53076
p-31

NASA Technical Memorandum 104166

USER'S GUIDE FOR NETS/PROSSS

James L. Rogers
William J. LaMarsh II

(NASA-TM-104166) USER'S GUIDE FOR NETS/PROSSS (NASA) 31 p CSCL 09B

N92-13690

Unclas
G3/61 0053076

October 1991

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

1. INTRODUCTION

Expensive analysis programs are often combined with optimization procedures to solve engineering problems. To obtain an optimal solution requires numerous iterations between the analysis program and the optimizer. This often becomes prohibitive due to the cost and amount of computer time needed to converge to an optimal solution. Therefore, any new software package that could significantly reduce the amount of computer time required to reach an optimal solution would be beneficial.

PROSSS (Programming System for Structural Synthesis, ref. 1,2,3) was developed several years ago to provide an open-ended system for coupling analysis and optimization. Although PROSSS was designed to handle any type of analysis program, most of the work has evolved around coupling a finite element structural analysis program with an optimization program. In NETS/PROSSS, the structural analysis program has been replaced by NETS (ref. 4), a neural network program developed at NASA Johnson Space Center, and coupled with CONMIN (ref. 5), an optimization program. The neural network approximates the results from the analysis program allowing the user to reach a near-optimal solution in much less time than before. The user can then use these results as a starting point in a normal optimization process and converge to an optimal solution with significantly fewer iterations.

The purpose of this paper is to serve as a user's guide for NETS/PROSSS. The key features include the neural network, determining the training pairs for the neural network, and the approximated analysis/optimization process. A small problem is given to serve as an example of how to apply the system.

2. THE NEURAL NETWORK

The user must first become familiar with the neural network program, NETS, which was developed at NASA Johnson Space Center and is available through COSMIC. NETS is an interactive, back propagation neural network simulator. The details for using NETS are given in the NETS user's guide (ref. 4).

After becoming familiar with NETS, the next step is to determine the format of the neural network. The neural network format consists of input nodes and output nodes, and should have hidden layer nodes. Hidden layers are not directly related to the input or the output of the network, but help perform the desired mapping between an input and its corresponding output to yield a better approximation. The input nodes typically consist of the design variables, while the output nodes consist of values for the constraint functions, behavior variables, and the objective function. The number of nodes to use in the hidden layer is still a research issue. A good first estimate is a number between the average and the sum of the number of nodes in the input and output layers. The neural network format is saved on a file and input to NETS with the **"c"** option in the NETS menu.

The neural network is trained with training pairs. Training pairs consist of known inputs and known outputs. The best results are obtained when these values have been scaled to be between 0.1 and 0.9. The training pairs are input to NETS with the **"i"** option in the NETS menu. More details on how to generate and scale training pairs appear in Section 3.

The training pairs are propagated through NETS using the **"t"** option in the NETS menu to generate a weight matrix connecting the input nodes to the output nodes. The weight matrix generated by training the neural network determines the functional relationship among the input and output nodes. Usually training is done to 1% error using the NETS default training values. This weight matrix can then be saved in a portable format with the **"s"** option in the NETS menu. The name ANALYSIS.PWT should be used for saving this file.

At this point, the user can generate a C routine for simulating the analysis program with the **"g"** option in the NETS menu. Details about this routine are in Section 4. NETS is then terminated with the **"q"** option in the NETS menu.

3. GENERATING TRAINING PAIRS

One of the keys to approximating the analysis results with a neural network is the generation of the training pairs. Training pairs consist of known inputs and known outputs. There are two ways to generate the training pairs. (1) Execute an analysis program several times with different design variables selected randomly over the range of design variables. (2) Begin the optimization process (use PROSSS) and save the analysis results from several cycles. Preliminary tests with the sample problem indicate that the second method generates a better set of training pairs if large move limits (30%) are used. (Note: If finite differences are used to generate gradients, each of the analysis results obtained with perturbed design variables can also be used as a training pair.)

It is very difficult to determine how many training pairs will be needed. For the sample problem described in section 6, both seven and thirty-two training pairs were used with good accuracy. These training pairs were obtained from five cycles through PROSSS (taking the baseline analysis and the five gradient analyses from each cycle) plus the initial baseline analysis data, and a final analysis execution with all design variables at the lower bounds. The lower bound execution was required because the neural network interpolates to obtain the output data.

The range of the data might also be an important factor because each value must be scaled to be between 0.1 and 0.9 for NETS. This is done by generating a file of training pairs for input to a scaling program provided in the delivery package to generate a file for input to NETS. The scaling equation is:

$$y = .1 + .8 * (x - xmin) / (xmax - xmin) \qquad (1)$$

where xmin and xmax are the minimum and maximum range values for the different input and output nodes.

4. MODIFYING THE SUBROUTINE TO CALL NETS

Section 2 mentions a routine for calling NETS from another program which is generated with the **"g"** option from the NETS menu. Name this routine ANALYNN. When called by NNOPT, the main program in NETS/PROSSS, ANALYNN will propagate input node data (the design variables) through the neural network and obtain output node data (constraints, behavior variables and an objective function) for input to the optimizer. However, several changes must be made to ANALYNN before it can be integrated into the NETS/PROSSS program. A sample of the ANALYNN routine is listed in Appendix A with comments to indicate the required changes.

Since NETS and its utilities are written in C, a C compiler is required. After making the modifications to the ANALYNN routine, save it on a file with a **".c"** suffix to distinguish it as C source code. Make sure the name of the routine is ANALYNN to match the call statement in the NNOPT main program. Replace the NETS routine **net.c** with **net-prosss.c** which is supplied in the delivery package. This routine changes the way NETS prints out information. It removes titles and places the information in a column format. This reduces the amount of I/O in NETS/PROSSS, hence speeding up the execution. The user must then link all FORTRAN and C routines together. The method for this depends on the computer being used. If the operating system is case sensitive, such as UNIX, then all file names should be lower case. Since most FORTRAN compilers are not case sensitive, the calls to routine ANALYNN and other external routines can remain upper case.

5. NETS/PROSSS

NETS/PROSSS is a variation of the original PROSSS described in references 1,2, and 3. The program requires several files for data storage and transfer.

