Paper 242-2010

Nonlinear Optimization in SAS/OR® Software: Migrating from PROC NLP to PROC OPTMODEL

Tao Huang and Ed Hughes, SAS Institute Inc., Cary, NC

ABSTRACT

PROC OPTMODEL, the flagship optimization procedure in SAS/OR software, is also intended to supersede PROC NLP in the long run for nonlinear optimization. The rich and flexible PROC OPTMODEL syntax enables natural and compact algebraic formulations of nonlinear models. In addition to supporting the major legacy algorithms present in PROC NLP, PROC OPTMODEL has access to powerful new algorithms that can solve large-scale problems and handle nonlinearity more robustly. These new algorithms exploit sparse structures and implement the state-of-the-art techniques in numerical optimization.

In addition to benefiting from ease of modeling and access to improved algorithms, PROC OPTMODEL users can use its programming capabilities to develop customized solution methods. PROC OPTMODEL has many features that make it an excellent replacement for PROC NLP not only for operations research practitioners but also for statisticians, econometricians, and data miners. This paper uses several examples to illustrate how to migrate from PROC NLP to PROC OPTMODEL and highlight the benefits of doing so.

INTRODUCTION

An optimization problem involves minimizing (or maximizing) an objective function subject to a set of constraints. The objective and constraint functions are any linear or nonlinear functions of some decision variables. In its simplest form, a constraint specifies the range of values that a decision variable can take. Mathematically a general optimization problem is described as

$$\begin{array}{ll} \text{minimize} \mid \text{maximize} & f(x) \\ \text{subject to} & c^l \le c(x) \le c^u \\ & x^l \le x \le x^u \end{array}$$

where x is a vector of decision variables, $f(x)$ is the objective function, $c^l \le c(x) \le c^u$ are the general constraints, and $x^l \le x \le x^u$ are the bound constraints. In the preceding formulation, the problem is a *nonlinear optimization problem* (NLP) if at least one of the functions in $f(x)$ or $c(x)$ is nonlinear. If you classify optimization problems by the types of constraints they have, you have the following definitions:

- unconstrained optimization problem—if neither the general constraints nor the bound constraints are present
- bound-constrained optimization problem—if the general constraints are absent but the bound constraints are present with at least one finite component in x^l or x^u
- linearly constrained optimization problem—if the functions in $c(x)$ are all linear
- nonlinearly constrained optimization problem—if at least one of the functions in $c(x)$ is nonlinear

In many statistical and econometric applications, the decision variables usually represent some parameters that you want to estimate.

PROC NLP came into existence as a general-purpose nonlinear optimization procedure with provision for algorithms for all four classes of optimization problems in addition to its own modeling statements. PROC NLP also provides algorithms that specialize in least squares problems. PROC NLP has an array of robust and efficient algorithms for the classes of nonlinear optimization problems for which it was designed, and the procedure has served the optimization needs of numerous SAS users well for nearly two decades.

Since the advent of PROC NLP, the nonlinear optimization community has seen remarkable progress in the areas of modeling languages and algorithmic theory and practice. A modern algebraic modeling language offers many enhanced capabilities and flexibility in formulating a nonlinear optimization problems. Recent algorithmic developments have made it possible to solve more difficult nonlinear optimization problems, such as highly nonlinear problems and large-scale problems. For the former, high nonlinearity refers both to the number of nonlinear constraints involved and also to the characteristics of the objective functions and constraints. PROC OPTMODEL was designed to enable all SAS users to benefit from the aforementioned progress and developments in nonlinear optimization. More specifically, PROC OPTMODEL has a modern algebraic modeling language whose rich and flexible syntax enables natural and compact algebraic formulation of nonlinear optimization problems. In addition to supporting the major legacy algorithms present in

PROC NLP, PROC OPTMODEL has access to powerful new algorithms that can solve large-scale problems and handle nonlinearity more robustly.

