he is often forced to construct much of the supporting software himself. This time-consuming and expensive approach is frequently required because existing software that the researcher could potentially exploit is poorly documented internally, is poorly designed and coded, or both. After enduring this time-consuming software development effort, the researcher often finds that a thorough, complete evaluation of his new method is still impossible due to limitations of the software. This is true, for example, in many "research-oriented" finite element codes which have a limited element library or have arbitrary restrictions on how elements of different types are combined in a single model.

To address the above difficulties in computational methods research, the Computational Structural Mechanics Group at NASA Langley Research Center has undertaken the construction of a structural analysis software "test bed". The purpose of the test bed is to provide a system which can be easily modified and extended by researchers. In part, this is being achieved by exploiting advances in computational software design such as a knowledge of the role of command languages and formal data management techniques in ensuring program modularity.

The NICE (Network of Interactive Computational Elements) system (refs. 1 and 2) developed at Lockheed Palo Alto Research Laboratories is an example of a modern software architecture for supporting engineering analyses. The NICE system consists of three major components: a data manager, a command language called CLAMP for controlling analysis flow, and a command interpreter called CLIP for interpreting CLAMP commands and decoding general analysis input. Computational modules in the NICE system, called processors, are semi-independent programs which perform a specific, well-defined task. To enforce modularity, processors do not communicate explicitly with each other but instead communicate only by exchanging named data objects in the data base. To utilize these semi-independent processors

in a particular, complex analysis task, CLAMP procedures are written to describe the analysis to be performed and the algorithm to be used.

The concept of independent processors communicating through a global data base was demonstrated in the SPAR structural analysis system (ref. 3). SPAR contains processors for a wide range of linear, static and dynamic analyses. Typical processor functions include model definition, formation of finite element system matrices, solution of linear equations, and solution of a generalized eigenproblem. In addition to the modularity achieved through the use of a global data base, the SPAR processors are designed to be computationally efficient.

The decision to couple NICE and SPAR for the CSM test bed was based on four considerations. First, the details of the data management system in the original SPAR and NICE are quite similar. The data manager requests within SPAR processors are compatible with the NICE entry points. Second, the reliability, utility, and performance of the SPAR processors have been proven by almost a decade of use. Third, the concept of a high order command language controlling the execution of independent computational modules has been validated with the highly successful, proprietary analysis system denoted EAL (ref. 4). EAL has been used extensively for the past 8 years as the primary structural analysis tool at NASA Langley. Fourth, both NICE and SPAR are public domain software.

This paper describes experiences associated with integrating NICE and SPAR. Examples are presented to illustrate the input form and to demonstrate the power of the CLAMP language in directing processor flow. One concern is the computational overhead in terms of both CP (central processor) time and number of disk I/O operations associated with the data manager. To address this question, a comparison is made between EAL, which is quite efficient in this regard, and NICE/SPAR. Information is also provided on operational differences and processor changes between original SPAR and the current VAX/VMS version of NICE/SPAR.

Integration of SPAR and NICE

Because the data management philosophy of NICE is similar to the SPAR approach, the installation of the SPAR computational processors under NICE was relatively straightforward. SPAR data management usage is described in reference 5. In NICE, however, the data library concept has been extended so that all records of the data structure on the library (data sets) do not have to be in physically contiguous locations on disk. In addition, NICE includes with each data set, information describing its contents. This feature tends to make the data sets more self-describing, allowing easier interface by computational processor developers. This extended library/data set concept is denoted GAL/82.

The specific steps in the installation process were as follows:

1. The SPAR data management routines were rewritten (DAL, RIO, TOCO, LTOC, and DEL) to use NICE GAL/82 data library utilities. The RIO calling sequence required modification to include record numbers and data type.

2. SPAR routines RWINDZ, OUTZ, INZ, RRINZ were rewritten to use NICE indexed record utilities for scratch libraries.
3. READ and FIN were modified to interface with the NICE command language interpreter; READER was also modified to accommodate changes to the input syntax as described in Appendix A.
4. The routines used to maintain the SPAR Table of Contents and master index data structures (MATCH, NTOC, RDIND, STATIO, WRTIND, WRTINX, XEVICT, XOPEN, XREWND) and low level SPAR I/O routines (READX, WRITEX) were deleted.
5. Individual processors were modified to include additional parameters required in RIO calls and to provide processor identification in the NICE TOC; variables were added to common blocks where required for accessing record counts, matrix row size, and record data type; scratch libraries were opened where required.
6. A main program was written to control execution of individual processors by calling them as subroutines.

Usage descriptions of the primary processor/NICE data management interface subroutines are given in Appendix B. These routines (DAL, RIO, TOCO and LTOC) are used by the existing NICE/SPAR processors as the bridge between the SPAR data management method and the NICE nominal data set/named data record utilities. The routines which interface with NICE command language interpreter (FIN, INTRO, READ, and READER) are also described in Appendix B.

NICE/SPAR data libraries are written to disk files named NS.Lxx, where xx is the library number (01-30). Most processors use library number 1. However, by using the CLAMP *OPEN directive, a user can explicitly associate any legal external file name with a library. The data libraries are NICE/GAL82 format; data sets are nominal data sets using the same naming convention as SPAR. Records of the data sets are named records (currently implemented with all records simply named DATA.i); data sets are written as one record per SPAR block, or one record per SPAR data segment. The current NICE/SPAR data record contents are identical to the SPAR data set contents described in reference 6.

Integration of new user-developed processors may be accomplished according to the guidelines given in Appendix C. NICE/SPAR usage is described in Appendix A.

EXAMPLE STRUCTURAL ANALYSIS PROBLEMS

During NICE/SPAR development, many analysis problems have been designed and executed to verify the correctness of the system. Two of these problems are presented here to illustrate the input syntax, analysis flow,and use of typical CLAMP directives in describing analysis algorithms.

The first problem is the static stress analysis of a section of a toroidal shell. The input for this example is shown in figure 1. The shell wall consists of four layers of composite material with orientations $90^{o}/0^{o}/45^{o}/-45^{o}$. The finite element model consists of 337 nodes and 320 combined membrane-bending elements. (The SPAR designation for this element is E43). This example demonstrates the relatively straightforward usage of

NICE/SPAR for a small, sequential analysis problem. Processors TAB and ELD are used to input all geometrical and property data describing the model. The JREF command in TAB is used to align the joint reference frames with the shell coordinate system. Both the applied loading (defined in AUS) and the calculated displacements (from processor SSOL) are relative to these reference frames. Later in the analysis, the calculated displacements and reactions are converted to the global reference frame using the LTOG (local-to-global) command in AUS and then printed using processor VPRT. Stress information is calculated by processor GSF and then selectively printed in three different formats by separate executions of processor PSF.