(1) The input values for CONMIN (ref. 5) are read from unit 7. In addition, this file contains values for the move limits (BL and BU), the finite difference step size (XINC), the minimum and maximum values for each of the input and output nodes of the neural net (problem dependent), and the number of optimization cycles (MAXCYC) desired.

(2) CONMIN output is on unit 6.

(3) The weights (portable format) for the neural network are on file ANALYSIS.PWT.

(4) Unit 8 contains data output from NETS for input to CONMIN.

(5) Unit 9 contains data output from CONMIN for input to NETS.

The user is required to write two routines for NETS/PROSSS. The first routine is called OPT2ANL. An example of this problem dependent routine is in Appendix B. This routine scales (equation 1) the input node data to a form suitable for input to NETS (between 0.1 and 0.9). In addition, this routine reads the minimum and maximum values for the input and output nodes from unit 7 on the first cycle through the system. This requires that the cycle number be input to the routine. The minimum and maximum values are stored in a common block called RANGES. If the parameter list for the user-supplied OPT2ANL routine is different from that in the main program then the calling statement must be changed also.

The second routine is ANL2OPT. An example of this problem dependent routine is in Appendix C. This routine unscales the data output from the neural network for input into CONMIN. The unscaling equation is:

$$x = xmin + ((y - .1) * (xmax - xmin)) / .8 \qquad (2)$$

The routine requires an option parameter, IOPT, where IOPT=1 implies unscaling baseline data (initial design variables in a cycle) and IOPT=2 implies unscaling gradient data (perturbed design variables). Data are read from unit 8. Two additional initial reads and one additional final read are required because of the NETS output format. The minimum and maximum values, input through routine OPT2ANL, are passed to ANL2OPT through the RANGES common block. If the parameter list for ANL2OPT routine is changed then the calling statement in the main program must also be changed.

A flowchart for NETS/PROSSS is shown in figure 1. A listing of the problem independent code is Appendix D. The main program, NNOPT, calls the routines shown in the figure. CONMIN values are input in subroutine INCNMN. The design values are scaled for input to NETS by routine OPT2ANL. The design values are propagated through NETS by calling routine ANALYNN. The NETS output is unscaled for input to CONMIN by routine ANL2OPT. Gradients are computed in routine EVALSUB. The program then iterates through CONMIN and a linear extrapolation analysis routine called ANALY. If the objective function has converged to an optimum (the objective function has not changed by more than a certain tolerance for three iterations) the system stops, otherwise the design variables are saved and a new cycle begins.

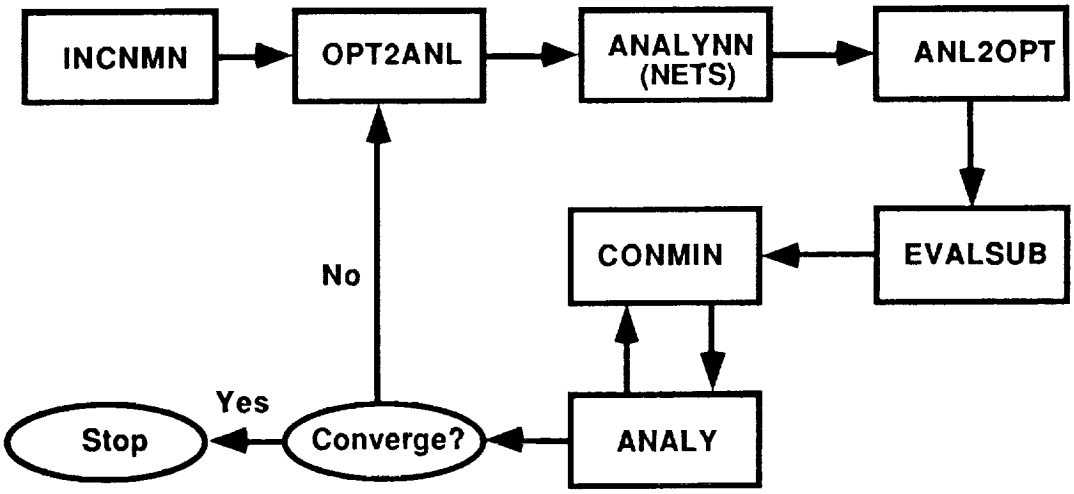


Figure 1 - NETS / PROSSS flowchart (Main program is NNOPT)

Figure 2 indicates how NETS/PROSSS can be integrated into an optimization process.