The goal of this paper is to encourage current PROC NLP users to migrate to PROC OPTMODEL. The following sections use examples to illustrate how to make such a migration and highlight the benefits of doing so. The three sections from "HOW TO FORMULATE AN NLP IN PROC NLP" to "ADDITIONAL MODELING ADVANTAGES OF PROC OPTMODEL" use several examples to draw a mapping from PROC NLP to PROC OPTMODEL in the aspects of formulating nonlinear optimization problems and describes the modeling advantages that PROC OPTMODEL offers. The next three sections from "NONLINEAR OPTIMIZATION ALGORITHMS IN PROC NLP" to "PERFORMANCE GAINS FROM USING ALGORITHMS IN PROC OPTMODEL" outline the algorithms available in PROC NLP versus the algorithms available in PROC OPTMODEL and discuss the performance gains from using the nonlinear optimization algorithms in PROC OPTMODEL. The sections "PROGRAMMING CAPABILITIES OF THE PROC OPTMODEL MODELING LANGUAGE" and "BUILDING CUSTOMIZED SOLUTIONS WITH PROC OPTMODEL" demonstrate additional benefits of using PROC OPTMODEL and show how to build customized solutions in PROC OPTMODEL.

ELEMENTS OF AN NLP

To solve a general nonlinear optimization problem as described in the section "INTRODUCTION," you first need to specify the following elements of an NLP:

- decision variables x
- objective function $f(x)$
- linear or nonlinear constraints $c^l \le c(x) \le c^u$
- bounds on the decision variables $x^l \le x \le x^u$

In addition it is often desirable for you to specify the initial values of x from which an algorithm starts.

Although there are different ways to specify these elements of a nonlinear optimization problem, you would certainly expect the modeling language or statements of a nonlinear optimization procedure to enable you to input a nonlinear optimization problem with ease and flexibility. The following two sections describe the PROC NLP statements for input of a nonlinear optimization problem, discuss their limitations in terms of modeling capabilities, map the PROC NLP statements to the equivalent PROC OPTMODEL statements, and describe the PROC OPTMODEL modeling capabilities that overcome the limitations of PROC NLP.

HOW TO FORMULATE AN NLP IN PROC NLP

Table 1 describes the modeling statements in PROC NLP that you would usually use to formulate a nonlinear optimization problem.

Table 1 Modeling Statements in PROC NLP

Statement	Description
DECVAR	Declares the decision variables or specifies their initial values
BOUNDS	Specifies bounds on the decision variables
MIN | MAX | LSQ	Specifies the objective function for a minimization, maximization, or least squares problem
LINCON	Specifies linear constraints
NLINCON	Specifies nonlinear constraints
ARRAY	Associates an array with a list of variables and constants

An alias for the keyword DECVAR is PARMS, which stands for "parameters" from the parameter estimation standpoint in statistical applications. Note that the LSQ statement in Table 1 is an alternative to the MIN statement for least squares problems. In addition to these modeling statements, the INEST=, DATA=, and INQUAD= options in the PROC NLP statement allow input from data sets. Table 2 summarizes some of the features of these options.

Table 2 Options for Input of Data in PROC NLP

Option	Description	Allow Multiple Data Sets?	Format	
			Dense	Sparse
DATA=	Inputs data to construct nonlinear objective function or constraints	No	X	
INEST=	Specifies initial values or bounds of decision variables, or linear constraints	No	X	
INQUAD=	Specifies a quadratic programming problem	No	X	X

Example A.1 through Example A.5 demonstrate the usage of these statements and options based on the common scenarios in which they are usually used by PROC NLP users. Any limitations these statements and options might have are pointed out. Since the INQUAD= option is used for input of a quadratic programming problem, only the usage of the INEST= and DATA= options are illustrated.

Example A.1. Consider the following constrained nonlinear least squares (CNLSQ) problem:

$$
\begin{array}{ll}
\text{minimize} & (x_1^2 - x_1x_2 + e^{3x_1})^2 + (2x_1x_2 + x_2^2 + e^{2x_2})^2 + (-3x_1x_2 + e^{x_1-2x_2})^2 \\
\text{subject to} & 2x_1 - x_2 \leq 1 \\
& x_1 + 4x_2 \geq 5 \\
& 2x_1^2 - 3x_1x_2 + 4x_2 \leq 8 \\
& 2x_1 - x_2 - e^{x_1^2} \geq -1.5 \\
& -5 \leq x_1 \leq 100 \\
& x_2 \geq 0
\end{array}
$$