The second example is the dynamic analysis of a planar, cantilever beam. The analysis is carried out using both a modal method and a direct integration of the system equations of motion using the Newmark integrator. This example shows how the SPAR processors and the NICE CLAMP command language can work together to perform a fairly complex analysis task.

The input for this example is shown in figure 2 and consists of five NICE CLAMP procedures. Procedure CANT_BEAM defines the beam model and calculates system stiffness and consistent mass matrices. The beam is excited by an initial displacement which is the static deformation shape resulting from a unit applied displacement at the tip.

If a modal transient response is being performed, procedure VIBR_MODES is called, followed by procedure TR_MODAL. A formal argument, nmodes, in VIBR_MODES indicates the number of vibration modes to be calculated. A similar parameter in TR_MODAL indicates the number of modes to be used in the transient response analysis. SPAR processor DR integrates the modal equations and performs the back transformation for selected physical coordinates.

If a transient response calculation by direct integration of the system equations is being performed, procedure TR_DIRECT is called, which in turn calls procedure NEWMARK. Procedure NEWMARK implements the well known Newmark integration method for second order, coupled systems. Parameters such as system stiffness and mass matrix names, the time step, and the total number of time steps in the analysis are formal arguments to procedure NEWMARK. In NEWMARK, extensive use is made of the CLAMP macro expression capability for calculating integration constants and controlling the algorithm. The initial acceleration at time t = 0 is calculated from the given initial displacement and velocity vectors. This is done by using processor AUS to set up the equations of motion at t=0, and INV and SSOL to solve for the acceleration. At each subsequent time step, processor AUS is used to set up the recursion relations, and processor SSOL is used to solve for the displacement vector at the next time step. Then velocity and acceleration vectors can be calculated and selectively printed. Although procedure NEWMARK is not intended as a "production" quality implementation of the Newmark method, it does illustrate many of the features and the potential of NICE/SPAR procedures.

PERFORMANCE

To assess the computational performance of NICE/SPAR, the two example problems were executed under both NICE/SPAR and EAL. A processor-by-processor comparison of execution time (CP TIME) and number of disk read or write operations (I/O) is presented in Tables 1 and 2. The number of disk I/O operations is an important statistic because this can have a substantial effect on overall elapsed execution time for an analysis. Accordingly, on computers with fast central processors, the amount of disk activity is weighted very heavily in the overall costing algorithm for executing jobs.

Consistently, NICE/SPAR requires substantially more computational resources than does EAL. As shown in Tables 1 and 2, the ratios for CP time range from about 1.0 to nearly 13.0 and I/O operations range from about .9 to over 22.0. Where possible, EAL processors are used which have capabilities similar to their SPAR ancestors. However, several "workhorse" processors such as TOPO and INV have been completely redesigned in EAL and some of the performance difference between the two systems can be attributed to this fact. The relative performance of NICE/SPAR in executing example 1 is better than for example 2. This is true because example 1 is a simple sequential analysis which places minimal demands on the NICE architecture. Example 2 involves many executions of the AUS and SSOL processors which results in a substantial overhead penalty in NICE/SPAR. Future work on NICE/SPAR will be aimed at improving the performance of the system.

Concluding Remarks

Construction of a new structural analysis system based on the original SPAR finite element code and the NICE system has been described. This work was motivated by requirements for a modular and flexible structural analysis system to use as a tool in carrying out research in computational methods and exploring new computer hardware. Installation was reasonably straightforward because of similar concepts in software architecture between SPAR and NICE.

Example problems have been executed and are presented to verify the correctness and performance of the system and to demonstrate the utility of the NICE/CLAMP command language combined with the SPAR processors. The utility and correctness of the system have been verified; however the computational performance in terms of both CP time and direct disk I/O of NICE/SPAR has been found to be poor compared with EAL. Work is currently underway to improve the performance and to enhance the functionality of the system.

The NICE/SPAR procedure for direct integration of the transient response equations demonstrates the power of combining a high level command language (CLAMP) with a set of modular, command-driven computational processors. With this concept, for example, numerous time integration algorithms could be studied without any changes to the low level FORTRAN code. Many other research activities in computational structural mechanics should lend themselves equally well to this approach.

Table 1. Performance Comparison Between EAL and NICE/SPAR for the Composite Toroidal Shell Example

PROCESSOR	EAL CP TIME (secs)	EAL I/O (counts)	NICE/SPAR CP TIME (secs)	NICE/SPAR I/O (counts)	I/O CP TIME RATIO	I/O OPERATION RATIO
U1	5.2	45	-	-	-	-
TAB	5.0	60	5.5	223	1.1	3.7
DCU	.4	5	.8	26	2.3	5.2
AUS	.1	17	.7	50	5.0	2.9
ELD	3.0	60	3.1	246	1.0	4.1
E	4.7	172	5.3	245	1.1	1.4
EKS	33.9	106	85.8	85	2.5	.8
TOPO	3.5	209	10.8	519	3.0	2.5
K	24.2	1090	25.9	985	1.1	.9
INV	134.7	743	243.9	886	1.8	1.2
SSOL	11.9	344	21.4	471	1.8	1.4
AUS	.5	51	1.5	95	3.0	1.9
DCU	.4	25	.9	37	2.3	1.5
GSF	1.1	109	2.7	116	2.4	1.1
PSF	12.4	57	11.9	535	1.0	9.4
PSF	1.2	51	3.2	171	2.7	3.4
PSF	1.0	51	3.2	168	3.2	3.3
VPRT	3.7	63	4.5	148	1.2	2.3
DCU	.5	26	.8	19	1.7	.7
TOTAL	247.3	3284	431.9	5025	Average 1.7	Average 1.5

Table 2. Performance Comparison Between EAL and NICE/SPAR for the Cantilever Beam Example

Processor	No. of Executions	EAL CP Time (Secs)	EAL I/O (Counts)	NICE/SPAR CP Time (Secs)	NICE/SPAR I/O (Counts)	CP Time Ratio	IO Op. Ratio
AUS	114	96.6	7948	1246.4	62191	12.9	7.8
DCU	3	.9	36	2.2	127	2.4	3.5
E	1	.4	19	1.3	100	3.3	5.3
EKS	1	.1	5	.4	32	4.0	6.4
ELD	1	.2	11	1.6	147	8.0	13.4
INV	3	1.1	77	1.8	151	1.6	2.0
K	1	.2	22	.6	70	3.0	3.2
M	1	.2	19	.8	77	4.0	4.1
SSOL	102	48.8	4134	417.0	19583	8.5	4.7
TAB	1	1.0	19	8.0	421	8.0	22.2
TOPO	1	.6	31	2.2	80	3.7	2.6
U1	2	5.9	54	-	-	-	-
VPRT	11	2.1	113	11.4	549	5.4	4.9
						Avg.	Avg.
	242	158.1	12488	1693.7	83528	10.7	6.7