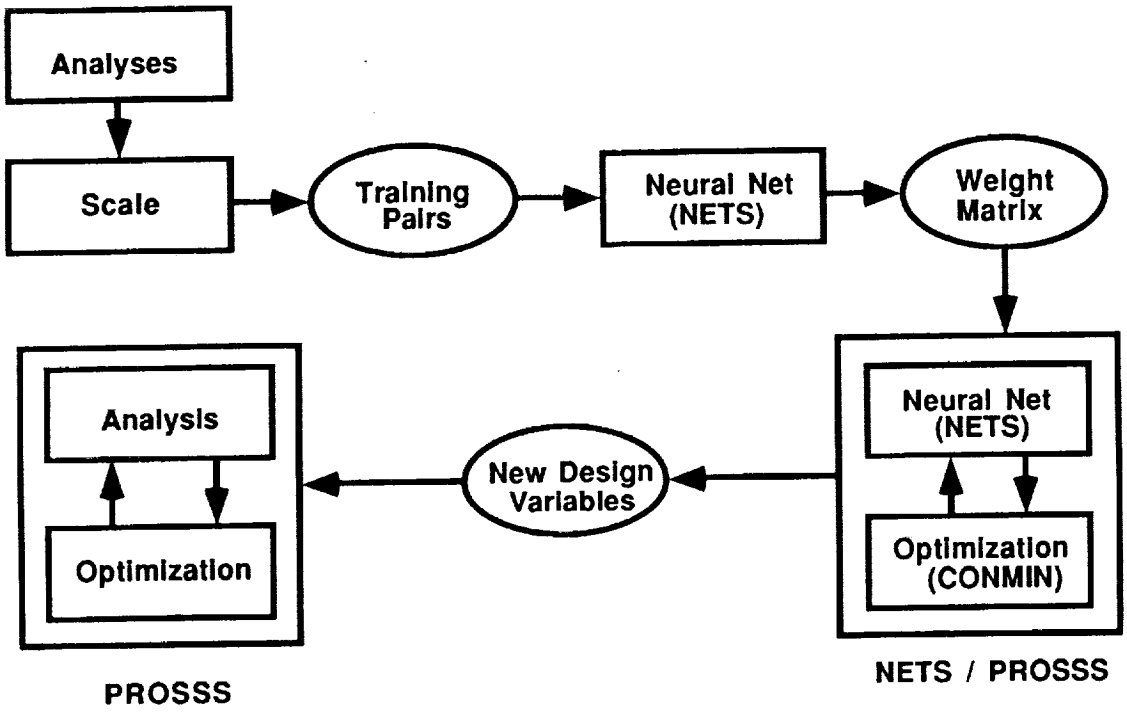
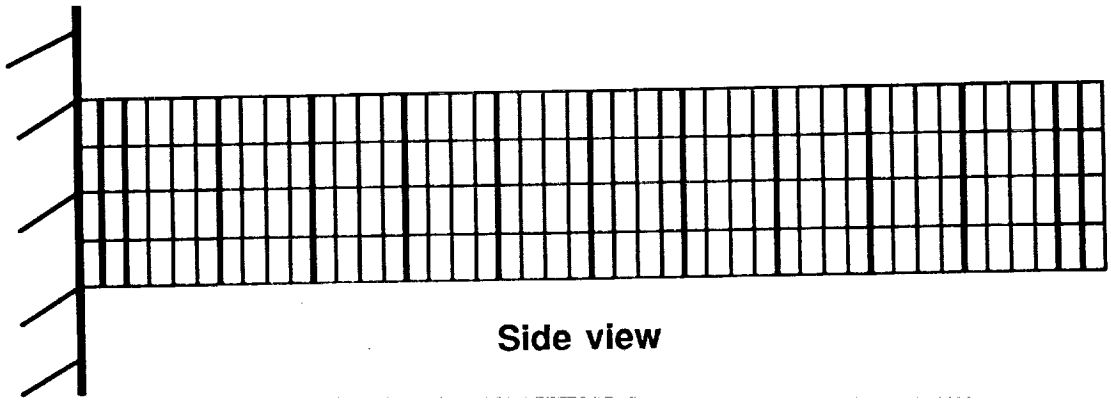


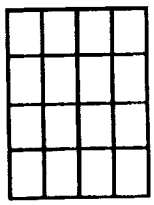
Figure 2 - Flowchart of optimization process with NETS/PROSSS

6. SAMPLE PROBLEM

A 3000 DOF cantilever beam (figure 3) serves as an example to demonstrate how to apply NETS/PROSSS. This finite element model for this beam has 1025 joints, 640 3D solid brick (1x1x2) elements (arranged 4x4x40). This problem has five design variables which determine the shape of the beam, forty cumulative stress constraints and one objective function (weight). The loads are applied in component 3 at the 25 joints on the outer edge of the beam as shown in figure 3.



Side view



End view

537	3418	4590	3418	537
1074	6836	9180	6836	1074
1074	6836	9180	6836	1074
1074	6836	9180	6836	1074
537	3418	4590	3418	537

Loads on joints at end view

Figure 3 - Test problem and model with loading conditions

The following steps indicate what is needed to execute NETS/PROSSS for this problem. Bold-face names indicate files which are available in the delivery package.

(1) The neural network has 5 input nodes, 41 output nodes, and 46 nodes on the hidden layer and is stored in file **BEAM.NET**.

(2) EAL/PROSSS (ref. 3) is executed to generate the training pairs. After creating an executable for program **SCALE.FOR**, this program is used to scale the data between 0.1 and 0.9, these pairs are stored in file **BEAM.IOP** for input into NETS to train the network and determine the weight matrix. (Note: The **SCALE.FOR** routine is problem dependent and needs to be modified to read the data in a format output from the analysis program. An example of this program for this particular sample problem appears in Appendix E.) The weight matrix is then stored in a portable format in file **ANALYSIS.PWT** using the **"s"** option in NETS.

(3) The **"g"** option in the NETS menu is used to create the routine ANALYNN. After making the changes discussed in Section 4, the routine is stored in file **BEAMNN.C** (Appendix A).

(4) A file, **BEAM.IN**, containing the input values for CONMIN, the move limits, finite different step size, the minimum and maximum values for each of the input and output nodes of the neural network, and the number of optimization iterations is created.

(5) Routines OPT2ANL and ANL2OPT are written and stored in files **OPT2ANL.FOR** and **ANL2OPT.FOR,** respectively.

(6) The problem dependent files: **BEAMNN.C** (Appendix A), **OPT2ANL.FOR** (Appendix B), and **ANL2OPT.FOR** (Appendix C); the optimizer, CONMIN; and the problem independent routines stored on file **NNPROSSS.FOR** (Appendix D) are compiled and linked to form the executable program.

(7) **BEAM.IN** is assigned to unit 7 and the program can be executed.

The complete optimization process with 5% move limits using PROSSS alone is shown in figure 4. The starting point has all design variables at 8. The final objective function is 219.61, with design variables of 6.68, 5.53, 4.53, 4.00, 4.00 (4.00 is the lower bounds for the optimizer).

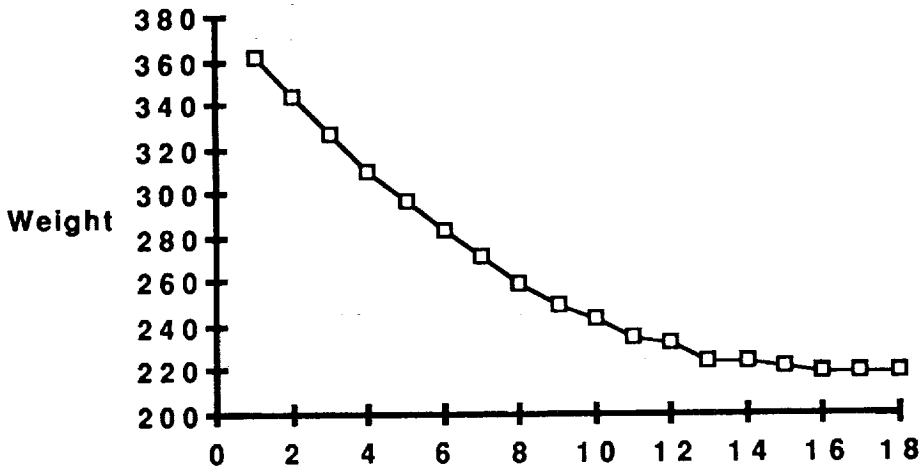


Figure 4 - Optimization process with PROSSS

7. REFERENCES

1. Sobieszczanski-Sobieski, J.; and Bhat, R. B.: "Adaptable Structural Synthesis Using Advanced Analysis and Optimization Coupled By a Computer Operating System." A Collection of Technical Papers on Structures - AIAA/ASME/ASCE/AHS 20th SDM Conference, April 1979, pp. 20-71, AIAA Paper No. 79-0723.