You can formulate the CNLSQ problem with PROC NLP as shown in the following SAS program:

```
proc nlp;
   array x[2] x1 x2;
   decvar x1 x2 = 0.5;
   bounds -5 <= x1 <= 100,
               x2 >= 0;
   min f;
   lincon 2*x1 - x2 <= 1,
          x1 + 4*x2 >= 5;
   nlincon nc1 <= 8,
           nc2 >= -1.5;
   f = (x[1]**2 - x[1]*x[2] + exp(3*x[1]))**2
       + (2*x[1]*x[2] + x[2]**2 + exp(2*x[2]))**2
       + (-3*x[1]*x[2] + exp(x[1] - 2*x[2]))**2;
   nc1 = 2*x[1]**2 - 3*x[1]*x[2] + 4*x[2]**2;
   nc2 = 2*x[1] - x[2] - exp(x[1]**2);
run;
```

In this example all the data are hardcoded in the statements of PROC NLP. In practice it is often desirable to separate the data from the model and let the model be data-driven. This is essential if you want to run the same model with different data or when the problem size becomes large. The following four examples use the INEST= or DATA= option to provide some degree of separation between the data and the model.

Example A.2. The INEST= option can be used for a data set that specifies the linear constraints and the initial values and bounds of the decision variables. The following SAS program adapted from Test Example 24 of Hock and Schittkowski (1981) demonstrates a typical usage of inputting data for a linearly constrained problem:

```
data betts(type=est);
   input _type_ $ x1 x2 _rhs_;
   datalines;
parms    1        .5       .
lowerbd  0        0        .
ge       .57735   -1       .
ge       1        1.732    .
le       1        1.732    6
;

proc nlp inest=betts;
   min y;
```

```
   parms x1 x2;
   y = (((x1 - 3)**2 - 9) * x2**3) / (27 * sqrt(3));
run;
```

The DATA= option in the PROC NLP statement can be used to input data to construct the objective function or nonlinear constraints. It resembles the DATA= option in a SAS data step in that it uses observation-driven processing of the data; that is, with each observation read from the data set, a term in the objective function or constraints is constructed. With the DATA= option, PROC NLP is well-suited for building a nonlinear regression model in which each residual function is constructed from the data in an observation.

Example A.3. The following SAS program uses the Bard function (Moré, Garbow, and Hillstrom 1981) to formulate a least squares problem with three parameters and 15 residual functions:

```
data bard;
   input u v w y;
   datalines;
 1  15  1  0.14
 2  14  2  0.18
 3  13  3  0.22
 4  12  4  0.25
 5  11  5  0.29
 6  10  6  0.32
 7   9  7  0.35
 8   8  8  0.39
 9   7  7  0.37
10   6  6  0.58
11   5  5  0.73
12   4  4  0.96
13   3  3  1.34
14   2  2  2.10
15   1  1  4.39
;

proc nlp data=bard;
   parms x1-x3 = 1;
   min f;
   f = 0.5*( y - (x1 + u/(v*x2 + w*x3)) )**2;
run;
```

As you can see, this is an unconstrained nonlinear least squares problem. When nonlinear constraints are present, the DATA= option can be used with the NLINCON statement to construct the nonlinear constraints. The NLINCON statement has the following syntax:

NLINCON *nonlin_con1 [, nonlin_con2 ...] [/ option]* **;**

option can be empty or have a value of SUMOBS or EVERYOBS. When *option* is empty, only the first observation in the data set specified via the DATA= option is used to generate the nonlinear constraints.

Example A.4. Suppose you add a nonlinear constraint to Example A.3 and you want to input the constants in the nonlinear constraint from the data set. The following SAS program shows how to incorporate these constants into the data set BARD in Example A.3 and use the DATA= option to input the data:

```
data bard1;
   input u v w y a0 a1 a2 a3 b1 b2 b3;
   datalines;
 1  15  1  0.14  127   2  3   1  2  4  1
 2  14  2  0.18    0   0  0   0  0  0  0
 3  13  3  0.22    0   0  0   0  0  0  0
 4  12  4  0.25    0   0  0   0  0  0  0
 5  11  5  0.29    0   0  0   0  0  0  0
 6  10  6  0.32    0   0  0   0  0  0  0
 7   9  7  0.35    0   0  0   0  0  0  0
 8   8  8  0.39    0   0  0   0  0  0  0
 9   7  7  0.37    0   0  0   0  0  0  0
10   6  6  0.58    0   0  0   0  0  0  0
11   5  5  0.73    0   0  0   0  0  0  0
12   4  4  0.96    0   0  0   0  0  0  0
13   3  3  1.34    0   0  0   0  0  0  0
14   2  2  2.10    0   0  0   0  0  0  0
```

```
15   1  1  4.39    0   0  0   0  0  0  0
;

proc nlp data=bard1;
   parms x1-x3 = 1;
   min f;
   nlincon con >= 0;
   f = 0.5*( y - (x1 + u/(v*x2 + w*x3)) )**2;
   con = a0 - a1*x1**b1 - a2*x2**b2 - a3*x3**b3;
run;
```