Figure 1. NICE/SPAR Input for the Composite Toroidal Shell Example

```
$!
$! NICE/SPAR DEMONSTRATION PROBLEM 13
$! COMPOSITE TOROIDAL SHELL
$!
$ SET VERIFY
$ SET DEF NICESPAR$DEMO
$ nicespar
*set echo=off
*open 1, demo13.101 /new
[XQT TAB
  ONLINE=0
  START 337
 title' composite toroidal shell
  JLOC; FORMAT=2
   2 650.0125 0. 0.       650.0125  5.2888 0.       21  16
   3 650.1866 0. -.8754   650.1866  5.2888 -.8754   21  16
   4 650.1866 0. +.8754   650.1866  5.2888 +.8754   21  16
   5 650.6825 0. -1.6175  650.6825  5.2888 -1.6175  21  16
   6 650.6825 0. +1.6175  650.6825  5.2888 +1.6175  21  16
   7 651.4246 0. -2.1134  651.4246  5.2888 -2.1134  21  16
   8 651.4246 0. +2.1134  651.4246  5.2888 +2.1134  21  16
   9 652.3    0. -2.2875  652.3     5.2888 -2.2875  21  16
  10 652.3    0. +2.2875  652.3     5.2888 +2.2875  21  16
  11 653.1754 0. -2.1134  653.1754  5.2888 -2.1134  21  16
  12 653.1754 0. +2.1134  653.1754  5.2888 +2.1134  21  16
  13 653.9175 0. -1.6175  653.9175  5.2888 -1.6175  21  16
  14 653.9175 0. +1.6175  653.9175  5.2888 +1.6175  21  16
  15 654.4134 0. -.8754   654.4134  5.2888 -.8754   21  16
  16 654.4134 0. +.8754   654.4134  5.2888 +.8754   21  16
  17 654.5875 0. 0.       654.5875  5.2888 0.       21  16
   1 652.3    5.2888  0.
  MATC: 1 .114+07 0.28
  BA:  DSY 1 .675-03 0. .675-03 0. .09 .270-02  :  .
  MREF: 1 1 2 1 .99574
  JREF: NREF=-1: 1,337
  CON=1: FIXED PLANE=2
  SA(4)
     FORMAT=laminate: 1 .   4 LAYER COMPOSITE
        -9.375-03 90. .00625> .   LAYER 1, INSIDE SURFACE
        1.8560+05 2.0010+03 7.1470+03 0. 0. 4.0620+03>
        6.0400-01 6.5140-03 2.3260-02 0. 0. 1.3220-02
        -3.125-03 0.0 .00625> .   LAYER 2
        1.8560+05 2.0010+03 7.1470+03 0. 0. 4.0620+03>
        6.0400-01 6.5140-03 2.3260-02 0. 0. 1.3220-02
        3.125-03 45. .00625> .    LAYER 3
        1.8560+05 2.0010+03 7.1470+03 0. 0. 4.0620+03>
        6.0400-01 6.5140-03 2.3260-02 0. 0. 1.3220-02
        9.375-03 -45. .00625> .    LAYER 4, OUTSIDE SURFACE
        1.8560+05 2.0010+03 7.1470+03 0. 0. 4.0620+03>
        6.0400-01 6.5140-03 2.3260-02 0. 0. 1.3220-02
```

Figure 1. Continued

```
2 .  4 LAYER COMPOSITE DIFFERENT INPUT FORMAT
-.009375 90. .00625 185600. 2001. 7147. 0. 0. 4062. .604  .0065 .023 0. 0.
.0132
-.003125 0.0 .00625 185600. 2001. 7147. 0. 0. 4062. .604  .0065 .023 0. 0.
.0132
 .003125 45. .00625 185600. 2001. 7147. 0. 0. 4062. .604  .0065 .023 0. 0.
.0132
.009375 -45. .00625 185600. 2001. 7147. 0. 0. 4062. .604  .0065 .023 0. 0.
.0132
3 .  4 LAYER COMPOSITE DIFFERENT INPUT FORMAT AND VALUES
  -9.375-3  90. .00625 1.856+5 2.001+3 7.147+3  0. 0. 4.062+3
   -3.125-3 0. .00625 1.856+5 2.001+3 7.147+3  0. 0. 4.062+3
  3.125-3 45. .00625 1.856+5 2.001+3 7.147+3 0. 0. 4.062+3
  9.375-3 -45. .00625 1.856+5 2.001+3 7.147+3 0. 0. 4.062+3
[XQT DCU .
PRINT 1 SA .
[XQT AUS
  SYSVEC: APPLIED FORCES 1
     CASE 1: I=3: J=1: 1.0
     CASE 2: I=2: J=1: 322,337: 0.058824
  ALPHA: CASE TITLE 1
     1' TRANSVERSE SHEAR LOAD
     2' AXIAL LOAD
[XQT ELD
  ONLINE=0
  E43
  GROUP 1' 0 TO 22.5 DEG.
       2  18   19 3  1 20 1
  GROUP 2' 22.5 TO 180 DEG.
       3  19   21 5  1 20 7
  GROUP 3' 180 TO 202.5 DEG.
       17 33   32 16 1 20 1
  GROUP 4' 202.5 TO 360 DEG.
       16 32   30 14 1 20 7
  E21: 1  322  3  16 1
 ONLINE=1
[XQT E
T= .1-19,-.001,.0001,.0001,20.,.0001,.0001,.0001
[XQT EKS
[XQT TOPO
[XQT K
[XQT INV
 ONLINE=2
[XQT SSOL
[XQT AUS
  DEFINE D=STAT DISP
  DEFINE R=STAT REAC
  GLOB DISP=LTOG(D)
  GLOB REAC=LTOG(R)
[XQT DCU
  TITLE 1' 337 JOINT COMPOSITE TOROIDAL SHELL
  TOC 1
```

Figure 1. Concluded

```
[XQT GSF
  E43: 1: 3 .
[XQT PSF
[XQT PSF
  RESET DISP=2, CROS=0, NODES=0
  DIV=1. .001 .001 1.
[XQT PSF
  RESET DISP=3, CROS=0, NODES=0
  DIV=1. .001 .001 1.
[XQT VPRT
 JOINTS=2,322,16: 9,329,16: 17,337,16: 10,330,16 .
  TPRINT STAT DISP
  TPRINT GLOB DISP
 JOINTS=2,17
  TPRINT STAT REAC
[XQT DCU
 TOC 1
[xqt exit
```