2. Rogers, J. L. Jr.; Sobieszczanski-Sobieski, J.; and Bhat, R. B.: "An Implementation of the Programming Structural Synthesis System (PROSSS)." NASA TM 83180, December 1981.

3. Rogers, J. L. Jr.: "Combining Analysis with Optimization at Langley Research Center - An Evolutionary Process." Proceedings of the Second International ASME Computers in Engineering Conference, Vol. 3, pp. 83-91, San Diego, CA, August 1982.

4. Baffes, P. T.: "NETS 2.0 User's Guide." LSC-23366, NASA Lyndon B. Johnson Space Center, September 1989.

5. Vanderplaats, G. N.: "CONMIN - A FORTRAN Program for Constrained Function Minimization User's Manual." NASA TM X-62282, August 1973.

Appendix A - Source listing for the ANALYNN routine for calling NETS

```
/*============================*/
/* NETS Network Delivery File  */
/*                            */
/* a product of the AI Section */
/* NASA, Johnson Space Center  */
/*                            */
/* principal author:          */
/*      Paul Baffes           */
/*                            */
/* contributing authors:      */
/*      Brian Dulock          */
/*      Chris Ortiz           */
/*============================*/
/* Modified for the NETS/PROSSS*/
/* application.  Developed at  */
/* NASA Langley Research Center*/
/*                            */
/* NETS/PROSSS developed by:   */
/*   Jim Rogers               */
/*                            */
/* Modified by:               */
/*   Bill LaMarsh II          */
/*============================*/

/*----------------------      NETS/PROSSS  mods  ------------------*/
/*  Make sure these ".h" files are in you current directory or  */
/*  modify these 5 lines to point to where they reside.        */
/*----------------------     end  NETS/PROSSS  mod  ----------------*/

#include "common.h"
#include "weights.h"
#include "layer.h"
#include "net.h"
#include "netio.h"
#define INPUT_SIZE  5
#define OUTPUT_SIZE 41

/*----------------------      NETS/PROSSS  mods  ------------------*/
/*  Include the external calls to:                          */
extern int     IO_get_default_int();
extern void    N_query_net();
/*----------------------     end  NETS/PROSSS  mod  ----------------*/


extern Net    *B_create_net();
extern Net    *B_free_net();
extern int     N_reset_wts();
extern void    P_prop_input();
```

```
extern void    PA_initialize();
extern void    D_initialize();
extern Sint    C_float_to_Sint();
extern float   C_Sint_to_float();
extern void    sys_init_rand();


/*------------------*/
/* Global Variables */
/*------------------*/
float  beam16_Inputs[INPUT_SIZE];
float  beam16_Outputs[OUTPUT_SIZE];
Net    *beam16_NetPtr;


/*-------------------------------------------------------------*/
/* Here is an example of a main routine. Note that the network */
/* is initialized once BEFORE before the propagate routine is  */
/* called and cleaned up once AFTER ALL calls to propagate are */
/* completed. That is, you only need to call initialize once   */
/* to build the network and once to throw it away. Note also   */
/* that the inputs and outputs are communicated via the two    */
/* global arrays defined at the top of the file.               */
/*                                                             */
/* This routine should be replaced with your own routines(s)   */
/* designed for your application.                              */
/*-------------------------------------------------------------*/

/*----------------------     NETS/PROSSS  mods  ----------------------*/
/* Modify subroutine name to analynn() from main() to analynn() */
/* Also, the user should be aware that the names of pointers    */
/* and subroutines will change.  In this example, all pointers  */
/* and subroutines will begin with "beam16" (example in User's  */
/* Guide).                                                      */
/*----------------------     end  NETS/PROSSS  mod  ----------------*/

analynn()
{
   int  i;
    void  beam16_initialize();      /* These names will change */
    void  beam16_propagate();          /* to whatever your net is */
   void  beam16_cleanup();         /* called.                 */

    beam16_initialize();

   for (i = 0; i < INPUT_SIZE; i++)
      beam16_Inputs[i] = .9;
    beam16_propagate();

/*----------------------     NETS/PROSSS  mods  ----------------------*/
/* These two lines are not necessary.  They only print out      */
/* the neural net give for the output.  This only clutters      */
/* the output file.  Leave them in for checkout (or curiousity) */
/* purposes.                                                    */
```

```
/*                                                    */
/*   for (i = 0; i < OUTPUT_SIZE; i++)                */
/*       printf("\n output %d = %10.6f", i, beam16_Outputs[i]);   */
/*----------------------     end  NETS/PROSSS  mod  ----------------*/

   beam16_cleanup();

} /* example main program */


/*-------------------------------------------------*/
/* call this routine once to setup the network */
/*-------------------------------------------------*/
void   beam16_initialize()
{
/*----------------------     NETS/PROSSS  mods  -------------------*/
/* Modification to the "_initialize" routine was necessary to    */
/* include the weights (.pwt), input (fort.8) and                */
/* output (fort.9) all using files as opposed to interactive.    */
/*                                                    */
/* Another int was declared (t1) as well as a FILE pointer       */
/* (fp).                                               */
/*                                                    */
   int t1;
   FILE *fp;
/*                                                    */
/*----------------------     end  NETS/PROSSS  mod  ----------------*/
   int  i;

        /*-------------------------*/
    /* call initialization code */
        /*-------------------------*/
   beam16_NetPtr = NULL;
    sys_init_rand();
    PA_initialize();
    D_initialize();

       /*---------------*/
     /* create network.*/
    /* uses network    */
     /* configuration   */
    /* file (.net)     */
    /* that was used   */
     /* for training.   */
       /*---------------*/
    beam16_NetPtr = B_create_net(1, "beam16.net");
   beam16_NetPtr->use_biases = TRUE;
   beam16_NetPtr->num_inputs  = INPUT_SIZE;
   beam16_NetPtr->num_outputs = OUTPUT_SIZE;

        /*-----------------------------------------*/
    /* reset weights and the input, output arrays */
        /*-----------------------------------------*/
/*----------------------     NETS/PROSSS  mods  -------------------*/
```

```
/* Here is where the files are being used.  Comment or delete   */
/* these lines:                                                 */
/*                                                              */
/*      N_reset_wts(beam16_NetPtr, "temp.pwt", PORTABLE_FORMAT);  */
/*  for (i = 0; i < INPUT_SIZE; i++)                            */
/*      beam16_Inputs[i] = 0.0;                                 */
/*   for (i = 0; i < OUTPUT_SIZE; i++)                          */
/*      beam16_Outputs[i] = 0.0;                                */
/*                                                              */
/* and replace with:                                            */
/*                                                              */
/*----------------------     end  NETS/PROSSS  mod  ----------------*/

    t1 = (N_reset_wts(beam16_NetPtr, "analysis.pwt", PORTABLE_FORMAT));
   if (t1 == ERROR) printf ("ERROR in resetting weights ");
    fp = fopen("fort.8", "wt");
    N_query_net(beam16_NetPtr, "fort.9", fp, -1);
   fclose(fp);
   for (i = 0; i < INPUT_SIZE; i++)
      beam16_Inputs[i] = 0.0;
   for (i = 0; i < OUTPUT_SIZE; i++)
      beam16_Outputs[i] = 0.0;


} /* beam16_initialize */


/*-------------------------------------------------*/
/* call this routine every time you want to query */
/* the network. Note that it assumes the "Input"  */
/* array is already loaded with input values      */
/*-------------------------------------------------*/
void  beam16_propagate()
{
   int  i;
    Layer  *input, *output;

          /*---------------------------------------*/
    /* get pointers to network input and output */
          /*---------------------------------------*/
    input  = beam16_NetPtr->input_layer;
    output = beam16_NetPtr->output_layer;

          /*---------------------------------------*/
    /* load input values; propagate network */
          /*---------------------------------------*/
   for (i = 0; i < INPUT_SIZE; i++)
       input->node_outputs[i] = C_float_to_Sint(beam16_Inputs[i]);
    P_prop_input(beam16_NetPtr);

         /*--------------------*/
    /* setup output values */
         /*--------------------*/
   for (i = 0; i < OUTPUT_SIZE; i++)
```

```
        beam16_Outputs[i] = C_Sint_to_float(output->node_outputs[i]);

} /* beam16_propagate */


/*-----------------------------------------*/
/* call this routine once to free the space */
/* used by the network.                     */
/*-----------------------------------------*/
void  beam16_cleanup()
{
    B_free_net(beam16_NetPtr);

} /* beam16_cleanup */
```