This approach results in inefficient use of the data set BARD1 because for variables A0 to B3, only the first observation of the data is used.

Example A.5. In many applications, the construction of a nonlinear objective function or nonlinear constraints requires access to the data across all the observations in the data set. Suppose you want to formulate and solve the following nonlinear optimization problem:

$$\begin{array}{ll} \text{maximize} & \displaystyle\sum_{i=1}^{5}\left((x_i - c_i)\exp\left(\sum_{j=1}^{5}\beta_{ij}x_j\right) + \sum_{j=1}^{5}\beta_{ji}x_j\right) \\ \text{subject to} & 0 \le x_i \le 10, \quad i = 1,\ldots,5 \end{array}$$

Using intermediate variables and IF _OBS_ = *num* THEN control logic, you can formulate this problem in PROC NLP as follows:

```
data mat;
   input beta1 beta2 beta3 beta4 beta5 cost;
   datalines;
-0.888  0.000  0.000  0.000  0.000  2.507
-0.264 -0.730  0.906 -0.104  0.000  0.972
-0.073  0.134 -0.955  0.000 -0.004  0.253
-0.142 -0.061  0.664 -0.718  0.008  4.925
 0.000  0.000  0.361  0.000 -0.417 11.045
;

proc nlp data=mat;
   array b[5] beta1-beta5;
   array x[5] x1-x5;
   array v[5]; /* intermediate variables */
   parms x1-x5 = .1;
   bounds 0 <= x1-x5 <= 10;
   max f;

   w = 0;
   do j = 1 to 5;
      w + b[j]*x[j];
      v[j] + b[j]*x[_obs_];
   end;
   f = (x[_obs_] - cost)*exp(w);
   if _obs_ = 5 then do;
      do j = 1 to 5;
         f + v[j];
      end;
   end;
run;
```

As you can immediately see, the logic in the preceding SAS program is not so intuitive and straightforward. For complex models it could easily lead to programming errors.

Given these limitations of the DATA= option in PROC NLP and several workarounds to build nonlinear objective functions and constraints from an input data set, you can see that you might not be able to formulate a more complex nonlinear optimization model in PROC NLP. For example, because of the way the NLINCON statement interacts with the DATA= option, you have to incorporate all the constants in the two nonlinear constraints into the first observation of the data set BARD in order to formulate the CNLSQ problem in Example A.1 by using the DATA= option in PROC NLP to input the data. This approach is impractical when the number of nonlinear constraints becomes large. In addition, lack of support for input of data in sparse matrix format in the INEST= and DATA= options further limits the modeling capabilities of PROC NLP, especially for large-scale problems.

The limitations of the modeling statements and options in PROC NLP (in particular, the INEST= and DATA= options) can be summarized as follows:

- You cannot input data from multiple data sets. This limitation results in inefficient use of data because different-sized data sets have to be combined into a single data set.
- You cannot access the data across all the observations in the data set.
- The NLINCON statement is too restrictive when it is used with the DATA= option to construct nonlinear constraints.
- There is no support for input of data in sparse matrix format.

HOW TO FORMULATE AN NLP IN PROC OPTMODEL

In PROC OPTMODEL, a modern algebraic modeling language is implemented for formulating general (both linear and nonlinear) optimization problems. A full exploration of the rich syntax and vast number of statements and expressions of PROC OPTMODEL is beyond the scope of this paper. This section focuses on the statements and expressions that are essential to formulate an optimization problem, particularly on the PROC OPTMODEL statements and expressions that correspond to the PROC NLP statements and options in Table 1 and Table 2.

Table 3 shows the PROC OPTMODEL declaration statements that are used to define the elements of an optimization problem.