Figure 2. Clamp Procedures for Transient Response Analysis of a Cantilever Beam

```
$ SET VERIFY
$ set def ns$demo
$ del cbeam.101;*,cbeam.102;*,ns.*;*,cbeam.128;*
$ nicespar
*set echo off
*set plib = 28
*open 28 cbeam.128
*open  1 cbeam.101

*def/i jt = 11
*procedure CANT_BEAM
[xqt tab
 start <jt> 3,4,5
 jloc
 1 0. 0. 0.    25. 0. 0. <jt>
 matc
 1 10.+6 .3 .101
 ba
    rect 1 1.0 .1
 mref
 1 1 2 1 1.0
 con 1
 zero 1,2,6 : 1
 con 2
 zero 1,2,6 : 1
 nonzero 2 : <jt>
[xqt eld
 e21
*def/i jtm1 = <<jt> - 1>
 1 2 1 <jtm1>
[xqt e
[xqt eks
[xqt topo
[xqt k
[xqt m
 reset g=386.
.
. compute initial displacement due to a static end load
.
[xqt aus
 sysvec : appl moti
 i=2 : j=<jt> : -1.0
[xqt inv
 reset con=2
[xqt ssol
 reset con=2
[xqt dcu
 change 1 stat disp 1 2 u0 aus 1 1
[xqt dcu
 toc 1
*end
```

Figure 2. Continued

```
*procedure VIBR_MODES (nmodes)
.
. computes "nmodes" vibration modes
.
*def/i nmodes = [nmodes]
[xqt inv
*def init = <min(<2*<nmodes>> ; <<nmodes> + 8>)>
[xqt eig
 reset init=<init>, nreq=<nmodes>, m=cem
[xqt vprt
 vectors = 1, <nmodes>
 print vibr modes
[xqt dcu
 toc 1
*end


*procedure TR_MODAL (nmodes)
.
. performs transient response analysis (modal superposition)
.
*def/i nmodes = [nmodes]
[xqt aus
  define x = vibr mode 1 1 1,<nmodes>
  define e = vibr eval
. compute modal initial displacements
  define id = 1 u0 aus 1 1
  idm = prod(cem, id)
  iqx = xty(x,idm)
  table(nj=<nmodes>) : xtmx : j=1,<nmodes> : 1.0
  table(nj=<nmodes>) : xtkx : transfer(source=e)
  table(ni=1,nj=<nmodes>) : td
*def/i sbase = <(<jt> - 1)*3 + 1>
*show/macro sbase
  transfer(source=x, sbase=<sbase>, ilim=1)
[xqt dr
  dtex(dt=.001)
  tr1(qxlib=1,qx1lib=1,t1=0.0,t2=.12)
  back
   t = td : y = qx
   z = zd aus
[xqt dcu
  toc 1
  print 1 iqx
  print 1 td
  print 1 zd
*end
```

Figure 2. Continued

```
*procedure TR_DIRECT
.
. performs transient response analysis by direct integration
. of the equations of motion
.
[xqt aus
  sysvec : ud0 . initial velocities = 0
    i = 1 : j = 1 : 0.0
*open 2 cbeam.102
*call NEWMARK (mname = cem; delt = .001; nstep = 100; pfreq = 10)
*end

*procedure NEWMARK (                --
                    kname = k  ; -- . first name of global k
                    mname = dem; -- . first name of global m
                    beta  = .25; --
                    gamma = .50; --
                    delt =  0.0; -- . time step
                    nstep      ; -- . number of time steps
                    slib  =   2; -- . number for temp. library
                    pfreq =   1  -- . print frequency for results
                   )
.
. Performs dynamic analysis on a linear system using the
. Newmark-Beta implicit integration method
.
.
. Initialization
.
*def/a kname = [kname]
*def/a mname = [mname]
*def/e beta  = [beta]
*def/e gamma = [gamma]
*def/e delt  = [delt]
*def/i nstep = [nstep]
*def/i slib  = [slib]
*if <delt> /eq 0.0 /then
   *remark error: time step (delt) = 0.0
   *stop
*endif
*def/e a0 = (1.0/<beta>/<delt>/<delt>)
*def/e a1 = (<gamma>/<beta>/<delt>)
*def/e a2 = (1.0/<beta>/<delt>)
*def/e a3 = (1.0/2.0/<beta> - 1.0)
*def/e a4 = (<gamma>/<beta> - 1.0)
*def/e a5 = (<<gamma>/<beta> - 2.0>*<delt>/2.0)
*def/e a6 = (<1.0 - <gamma>>*<delt>)
*def/e a7 = (<gamma>*<delt>)
*def/e ma2 = <-<a2>>
*def/e ma3 = <-<a3>>
*show macro
```

Figure 2. Continued

```
.
[xqt aus
 khat = sum(<kname>, <a0> <mname>)
.
. calculate initial acceleration vector
.
 inlib = 3 : outlib = 3
 define k = 1 <kname>
 define u0 = 1 u0
 appl forc 1 = prod(k, -1.0 u0)
[xqt inv
 reset k = <mname>, kilib=3, dzero = 1.e-9
[xqt ssol
 reset k=<mname>, kilib=3, qlib=3, reac=0
[xqt inv
 reset k=khat
[xqt dcu
 copy 1, <slib> u0
 copy 1, <slib> ud0
 copy 3, <slib> stat disp
 change <slib> u0 mask mask mask stat disp 0 1
 change <slib> ud0 mask mask mask ud vec 0 1
 change <slib> stat mask 1 1 udd vec 0 1
 toc 1
 toc <slib>
*close 3 /delete
[xqt vprt
 lib = <slib>
 format = 4
 print stat disp 0 ' initial displacement vector
 print ud vec 0 ' initial veclocity vector
 print udd vec 0 ' initial acceleration vector
.
. iterate for "nstep" time steps
.
[xqt aus
*def/i pcnt = 1
*do $step = 0,<nstep>
  inlib = 21 : outlib = 21
  define u = <slib> stat disp <$step>
  define ud = <slib> ud vec <$step>
  define udd = <slib> udd vec <$step>
  define m = 1 <mname>
.
  r1 = sum(<a0> u     <a2> ud)
  r2 = sum(<a3> udd        r1)
*def/i stp1 =  <<$step> + 1>
  outlib = <slib>
   applied force <stp1> = prod(m, r2)
[xqt ssol
  reset k=khat, set=<stp1>, qlib=<slib>, reac=0
```