Appendix B - Source listing for problem dependent subroutine OPT2ANL

```
      SUBROUTINE OPT2ANL(ICYCLE,NDV,N1,XINC,X)
C*************************************************************
C
C  THIS SUBROUTINE CONVERTS DESIGN VARIABLES OUTPUT FROM CONMIN
C  TO INPUT FOR THE NEURAL NET BY SCALING THE DESIGN
C  VARIABLES AND THEIR PERTURBATIONS TO BETWEEN .1 AND .9
C
C*************************************************************
      COMMON/RANGES/DVMAX,DVMIN,OBJMAX,OBJMIN,CONMAX,CONMNN
      DIMENSION X(N1)
C
C INPUT RANGES ON FIRST PASS
C
      IF(ICYCLE.EQ.1) THEN
        READ(7,40) DVMAX
        READ(7,40) DVMIN
        READ(7,40) OBJMAX
        READ(7,40) OBJMIN
        READ(7,40) CONMAX
        READ(7,40) CONMNN
      ENDIF
      REWIND 9
C
C  WRITE SCALED DESIGN VARIABLES
C
       WRITE(9,10)
      DO I = 1,NDV
          DVSCAL = .1+.8*((X(I)-DVMIN)/(DVMAX-DVMIN))
        IF(DVSCAL.LT..1) DVSCAL = .1
        IF(DVSCAL.GT..9) DVSCAL = .9
        WRITE(9,20) DVSCAL
      ENDDO
       WRITE(9,30)
C
C  LOOP THROUGH DESIGN VARIABLES
C  MAKE PERTURBATIONS TO COMPUTE FINITE DIFFERENCES
C
      DO J = 1,NDV
         X(J) = X(J)*(1.-XINC)
        WRITE(9,10)
        DO I = 1,NDV
            DVSCAL = .1+.8*((X(I)-DVMIN)/(DVMAX-DVMIN))
          IF(DVSCAL.LT..1) DVSCAL = .1
          IF(DVSCAL.GT..9) DVSCAL = .9
          WRITE(9,20) DVSCAL
       ENDDO
        WRITE(9,30)
          X(J) = X(J)/(1.-XINC)
      ENDDO
```

```
      REWIND 9
C
   10  FORMAT(1H()
   20  FORMAT(F10.5)
   30  FORMAT(1H))
   40  FORMAT(10X,F10.5)
      RETURN
      END
```