Table 3 Declaration Statements in PROC OPTMODEL

Statement	Description
SET	Declares a set whose members are used for indexing variables or constants
NUMBER	Declares numeric constants
VAR	Declares decision variables
MIN | MAX	Declares the objective function of a minimization or maximization problem
CONSTRAINT	Declares constraints

The short form keywords for statements NUMBER and CONSTRAINT are NUM and CON respectively. The statements in Table 3 are much more versatile and capable than the PROC NLP statements in Table 1 in terms of their modeling capabilities. Table 4 maps the similar functionalities from PROC NLP to PROC OPTMODEL.

Table 4 Statements in PROC NLP versus PROC OPTMODEL

PROC NLP Statement	PROC OPTMODEL Statement
DECVAR	VAR *var_name* INIT *num*
BOUNDS	VAR *var_name* <= *num* >= *num*
MIN | MAX	MIN | MAX
LINCON	CONSTRAINT
NLINCON	CONSTRAINT
ARRAY	SET, NUMBER, VAR, or CONSTRAINT or any combination

In an optimization problem, the mathematical expressions of the objective function or constraints frequently contain summation or product operations, such as

$$\sum_{i=1}^{n} a_i x_i \qquad \text{or} \qquad \prod_{i=1}^{n} y_i$$

In PROC NLP DO loops are required for summation or product operations. The following SUM and PROD expressions in PROC OPTMODEL offer more natural ways to accomplish these tasks:

```
sum {i in 1..n} a[i]*x[i]
prod {i in 1..n} y[i]
```

PROC OPTMODEL also provides far better means for using external data than PROC NLP. The READ DATA statement in PROC OPTMODEL reads data from a data set and has the following features:

- Data can be imported from multiple data sets.
- Data are imported from data sets as matrices or one-dimensional arrays, allowing transparent access of the data.
- Data can be inputted in sparse matrix format.

These features enable the READ DATA statement in PROC OPTMODEL to overcome the limitations of the INEST= and DATA= options in PROC NLP.

Example B.1 through Example B.5 demonstrate the usage of the PROC OPTMODEL statements in Table 3 in formulating the same optimization problems as in Example A.1 through Example A.5. These examples show how easy it is to convert the existing PROC NLP programs to equivalent PROC OPTMODEL programs. The advantages of using PROC OPTMODEL are also highlighted.

Example B.1. You can formulate the CNLSQ problem in Example A.1 with PROC OPTMODEL as shown in the following SAS program:

```
proc optmodel;
   num l{1..2} = [-5 0];
   num u{1..2} = [100 1e+20];
   var x{i in 1..2} >= l[i] <= u[i] init 0.5;

   min f = (x[1]^2 - x[1]*x[2] + exp(3*x[1]))^2
         + (2*x[1]*x[2] + x[2]^2 + exp(2*x[2]))^2
         + (-3*x[1]*x[2] + exp(x[1] - 2*x[2]))^2;
   con c1: 2*x[1] - x[2] <= 1;
   con c2: x[1] + 4*x[2] >= 5;
   con c3: 2*x[1]^2 - 3*x[1]*x[2] + 4*x[2]^2 <= 8;
   con c4: 2*x[1] - x[2] - exp(x[1]^2) >= -1.5;

   solve;
   print x;
quit;
```

In the preceding PROC OPTMODEL program, the initial values and bounds of the decision variables are all specified in a single VAR statement. You specify $1e+20$ for the upper bound on x_2, and the PROC OPTMODEL nonlinear optimization algorithms effectively treat it as infinity. Note that the linear and nonlinear constraints are both specified with the CON statements, unlike in PROC NLP where separate LINCON and NLINCON statements are needed. The PRINT statement prints the values of the decision variables at the solution.