Figure 2. Concluded

```
[xqt aus
  inlib = 21 : outlib = 21
  define utdt = <slib> stat disp <stp1>
  define ut   = <slib> stat disp <$step>
  define udt  = <slib> ud   vec  <$step>
  define uddt = <slib> udd  vec  <$step>
  u1 = sum(utdt -1.0 ut)
  u2 = sum(<a0> u1 <ma2> udt)
  u3 = sum(udt <a6> uddt)
   outlib = <slib>
   udd vec <stp1> = sum(u2, <ma3> uddt)
  define utt = <slib> udd vec <stp1>
   ud vec <stp1> = sum(u3 <a7> utt)
*show/macro pcnt
*if <<pcnt> /eq [pfreq]> /then
.
. print every pfreq'th solution
.
[xqt vprt
 lib = <slib>
 format = 4
 print stat disp <stp1> ' displacement vector
*def/i pcnt = 1
[xqt aus
*else
*def/i pcnt = <<pcnt> + 1>
*endif
*enddo
*end


*call CANT_BEAM
#call VIBR_MODES (nmodes=4)
#call TR_MODAL (nmodes=4)
*call TR_DIRECT
[xqt exit
```

Appendix A. NICE/SPAR Usage

On the CSM VAX/VMS computer system, the NICE/SPAR executive is invoked by typing NICESPAR in the interactive mode. The command used to invoke a NICE/SPAR processor is "[XQT processor-name"; the command to exit a processor and the NICE/SPAR executive is "[XQT EXIT". NICE directives (prefixed by *) may be entered, intermixed with SPAR commands. Batch mode processing is also available, with all commands and directives supplied from a disk file.

Because NICE converts all input (except labels) to uppercase characters, which SPAR requires, raw input data may be entered in either upper or lower case.

NICE directives are documented in reference 2. SPAR commands are documented in reference 3. Differences between NICE/SPAR and the documented version of SPAR are described below.

Modifications to SPAR Reference Manual For NICE/SPAR Usage

The section numbers below refer to the sections in reference 3 to which the modifications apply.

2.2 The Data Complex

By default, the file names corresponding to NICE/SPAR libraries are formed by appending the extension Lxx to a root file name ("NS" currently) where xx is the library number (i.e., NS.L01 for library 1).

The table of contents (TOC) is maintained by the NICE data manager in a different format than the SPAR TOC. The NICE/SPAR TOC items displayed by DCU are: sequence no., date, time, lock code, no. of records, name of creating processor, data set name. Other items in the SPAR TOC which are required by the processors (data set length, record length, no. of columns per block and data type) are obtained in NICE/SPAR via GAL record level utilities.

2.3 Card Input Rules

The same input rules are followed except:

1) Real data input may contain "E" at the beginning of the exponent field as in FORTRAN.

2) The comment character is # instead of $.

2.5 Data Set Structure

The SPAR data set structure is followed except:

1) NWORDS is always an integral multiple of NI*NJ.

2) In most cases, one SPAR block corresponds to a single NICE record. However, in some data sets a SPAR block corresponds to a NICE record group, where an individual NICE record corresponds to a segment of the SPAR block.

2.5.1 Table

Tables can be of any SPAR data type; tables with ITYPE = ± 1 may not contain mixed data, but ITYPE = 0 tables may contain values of integer, real or alphanumeric type.

4.2 K - The Stiffness Matrix Assembler

Under RESET controls, the default for SPDP is 2, so double precision output is obtained because of the smaller word size on VAX.

5.1 AUS

5.1.3.1 TABLE

The command line is:

TABLE,U(NI = ni, NJ = nj, ITYPE = n): N1 N2 n3 n4: data...

where the optional parameter ITYPE has been added, being the SPAR data type code of the data set.

The footnote should read:

* Loop-limit format is permitted for ITYPE = ± 1 only. It is not permitted for ITYPE = 0 or 4.

5.2 DCU - The Data Complex Utility Program

The following commands are not implemented in the current version of NICE/SPAR:

XCOPY, XLOAD, REWIND, TWRITE, TREAD, NTAPE, STORE, RETRIEVE

Section 8. EIG - Sparse Matrix Eigensolver

Instead of using the Cholesky-Householder method for solving the low-order eigenproblem (4), the combination shift QZ algorithm described in reference 8 is used.

Appendix B. NICE/SPAR User Interface Subroutine Descriptions

Subroutine DAL (NU, IOP, KA, KORE, IEA, KADR, IERR, NWDS, NE, LB, ITYPE, NAME1, NAME2, NAME3, NAME4)

Purpose: Read or write a nominal data set named NAME1.NAME2.NAME3.NAME4 in library NU

Parameters:

NU	library number (integer, input)
IOP	operation code (integer, input) = - 1, Rename current data set; set KADR to data set sequence number = 0, Set up an entry in TOC for new data set; disable old data sets of same name; set KADR to data set sequence number = 1, Same as 0 but also write one record of data from KA. = 2, Same as 1 but does not disable old data sets. = 10, Get TOC information without reading data; set IERR if not found. = 11, Same as 10 but also read one record (LB items) of data into KA.
KA	initial address of array containing data to be read or written; actual data type depends on ITYPE. (input for write operation, output for read operation)
KORE	number of words available for data set (integer, input) If LB>KORE and IOP>9, IERR set to 2. If KORE = 0, the check for space is skipped.
IEA	error condition check code (integer, input) = 1, Print message and return if error encountered. = 2, Disregard error. other, Print message and abort.
KADR	Data set sequence number, = 0 if not found. (integer, output)
IERR	error code on return (integer, output) = 0, No error = 1, data set not found = - 2, Insufficient space for data set
NWDS	number of words in data set (integer, input for write, output for read)
NE	number of columns per block (integer, input for write, output for read)
LB	record size (integer, input for write, output for read)
ITYPE	SPAR data type code (integer,input for write,output for read) = 0 integer data = ± 1 real data = ± 2 double precision data = 4 alphanumeric data
NAME1	1st component of data set name, 4 bytes (alphanumeric, input)
NAME2	2nd component of data set name, 4 bytes (alphanumeric, input)
NAME3	3rd component of data set name (integer, input)
NAME4	4th component of data set name (integer, input) (any component of the data set name may be 4HMASK which is a "wildcard" matching parameter)

Functional description:

1. Library NU is checked to be open; if not, it is opened as a GAL82 library on disk file NS.Lxx, where xx is the library number.
2. For writing, the name is entered in TOC via GMPUNT. KADR is set to the data set sequence number of the new data set. If IOP = 1, one record of LB words of the appropriate type are written via GMPUTN or GMPUTC. If IOP = 2, old data sets are enabled.
3. For reading, the data set sequence number is located and the matched data set name components are returned in common block /TOCLIN/. The data set length, record length, rowsize, and data type are returned in /TOCLIN/ and in the argument list. If IOP = 11, the available space (KORE) is checked and one record of data is read via GMGETN or GMGETC.
4. For IOP = - 1, the current data set is renamed. No other parameters can be changed at this time in NICE/SPAR.