Appendix C - Source listing for problem dependent subroutine ANL2OPT

```
      SUBROUTINE ANL2OPT(IOPT,NCON,N2,G,GI,OBJ,OBJI)
C*****************************************************************
C
C THIS SUBROUTINE CONVERTS DATA OUTPUT FROM THE NEURAL NET
C TO INPUT FOR CONMIN BY UNSCALING FROM .1 TO .9
C
C*****************************************************************
      COMMON/RANGES/DVMAX,DVMIN,OBJMAX,OBJMIN,CONMAX,CONMNN
      DIMENSION G(N2),GI(N2),UNSCLC(3)
      CHARACTER*80 TEMP
C
C  IOPT = 1 => BASE DATA
C  IOPT = 2 => PERTURBED DATA
C
C  UNSCALE INPUT VALUES.  Two initial read are required since
C  the output from nets has a blank line then an 'Output' statement.
C
      READ(8,30) TEMP
      READ(8,30) TEMP
C
C UNSCALE CONSTRAINTS
C
      READ(8,40) UNSCLC(1),UNSCLC(2),UNSCLC(3),UNSCLO
      DO II = 1,NCON
        IF(IOPT.EQ.1)
     1    GI(II) = CONMNN + (((UNSCLC(II)-.1)*(CONMAX-CONMNN))/.8)
        IF(IOPT.EQ.2)
     1    G(II) = CONMNN + (((UNSCLC(II)-.1)*(CONMAX-CONMNN))/.8)
      ENDDO
C
C UNSCALE OBJECTIVE FUNCTION
C
      IF(IOPT.EQ.1)
     1  OBJI = OBJMIN + (((UNSCLO-.1)*(OBJMAX-OBJMIN))/.8)
      IF(IOPT.EQ.2)
     1  OBJ = OBJMIN + (((UNSCLO-.1)*(OBJMAX-OBJMIN))/.8)
C
C  One more read is required to read the last ')'.
C
      READ(8,30) TEMP
C
   30  FORMAT(80A)
   40  FORMAT(3X,F8.6,3X,F8.6,3X,F8.6,3X,F8.6)
      RETURN
      END
```