Example B.2. For the linearly constrained problem in Example A.2, you can break the data set BETTS into two separate data sets—one for the initial values and lower bounds of the decision variables and the other for the coefficients and right-hand sides of the linear constraints. Then you use three READ DATA statements to read the data into PROC OPTMODEL:

```
data betts1;
   input lb x0;
   datalines;
0  1
0 .5
;

data betts2;
   input c1 c2 c3;
   datalines;
.57735     -1  0
     1  1.732  0
    -1 -1.732 -6
;

proc optmodel;
   number a{1..3, 1..2};
   number b{1..3};
   number lb{1..2};
   number x0{1..2};
```

```
   var x{i in 1..2} >= lb[i] init x0[i];

   min f = ((x[1] - 3)^2 - 9)*x[2]^3/(27*sqrt(3));
   con lc {i in 1..3}: sum{j in 1..2} a[i,j]*x[j] >= b[i];

   read data betts1 into [_n_] lb x0;
   read data betts2 into [_n_] {j in 1..2}<a[_n_,j] = col("c"||j)>;
   read data betts2 into [_n_] b = c3;
   solve;
   print x;
quit;
```

Example B.3. You can formulate the unconstrained nonlinear least squares problem in Example A.3 using the same data set BARD as follows:

```
proc optmodel;
   set S = 1..15;
   number u{S};
   number v{S};
   number w{S};
   number y{S};
   var x{1..3} init 1;
   min r = .5*sum{i in S}(y[i] - (x[1] + u[i]/(v[i]*x[2] + w[i]*x[3])))^2;

   read data bard into [_n_] u v w y;
   solve;
   print x;
quit;
```

The preceding PROC OPTMODEL program uses the concept of an index set (that is, S = 1..15) for indexing the array constants U, V, W, and Y. This enables a much more direct statement of the objective function.

Example B.4. For the modified nonlinear least squares problem in Example A.4, you create a separate data set named NCPARM for the constants in the nonlinear constraint. You use this data set with the existing data set BARD from Example A.3 instead of combining them into a single data set, as shown in the following PROC OPTMODEL program:

```
data ncparm;
   input i c;
   datalines;
0 127
1   2
2   3
3   1
4   2
5   4
6   1
;

proc optmodel;
   set S = 1..15;
   number u{S};
   number v{S};
   number w{S};
   number y{S};
   number c{0..6};
   var x{1..3} init 1;
   min r = .5*sum{i in S}(y[i] - (x[1] + u[i]/(v[i]*x[2] + w[i]*x[3])))^2;
   con nc: c[1]*x[1]^c[2] + c[3]*x[2]^c[4] + c[5]*x[3]^c[6] <= c[0];

   read data bard into [_n_] u v w y;
   read data ncparm into [i] c;
   solve;
   print x;
quit;
```

It is easy to see that this approach results in more efficient use of the data sets than the approach in Example A.4. Note that the index set for C (an array constant) starts from 0 to account for the right-hand side value of the constraint.

Example B.5. The optimization problem in Example A.5 requires access to the data across all the observations in the data set. The PROC OPTMODEL READ DATA statement solves this problem by reading the data into a matrix, enabling easy access to the data in a very flexible manner as shown in the following program:

```
proc optmodel;
   set S = 1..5;
   num cost{S};
   num beta{S,S};
   var x{S} >= 0 <= 10 init .1;
   max f = sum{i in S}(
             (x[i] - cost[i])*exp(sum{j in S}(beta[i,j]*x[j]))
             + sum{j in S}(beta[j,i]*x[j])
           );
   read data mat into [_n_] cost;
   read data mat into [_n_] {j in S} < beta[_n_,j] = col("beta"||j) >;
   solve;
   print x;
quit;
```

No complicated programming logic is used. In fact, the MAX statement in the preceding program is almost a one-to-one "symbol-to-symbol" translation from the algebraic expressions of the objective function shown in Example A.5.