Subroutine FIN (NERR, NER)

Purpose: Terminate NICE/SPAR processor.

Parameters:

NERR	error code (integer, input) = 0 , no error ≠ 0 , 4-byte alphanumeric error code to be printed (A4)
NER	error no. to be printed if NERR = 0 (I10 format) (integer, input)

Functional description:

1. Close libraries 1-20 and 27-30 conditionally; close libraries 21-26 unconditionally.
2. Print execution statistics (CPU, clock time, buffered I/O, direct I/O).
3. Print error messages according to input parameters.
4. Chain to NICE/SPAR executive via CLPUT.

Subroutine INTRO (IDPROC)

Purpose: Log processor name with data manager and get unit assignment for printed output file.

Parameter:

IDPROC - Processor name, in upper case (input, character*4)

Function description:

1. Call GMSIGN to enter processor name to be "signed" into data sets created by the processor.

2. Call ICLUNT to get the unit number assigned to the print file and assign this valve to the second integer variable in common block /IANDO/. This variable is used by NICE/SPAR processors for normal output.

Subroutine LTOC (NU, J, NAME1, NAME2, NAME3, NAME4)

Purpose: To get item from TOC

Parameters:

NU	library number (integer, input)
J	TOC item number desired, 1-12 (integer, input)
NAME1	1st component of data set name, 4 bytes (alphanumeric, input)
NAME2	2nd component of data set name, 4 bytes (alphanumeric, input)
NAME3	3rd component of data set name (integer, input)
NAME4	4th component of data set name (integer, input)

Functional description:

1. Find data set in NICE TOC via LMFIND.
2. Get TOC information via GMGENT, GMGETN, and LMRECS.
3. Set function value to the value of the desired item.

The TOC items are:

1	Data set sequence no.
2-4	Unused
5	Number of words in data set.
6	Number of columns per block (for matrix type data)
7	Record size
8	SPAR data type code
9	1st component of data set name
10	2nd component of data set name
11	3rd component of data set name
12	4th component of data set name

Subroutine READ (IA, IEOF)

Purpose: Get one unparsed user input record from NICE

Parameters:

IA	input record contents, array of 80 words, one character of input per word (integer, output)
IEOF	end of input flag (integer, output) = 0, successful input operation = 1, no input obtained

Functional description:

1. Initialize IA to blank.
2. Call CLGET to get an input record using the 4 character prompt string from common block /PID/. All macro expansions in the record have been performed by CLGET prior to return.
3. Store individual characters from input record into IA, one character per word, left justified.

Subroutine READER

Purpose: Get one line of user input and parse it according to the SPAR command input syntax. Input data items are stored in common block /INREC/.

Functional description:

1. Call READ to get user input record via CLGET.
2. Parse the input according to the SPAR input syntax specified in reference 3, with modifications described in Appendix A. Up to 40 items per record are allowed.
3. Return data items in common block /INREC/ described below.
4. If first item is "FIN ", call subroutine FIN to terminate the processor.
5. If the first item is "RUN " or "[XQT", set NAME to "STOP". NICE/SPAR processors use this value of NAME as the end-of-input flag.

Common block /INREC/ contents:

IDATA(40)	Parsed input data items; actual data stored in IDATA may be of integer, real, or alphanumeric type.
KIND(40)	Integer SPAR data types of corresponding words in IDATA.
NAME	Alphanumeric command key; set to IDATA(1) if KIND(1) = 4 (alphanumeric); set to "STOP" if IDATA(1) = "RUN " or "[XQT"; otherwise = 0.
NA41	Integer index in IDATA where alphanumeric label begins, if a label is included in this record.
NA42	Integer index in IDATA where alphanumeric label ends.

Subroutine RIO (NU, IWR, IOP, IDSN, KSHFT, KSHFT2, KA, L, ITYPE, NE)

Purpose: Read or write named records to NICE nominal data set.

Parameters:

NU	Library number (integer, input)
IWR	Operation code (integer, input) = 1, Write records KSHFT:KSHFT2 = 2, Read records KSHFT:KSHFT2 = 10, Write records KSHFT:KSHFT2 and return next record number in KSHFT. = 20, Read records KSHFT:KSHFT2 and return next record number in KSHFT.
IOP	record location code (integer, input) = 1 or 2, Read or write records KSHFT:KSHFT2 = 3, Append records to end of data set
IDSN	data set sequence number (integer, input)
KSHFT	initial record number to be accessed (integer, input always, output if IWR>9)
KSHFT2	final record number to be accessed (integer, input)
KA	initial address of array of data (data type depends on ITYPE input for write operation, output for read operation)
L	number of data items to be read or written (integer, input)
ITYPE	SPAR data type code (integer, input)
NE	number of columns per block (integer, input for write, dummy argument for read)

Functional description:

1) Get NICE data type code from ITYPE. Construct NICE record name.
2) For IOP = 1 or 2, if IWR=1 or 10, write records via GMPUTN or GMPUTC; if IWR = 2 or 20, read record via GMGETN or GMGETC. For IOP = 3, get number of records written on data set; construct record name DATA.nrec+1:nrec+(KSHFT2-KSHFT); write records via GMPUTN or GMPUTC.

3) For IWR > 9, return next record number in KSHFT.

Subroutine TOCO (NU, NAME, IOP, NLINE)

Purpose: Find data set and return TOC information

Parameters:

NU	Library no. (integer, input)
NAME	4 word array, data set name (alphanumeric and integer, input)
IOP	operation code (integer, input) = 1, find first matching data set ≠ 1, Disable all matching data sets after entry NLINE
NLINE	data set sequence number to start search at (integer, input, output if IOP = 1)

Functional description:

1. Find all matching data sets via GMATCH; if no matches found, set NLINE = - 1 and return.
2. For IOP = 1, set NLINE = seq. no. of first matching data set after input NLINE. Get TOC information via GMGENT, GMGETN, LMRECS. Return TOC information in common block /TOCLIN/.
3. For IOP ≠ 1, disable matching data sets after NLINE. Set NLINE to the number of disabled data sets.

Appendix C. NICE/SPAR Processor Integration Guidelines

A major goal of the NICE/SPAR system is to provide mechanisms for easy interface of user-supplied computational processors.Processors can either be written completely from "scratch" or existing processors can be modified. The most critical issue faced by the developer or modifier of a processor is maintaining compatibility with existing processors.