Appendix D - Source listing for problem independent routines of NETS/PROSSS

```
      PROGRAM NNOPT
C
C PROGRAM TO INTERFACE OPTIMIZER WITH NEURAL NET TO
C PERFORM CONSTRAINED MINIMIZATION OF A FUNCTION FOR FINITE DIFFERENCE
C INPUTS : VECTORS OF DESIGN VARIABLES AND BOUNDS
C        VECTORS OF OBJECTIVE FUNCTION AND GRADIENTS
C        VECTORS OF CONSTRAINTS AND GRADIENTS
C
C A TAYLOR SERIES EXPANSION IS USED FOR APPROXIMATE ANALYSIS
C
C
C FILES USED BY THIS PROGRAM
C   FILE 6  CONTAINS OUTPUT FROM CONMIN
C   FILE 7  CONTAINS INPUT DATA FOR CONMIN
C   FILE 8  CONTAINS DATA OUTPUT FROM THE NEURAL NET
C   FILE 9  CONTAINS DATA INPUT FOR THE NEURAL NET
C
      REAL*4 A(20000)
     COMMON/OPTREAL/BL,BU,XINC
     COMMON/CNMN1/ DELFUN,DABFUN,FDCH,FDCHM,CT,CTMIN,CTL,CTLMIN,ALPHAX,
    1            ABOBJ1,THETA,OBJ,NDV,NCON,NSIDE,IPRINT,NFDG,NSCAL,
    2            LINOBJ,ITMAX,ITRM,ICNDIR,IGOTO,NAC,INFO,INFOG,ITER
      COMMON/CONSAV/RSAV(50),ISAV(25)
     DIMENSION KA(1)
     EQUIVALENCE (A(1), KA(1))
     EXTERNAL ANALYNN    !$PRAGMA C(ANALYNN)
C
C  Initialize convergence data.
C
      OBJ1 = 10.
      OBJ2 = 50.
      OBJ3 = 10.
      TOL = .001
C
C INPUT PARAMETERS FOR CONMIN
C
     CALL INCNMN
C
C SET UP BLANK COMMON
C
      N1 = NDV + 2
      N2 = NCON + 2*NDV
      N3 = N2
      N4 = N2
       N5 = 2*N2
      NDVP1 = NDV + 1
C
      IADX   = 1
      IADVLB = IADX + N1
```

```
      IADVUB = IADVLB + N1
      IADG   = IADVUB + N1
      IADSCA = IADG + N2
      IADDF = IADSCA + N1
      IADA = IADDF + N1
       IADS = IADA + N1*N3
      IADG1 = IADS + N1
      IADG2 = IADG1 + N2
      IADB = IADG2 + N2
       IADC = IADB + N3*N3
      IADISC = IADC + N4
      IADIC = IADISC + N2
      IADMS1 = IADIC + N3
      IADXI = IADMS1 + N5
      IAVLBI = IADXI + N1
      IAVUBI = IAVLBI + N1
      IADGI = IAVUBI + N1
     IAGRDO = IADGI + N2
    IAGRDG = IAGRDO + N1
     KREQ = IAGRDG + (N1*N2)
C
C ZERO OUT BLANK COMMON
C
     DO II = 1,KREQ
        A(II) = 0.
        KA(II) = 0
     ENDDO
C
C INPUT DATA FOR OPTIMIZATION
C
      DO II = 1,NDV
         READ(7,10) A(IAVLBI+II-1)
     ENDDO
      DO II = 1,NDV
         READ(7,10) A(IAVUBI+II-1)
     ENDDO
      DO II = 1,NCON
         READ(7,20) KA(IADISC+II-1)
     ENDDO
C
C READ IN INITIAL DESIGN VARIABLES
C
       DO II = 1,NDV
          READ(7,10) A(IADXI+II-1)
      ENDDO
C
C READ IN MAXIMUM OPTIMIZATION ITERATIONS
C
      READ(7,20) MAXCYC
C
C LOOP THROUGH OPTIMIZTION CYCLE
C
     DO ICYC = 1,MAXCYC
C
```

```
C SCALE DATA FOR INPUT TO NEURAL NET
C
     CALL OPT2ANL(ICYC,NDV,N1,XINC,A(IADXI))
C
C SIMULATE ANALYSIS BY PROPAGATING
C DATA THROUGH NEURAL NET
C
     CALL ANALYNN()
C
C INPUT BASE DATA COMPUTED BY NEURAL NET
C
    REWIND 8
      CALL ANL2OPT(1,NCON,N2,A(IADG),A(IADGI),OBJ,OBJI)
C
C CALL SUBROUTINE TO COMPUTE GRADIENTS
C
    DO ICOUNT = 1,NDV
      CALL ANL2OPT(2,NCON,N2,A(IADG),A(IADGI),OBJ,OBJI)
      CALL EVALSUB(N1,N2,NDV,NCON,A(IADX),A(IADXI),A(IADG),A(IADGI),
    1          A(IAGRDO),A(IAGRDG),OBJ,OBJI,XINC,ICOUNT)
    ENDDO
    REWIND 8
C
C STORE INITIAL VALUES FOR OPTIMIZATION
C
     DO II = 1,NDV
        A(IADX+II-1)   = A(IADXI+II-1)
        A(IADVLB+II-1) = A(IAVLBI+II-1)
        A(IADVUB+II-1) = A(IAVUBI+II-1)
    ENDDO
     DO II = 1,NCON
        A(IADG+II-1) = A(IADGI+II-1)
    ENDDO
     OBJ = OBJI
C
C COMPARE PHYSICAL BOUNDARIES WITH MOVE LIMITS
C CHANGE PHYSICAL BOUNDARIES IF NECESSARY
C
     DO II = 1,NDV
        TVLB = BL *  A(IADXI+II-1)
        IF(TVLB.GT.A(IADVLB+II-1)) A(IADVLB+II-1) = TVLB
        TVUB = BU * A(IADXI+II-1)
        IF(TVUB.LT.A(IADVUB+II-1)) A(IADVUB+II-1) = TVUB
    ENDDO
C
C LOOP THROUGH OPTIMIZER AND LINEAR ANALYSIS
C
     DO LOOP=1,50
C
C CALL TO OPTIMIZER
C
       CALL CONMIN(A(IADX),A(IADVLB),A(IADVUB),A(IADG),A(IADSCA),
      1            A(IADDF),A(IADA),A(IADS),A(IADG1),A(IADG2),A(IADB),
      2            A(IADC),KA(IADISC),KA(IADIC),KA(IADMS1),N1,N2,N3,N4,N5)
```

```
C
C SKIP OUT OF LOOP IF OPTIMIZATION IS COMPLETE
C
    IF(IGOTO.EQ.0) GO TO 60
C
C LINEAR EXTRAPOLATION ANALYSIS
C
      CALL  ANALY(OBJI,A(IADGI),A(IADG),A(IADXI),A(IADX),
     1  A(IAGRDO),A(IAGRDG),A(IADDF),KA(IADIC),A(IADA),N1,N2,N3)
C
   ENDDO
C
C RESET DESIGN VARIABLE DATA FOR USE BY NEURAL NET
C
 60   DO II = 1,NDV
        A(IADXI+II-1) = A(IADX+II-1)
   ENDDO
C
C SAVE DATA FOR PLOTTING
C
     WRITE(12,*) OBJ
C
C CHECK FOR TERMINATION BECAUSE NO CHANGE IN
C OBJECTIVE FUNCTION IN THREE PASSES
C
     OBJ3=OBJ2
     OBJ2=OBJ1
     OBJ1=OBJI
C
      DA=ABS((OBJ3-OBJ2)/OBJ2)
      DB=ABS((OBJ2-OBJ1)/OBJ2)
    IF ((DA.LE.TOL).AND.(DB.LE.TOL)) GO TO 40
   ENDDO
C
  10   FORMAT(10X,F10.5)
  20   FORMAT(10X,I5)
  30   FORMAT(4F10.5)
C
  40  STOP
    END
    SUBROUTINE INCNMN
C*********************************************************************
C
C THIS SUBROUTINE READS IN THE CONMIN PARAMETERS
C
C*********************************************************************
     COMMON/OPTREAL/BL,BU,XINC
     COMMON/CNMN1/ DELFUN,DABFUN,FDCH,FDCHM,CT,CTMIN,CTL,CTLMIN,ALPHAX,
    1            ABOBJ1,THETA,OBJ,NDV,NCON,NSIDE,IPRINT,NFDG,NSCAL,
    2            LINOBJ,ITMAX,ITRM,ICNDIR,IGOTO,NAC,INFO,INFOG,ITER
    REWIND 7
C
C INTEGER PARAMETERS
C
```

```
      READ(7,10) NDV
      READ(7,10) NCON
      READ(7,10) NSIDE
      READ(7,10) IPRINT
      READ(7,10) NFDG
      READ(7,10) NSCAL
      READ(7,10) LINOBJ
      READ(7,10) ITMAX
      READ(7,10) ITRM
      READ(7,10) ICNDIR
      READ(7,10) IGOTO
      READ(7,10) NAC
      READ(7,10) INFO
      READ(7,10) INFOG
      READ(7,10) ITER
C
C REAL PARAMETERS
C
      READ(7,20) DELFUN
      READ(7,21) DABFUN
      READ(7,20) FDCH
      READ(7,20) FDCHM
      READ(7,20) CT
      READ(7,20) CTMIN
      READ(7,20) CTL
      READ(7,20) CTLMIN
      READ(7,20) ALPHAX
      READ(7,20) ABOBJ1
      READ(7,20) THETA
      READ(7,20) OBJ
C
      READ(7,20) BL
      READ(7,20) BU
      READ(7,20) XINC
C
   10  FORMAT(10X,I5)
   20  FORMAT(10X,F10.5)
   21  FORMAT(10X,E15.10)
      RETURN
      END
```

```
      SUBROUTINE ANALY(OBJI,GI,G,XI,X,GRDOBJ,GRDG,
     1 DF,IC,A,N1,N2,N3)
C*************************************************************************
C
C  SUBROUTINE FOR LINEAR EXTRAPOLATION ANALYSIS
C
C*************************************************************************
      COMMON/CNMN1/ DELFUN,DABFUN,FDCH,FDCHM,CT,CTMIN,CTL,CTLMIN,ALPHAX,
     1          ABOBJ1,THETA,OBJ,NDV,NCON,NSIDE,IPRINT,NFDG,NSCAL,
     2          LINOBJ,ITMAX,ITRM,ICNDIR,IGOTO,NAC,INFO,INFOG,ITER
      DIMENSION GI(N2),G(N2),XI(N1),X(N1),GRDOBJ(N1),GRDG(N1,N2),
     1 DF(N3),A(N1,N2),IC(N3)
      IF(INFO.EQ.2) GO TO 24
C
C  RESTORE OBJECTIVE FUNCTION AND CONSTRAINTS TO INITIAL VALUES
C
      OBJ=OBJI
      DO 5 IJ=1,NCON
    5 G(IJ)=GI(IJ)
C
C  RECOMPUTE OBJECTIVE FUNCTION AND COONSTRAINTS BASED ON DIFFERENCE
C  IN PERMUTED DESIGN VARIABLE
C
      DO 20 I=1,NDV
      DXI = X(I)-XI(I)
      GINC = GRDOBJ(I)*DXI
      IF(GRDOBJ(I).LT.0.0 .AND. LINOBJ.EQ.0)
     1  GINC = GINC/(1.+DXI/X(I))
      OBJ = OBJ+GINC
      DO 10 J=1,NCON
      GINC = GRDG(I,J)*DXI
      IF(GRDG(I,J).LT.0.0) GINC = GINC/(1.+(DXI/X(I)))
      G(J) = G(J)+GINC
   10 CONTINUE
   20 CONTINUE
      GO TO 70
C
C  STORE GRADIENT OF OBJECTIVE FUNCTION FOR EACH DESIGN VARIABLE
C
   24 DO 25 IDF = 1,NDV
      DF(IDF) = GRDOBJ(IDF)
   25 CONTINUE
C
C  DETERMINE ACTIVE AND VIOLATED CONSTRAINTS
C
      NAC=0
      DO 30 J=1,NCON
      IF(G(J).LT.CTL)GO TO 30
      NAC=NAC+1
      IC(NAC)=J
   30 CONTINUE
C
C  STORE CONSTRAINT GRADIENTS
C
```

```
      DO 60 II=1,NDV
      DO 50 JJ=1,NAC
       J1=IC(JJ)
       A(II,JJ)=GRDG(II,J1)
 50  CONTINUE
 60  CONTINUE
 70  RETURN
     END
     SUBROUTINE EVALSUB(N1,N2,NDV,NCON,X,XI,G,GI
     1 ,GRDOBJ,GRDG,OBJ,OBJI,XINC,ICOUNT)
C*****************************************************************
C
C  SUBROUTINE TO COMPUTE GRADIENTS
C
C*****************************************************************
      DIMENSION X(N1),XI(N1),G(N2),GI(N2),GRDOBJ(N1),GRDG(N1,N2)
C
C  FIND CHANGE IN DESIGN VARIABLE
C
      X(ICOUNT) = XI(ICOUNT)*(1.-XINC)
     DELTX=X(ICOUNT)-XI(ICOUNT)
     X(ICOUNT) = XI(ICOUNT)
C
C  CONSTRAINT GRADIENTS
C
     DO L = 1,NCON
       GRDG(ICOUNT,L)=(G(L)-GI(L))/DELTX
     ENDDO
C
C  OBJECTIVE FUNCTION GRADIENTS
C
      GRDOBJ(ICOUNT)=(OBJ-OBJI)/DELTX
C
     RETURN
     END
```