Compatibility with the system is insured by making the input and output data structures (data sets) "match" those of other processors and making the capabilites of the new or replacement processor complement those of existing processors. The integral structure of many NICE/SPAR data sets is described in reference 6. If the intent of a processor developer is to completely replace an existing processor, then typically both the input and output data sets and capabilites would at least be equal to the replaced processor. Processors developed to merely augment existing processors do not have so rigorous a requirement. As long as their input and output data sets agree with those of processors with which they interact, compatibility is assured.

Sometimes this is easy and sometimes more difficult. A couple of examples serve to illustrate these extremes. The first example is the addition of a new, special-purpose processor to do interactive plotting of the geometry of a finite element model. It would be necessary for this processor to read the data sets containing joint locations and connectivity information for different elements. These data sets are relatively simple to access. And since no output data sets need be produced, no compatibility problem is introduced. A more difficult processor development task would be the replacement of the existing, system matrix factorization processor, INV. SPAR uses a storage scheme for system matrices that involves storing only the non-zero blocks in the upper half of the (assumed) symmetric matrix. Processor TOPO performs the complicated task of determining the necessary information required for assembly and factorization of these matrices. A capability compatible with TOPO would not be easy to produce. Finally, the data set output by INV has a special form and several processors in the system expect this particular form. As a result, development of a new INV with identical input and output data sets would require careful study. The alternative of replacement of the basic system matrix data structures would have an impact on many processors in the code. Consequently the effect of this type of change on computational efficiency, generality, and extendablity would have to be carefully considered.

Steps in Processor Integration

1. The name of the processor should be no longer than 4 characters; this should be the name of the source file with the extension ".FOR". This name must not be one of the installed processor names listed in Appendix D.

2. The processor should begin with a FORTRAN main program. It should include a common block named /PID/ with a CHARACTER*4 variable to which the processor name should be assigned at the beginning of the processor execution. A call to subroutine INTRO (Appendix B) with the processor name as the only argument should also be included at the beginning of executable statements in the processor.

3. The common block /IANDO/ with 2 integer variables containing user input and output unit numbers should be included in any subroutine which uses FORTRAN write statements. The unit numbers are assigned in the subroutine INTRO. All write statements should refer to the above output unit.

4. The NICE CLAMP utilities (ref. 2) or the NICE/SPAR subroutine READER (Appendix B) should be used to process command input; the READER routine uses the NICE CLGET routine to filter NICE commands and interprets user input according to the SPAR command format.

5. The NICE/SPAR data handling utilities DAL and RIO (Appendix B) may be used to interact with the database. As an alternative, NICE data manager calls (ref. 2) can be used directly.

6. The processor should terminate by calling subroutine FIN (Appendix B) which closes data libraries and returns control to the NICE/SPAR executive.

NICE/SPAR Processor Integration on a VAX/VMS System

On the CSM VAX/VMS system, the following logical names are defined to assist NICE/SPAR integration:

NS$SRC	-	Directory containing NICE/SPAR source files
NS$OBJ	-	Directory containing NICE/SPAR object and library files
NS$LIS	-	Directory containing NICE/SPAR listing files
NS$EXE	-	Directory containing NICE/SPAR executable files.
NS$DEMO	-	Directory containing demonstration problem files.

To interface a new processor to NICE/SPAR, the following steps should be performed:

1. The source version of the processor should be written according to the above guidelines. This source file may reside in one of the user's directories.

2. The default directory should be set to the directory where the source processor resides. The processor should be compiled and linked using the command

   ```
   @NS$SRC:BLDEXTP processor-name
   ```

   Executing this command procedure will create a listing file and an executable file in the default directory.

3. The processor can be executed in the NICE/SPAR environment as follows:
   a) The default directory should be set to the one where the experimental processor and NICE/SPAR data libraries reside.
   b) Type NICESPAR to invoke the executive program.
   c) Enter optional NICE/SPAR directives.
   d) Enter the NICE/SPAR directive "[XQT processor-name" to start execution of the processor.

An Example NICE/SPAR Processor

The following is a listing of a FORTRAN program which interfaces with NICE/SPAR according to the above guidelines. The program is similar to the one given in reference 5, Appendix E, but was modified where required for NICE/SPAR compatibility as noted in the listing. The test processor performs these simple operations:

1. A list of real numbers is read from input.
2. A library is opened, and a data set is created containing these values.
3. A matrix of integer values is read from input.
4. A second data set is created on the library containing the matrix values.
5. Both data sets are then read from the library and the values are printed.

```
      program TEST
c
c   /INREC/ is used by READER to return parsed input items
c
      common/INREC/idata(40),kind(40),name,ndw,na41,na42,nrpr,ichar(81)
      dimension cdata(40)
      equivalence (idata(1),cdata(1))
c
c   /TOCLIN/ is used by DAL, LTOC and TOCO to return table of contents
c   information about the requested data set
c
      common/TOCLIN/line(12)
c
c   Define blank common (different from Giles version)
      common kore,keven,kort,a(40000)
      dimension ka(1)
      equivalence (ka,a)
c
c   /IANDO/ is used by NICE/SPAR for accessing unit nos. for
c   input and output files
c
      common/IANDO/iin,ioutx
c
c   /PID/ is used by NICE/SPAR for processor identification
c
      character*4 idproc
      common/PID/idproc
c
c   IV is the table of RESET parameters for this processor
c
      integer vgo1,vgo2
      dimension na4(4)
      dimension iv(3,4),rv(3,4)
      equivalence (iv(1,1),rv(1,1))
      data nl/4/
      data iv/
```

```
     14HNLIB, 0, 1,
     24HNRV , 0, 0,
     34HNRM , 0, 0,
     44HNCM , 0, 0 /
c
c    Define the processor name for NICE/SPAR
c
      idproc = 'TEST'
      call intro(idproc)
c
c    Read reset values and first data card
c
      call RSET( iv, nl, 0 )
      nu = iv(3,1)
      nrowv = iv(3,2)
      nrowm = iv(3,3)
      ncolm = iv(3,4)
c
c    Set up initial addresses of data blocks relative to the beginning of
c    A in blank common
c
      vgo1 = 1
      mgo1 = vgo1 + nrowv
      vgo2 = mgo1+nrowm*ncolm
      mgo2 = vgo2+nrowv
c
c    Put vector in blank common starting at A(VGO1)
c
      do 10 i=1,nrowv
      j = vgo1+i-1
      a(j) = cdata(i)
   10 continue
c
c    Put matrix in blank common starting at A(MGO1)
c
      do 20 i=1,ncolm
      k = mgo1+nrowm*(i-1)
      call READER
      do 20 j=1,nrowm
      l = k+j-1
      ka(l) = idata(j)
   20 continue
c
c    write vector to disk (note UPPERCASE data set names)
c
      call DAL(nu,1,a(vgo1),0,1,kadr,ierr,nrowv,1,nrowv,-1,
     1 4HTEST, 4HVEC , 1, 1)
c
c    write matrix on disk in blocked form
c
      nwds = nrowm*ncolm
      nj=1
      itype=0
      call DAL(nu,0,a,0,1,kadr,ierr,nwds,nj,nrowm,itype,
     1 4HTEST, 4HMAT , 1, 1)
```

```
      do 30 i=1,ncolm
      irec=i
      j = mgo1+nrowm*(i-1)
      call RIO(nu,1,2,kadr,irec,irec,ka(j),nrowm,itype,nj)
   30 continue
c
c   Read vector into blank common starting at A(VGO2)
c
      call DAL(nu,11,a(vgo2),0,iea,kadr,ierr,nwds,nj,lb,itype,
     1 4HTEST, 4HVEC , 1, 1)
      igo = vgo2
      iend = igo+nwds-1
      write(ioutx,40) (a(i),i=igo,iend)
   40 format(5x,f10.2)
c
c   Use TOCO to get information to read matrix
c
      na4(1) = 4HTEST
      na4(2) = 4HMAT
      na4(3) = 1
      na4(4) = 1
      nline=0
      call TOCO(nu,na4,1,nline)
      kadr = line(1)
      nwds = line(5)
      nj = line(6)
      lb = line(7)
      nblks = nwds/lb
      itype = line(8)
c
c   Read matrix into blank common starting at A(MGO2)
c
      call RIO(nu,2,2,kadr,1,nblks,ka(mgo2),nwds,itype,nj)
      igo = mgo2
      iend=igo+nwds-1
      write(ioutx,60) (ka(i),i=igo,iend)
   60 format(/5x,10i7)
c
c   Call READER to get next line of input for NICE/SPAR
c   (Note: processors are expected to read input data until a record
c          beginning with "[XQT " is encountered.)
c
   70 call READER
      if( name.ne.4hSTOP ) go to 70
c
c   Use FIN to close libraries and return to NICE/SPAR
c
      call FIN(0,0)
      end
```

A VAX/VMS command procedure to execute this program is as follows:

```
$ del ns.101;*
$ nicespar
[xqt test
. set sizes of vector and matrix input
reset nrv=5, nrm=10, ncm=3
. input vector data (real)
100. 200. 300. 400. 500.
. input matrix data (integer)
 1  2  3  4  5  6  7  8  9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
. call dcu to check library
[xqt dcu
toc 1
print 1 test vec
print 1 test mat
[xqt exit
```

Appendix D. NICE/SPAR Installed Analysis Modules

AUS - Arithmetic Utility System
CEIG - Complex Eigensolver
C3D - Constitutive Property Generator for the Solid (3-D) Elements
DCU - Data Complex Utility
DR - Linear Dynamic Response Analyzer
E - E-State Initiation
EIG - Sparse Matrix Eigensolver
EKS - Element Intrinsic Stiffness and Stress Matrix Generator
ELD - Element Definition Processor
EQNF - Equivalent Nodal Force Generator
GSF - Stress Data Generator
INV - SPAR Format Matrix Decomposition Processor
K - System Stiffness Matrix Assembler
KG - System Initial Stress (Geometric) Stiffness Matrix Assembler
M - System Consistent Mass Matrix Assembler
PAMA - AMAP Data set Printer
PKMA - KMAP Data set Printer
PS - SPAR Format Matrix Printer
PLTA - Plot Specification Generator
PLTB - Production of Graphical Displays
PRTE - EFIL Data set printer
PSF - Stress Table Printer
SM - System Modification Processor
SQ5 - Laminate Analysis for 2-D Elements
SSBT - Substructure Back Transformation
SSOL - Static Solution Generator
STRP - Substructure Eigensolver
SYN - System Synthesis
TAB - Basic Table Input
TOPO - Element Topology Analyzer
VPRT - Vector Printer

REFERENCES

1. Felippa, Carlos A.: Architecture of a Distributed Analysis Network for Computational Mechanics. Computers and Structures, Vol. 13, 1981, pp. 405-413.

2. Felippa, Carlos A.: A Command Language for Applied Mechanics Processors, Vols. 1-3, LMSC-D 78511, Nov. 1983.

3. Whetstone, W. D.: SPAR Structural Analysis System Reference Manual, Vol. 1; NASA CR 158970-1, Dec. 1978.

4. Whetstone, W. D.: EISI-EAL Engineering Analysis Language Reference Manual, Engineering Information Systems, Inc., July 1985.

5. Giles, Gary L.; and Haftka, Raphael: SPAR Data Handling Utilities. NASA TM 78701, 1978.

6. Cunningham, Sally: SPAR Data Set Contents, NASA TM 83181, Oct. 1981.

7. Felippa, Carlos A.: The Global Database Manager EZ-GAL, LMSC-D766995, Nov. 1982.

8. Ward, R. C.: An Extension of the QZ Algorithm for solving Generalized Matrix Eigenvalue Problems, NASA TND-7305, July 1973.

Standard Bibliographic Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-87586		

4. Title and Subtitle	5. Report Date
Experiences with a Preliminary NICE/SPAR Structural Analysis System	October 1985
	6. Performing Organization Code: 505-63-31-02
7. Author(s)	8. Performing Organization Report No.
Christine G. Lotts and William H. Greene	
9. Performing Organization Name and Address	10. Work Unit No.
NASA - Langley Research Center Hampton, VA 23665-5225	11. Contract or Grant No.
12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered: Technical Memorandum
National Aeronautics and Space Administration Washington, DC 20546-0001	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

Construction of a new structural analysis system based on the original SPAR finite element code and the NICE system is described. The system is denoted NICE/SPAR. NICE was developed at Lockheed Palo Alto Research Laboratory and contains data management utilities, a command language interpreter, and a command language definition for integrating engineering computational modules. SPAR is a system of programs used for finite element structural analysis developed for NASA by Engineering Information Systems, Inc. It includes many complementary structural analysis and utility functions which communicate through a common database. The work on NICE/SPAR was motivated by requirements for a highly modular and flexible structural analysis system to use as a tool in carrying out research in computational methods and exploring new computer hardware. Analysis examples are presented which demonstrate the benefits gained from a combination of the NICE command language with the SPAR computational modules.

17. Key Words (Suggested by Authors(s))	18. Distribution Statement
Structural analysis software Finite element analysis Finite element software	Unclassified - Unlimited Subject Category 39

19. Security Classif.(of this report)	20. Security Classif.(of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	38	A03

For sale by the National Technical Information Service, Springfield, Virginia 22161