Appendix E - Source listing for the problem dependent program for scaling data for input to the neural net

```
      PROGRAM SCALE
C
C  THIS PROGRAM SCALES THE DESIGN VARIABLES, CONSTRAINTS,
C  AND OBJECTIVE FUNCTION FOR INPUT TO NETS (.1 < X < .9)
C
       READ(5,10) NDV
       READ(5,10) NCON
       READ(5,10) NPAIRS
       READ(5,20) DVMAX
       READ(5,20) DVMIN
       READ(5,20) OBJMAX
       READ(5,20) OBJMIN
       READ(5,20) CONMAX
       READ(5,20) CONMIN
C
C  LOOP ON NUMBER OF TRAINING PAIRS REQUIRED
C
       DO I = 1,NPAIRS
         WRITE(8,5)
    5    FORMAT(1H()
C
C  DESIGN VARIABLES
C
         DO II = 1,NDV
           READ(7,30) DV
             DVNN = .1 + (.8*((DV-DVMIN)/(DVMAX-DVMIN)))
           IF(DVNN.LT..1) DVNN = .1
           IF(DVNN.GT..9) DVNN = .9
           WRITE(8,30) DVNN
         ENDDO
C
C  CONSTRAINTS
C
         DO II = 1,NCON
           READ(7,30) CON
            CONNN = .1 + (.8*((CON-CONMIN)/(CONMAX-CONMIN)))
           IF(CONNN.LT..1) CONNN = .1
           IF(CONNN.GT..9) CONNN = .9
           WRITE(8,30) CONNN
         ENDDO
C
C  OBJECTIVE FUNCTION
C
         READ(7,30) OBJ
           OBJNN = .1 + (.8*((OBJ-OBJMIN)/(OBJMAX-OBJMIN)))
         IF(OBJNN.LT..1) OBJNN = .1
         IF(OBJNN.GT..9) OBJNN = .9
         WRITE(8,40) OBJNN
```

```
      ENDDO
C
C FORMATS
C
  10  FORMAT(10X,I5)
  20  FORMAT(10X,F10.5)
  30  FORMAT(F10.5)
  40  FORMAT(F10.5,1H))
C
      STOP
      END
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 1991	Technical Memorandum

4. TITLE AND SUBTITLE
User's Guide for NETS/PROSSS

5. FUNDING NUMBERS
505-63-50-06

6. AUTHOR(S)
James L. Rogers and William J. LaMarsh II

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, VA 23665-5225

8. PERFORMING ORGANIZATION REPORT NUMBER

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-104166

11. SUPPLEMENTARY NOTES
James L. Rogers: Langley Research Center, Hampton, Virginia.
William J. LaMarsh II: Unisys Corporation, Hampton, Virginia.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited

Subject Category 61

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Expensive analysis programs are often combined with optimization procedures to solve engineering problems. To obtain an optimal solution requires numerous iterations between the analysis program and the optimizer. This often becomes prohibitive due to the cost and amount of computer time needed to converge to an optimal solution. Therefore, any new software package that could significantly reduce the amount of computer time required to reach an optimal solution would be beneficial.

NETS/PROSSS was developed to help meet this need. The purpose of this paper is to serve as a user's guide for NETS/PROSSS. The key features include the neural network, determining the training pairs for the neural network, and the approximated analysis/optimization process. A small problem is given to serve as an example of how to apply the system.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Neural Networks; Optimization, Analysis; Structured Synthesis	30
	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	Unclassified	