Example B.6. In contrast to the difficulty with using the DATA= option in PROC NLP to input the data to construct the nonlinear constraints in Examples A.1, the READ DATA statements in PROC OPTMODEL enable you to import data matrices of different sizes and formulate the CNLSQ problem with ease, as shown in the following program:

```
data objpar;
   input c1 c2 c3 c4 c5;
   datalines;
1 -1 0 3  0
0  2 1 0  2
0 -3 0 1 -2
;

data lcpar;
   input a0 a1 a2;
   datalines;
-1 -2  1
 5  1  4
;

data nlcpar;
   input b0 b1 b2 b3 b4 b5 b6;
   datalines;
  8 2 -3 4  0 0 0
1.5 0  0 0 -2 1 1
;

proc optmodel;
   num l{1..2} = [-5 0];
   num u{1..2} = [100 1e+20];
   num a{1..2, 0..2};
   num b{1..2, 0..6};
   num c{1..3, 1..5};
   var x{i in 1..2} >= l[i] <= u[i] init 0.5;

   min f = sum{i in 1..3}(c[i,1]*x[1]^2 + c[i,2]*x[1]*x[2]
             + c[i,3]*x[2]^2 + exp(c[i,4]*x[1] + c[i,5]*x[2]))^2;
   con lc{i in 1..2}: sum{j in 1..2}a[i,j]*x[j] >= a[i,0];
   con nlc{i in 1..2}: b[i,1]*x[1]^2 + b[i,2]*x[1]*x[2]
                       + b[i,3]*x[2]^2 + b[i,4]*x[1] + b[i,5]*x[2]
                       + b[i,6]*exp(x[1]^2) <= b[i,0];

   read data objpar into [_n_] {j in 1..5} < c[_n_,j] = col("c"||j) >;
   read data lcpar  into [_n_] {j in 0..2} < a[_n_,j] = col("a"||j) >;
   read data nlcpar into [_n_] {j in 0..6} < b[_n_,j] = col("b"||j) >;

   solve;
   print x;
quit;
```

Example B.7. The preceding examples show the effectiveness of the PROC OPTMODEL READ DATA statements when they are used to input data from data sets. The following example (adapted from Test Example 284 of Schittkowski 1987) further demonstrates how PROC OPTMODEL can import multiple data sets of different shapes (for example, one-dimensional arrays or matrices) and sizes:

```
data par_c;
   input c @@;
   datalines;
20 40 400 20 80 20 40 140 380 280 80 40 140 40 120
;

data par_b;
   input b @@;
   datalines;
385 470 560 565 645 430 485 455 390 460
;

data par_a;
   input c1-c15;
   datalines;
100 100 10  5  10 0   0  25 0  10 55 5   45 20 0
90  100 10  35 20 5   0  35 55 25 20 0   40 25 10
70  50  0   55 25 100 40 50 0  30 60 10  30 0  40
50  0   0   65 35 100 35 60 0  15 0  75  35 30 65
50  10  70  60 45 45  0  35 65 5  75 100 75 10 0
40  0   50  95 50 35  10 60 0  45 15 20  0  5  5
30  60  30  90 0  30  5  25 0  70 20 25  70 15 15
20  30  40  25 40 25  15 10 80 20 30 30  5  65 20
10  70  10  35 25 65  0  30 0  0  25 0   15 50 55
5   10  100 5  20 5   10 35 95 70 20 10  35 10 30
;

proc optmodel;
   set ROWS;
   set COLS;
   number c{COLS};
   number b{ROWS};
   number a{ROWS, COLS};
   var x{COLS} init 0;

   max f = sum {i in COLS} c[i]*x[i];
   con c1{i in ROWS}: sum {j in COLS} a[i,j]*x[j]^2 <= b[i];

   read data par_c into COLS=[_n_] c;
   read data par_b into ROWS=[_n_] b;
   read data par_a into [_n_] {j in COLS}< a[_n_,j] = col("c"||j) >;

   solve;
   print x;
quit;
```

Note that the preceding model is completely data-driven. The number of variables and number of constraints are determined by the number of observations in data sets PAR_C and PAR_B. This is achieved through the expressions COLS=[_N_] and ROWS=[_N_] in the READ DATA statements. Furthermore, all the coefficients of the objective function and constraints are read from the three data sets. Such a data-driven model can be readily run on data of any sizes.

ADDITIONAL MODELING ADVANTAGES OF PROC OPTMODEL

The previous section discusses various advantages that the PROC OPTMODEL modeling language offers. This section highlights additional advantages with PROC OPTMODEL: it supports input of data in a sparse matrix format, and it captures important structures of an optimization problem. The latter includes automatic recognition of the type of an objective function or a constraint and extraction of sparse patterns of the optimization problem.

PROC OPTMODEL provides support for the input of data in a sparse matrix format. This support facilitates formulation of large-scale optimization problems with PROC OPTMODEL. The following chemical equilibrium problem (adapted from Bracken and McCormick 1968) illustrates how to form a data set in the coordinate sparse matrix format and then use the READ DATA statement in PROC OPTMODEL to read the data into the linear constraint coefficient matrix. Note that this is another example of data-driven model that is ready to be expanded for large-scale problems.

```
data smat;
   input i j v;
   datalines;
1   1  1
1   2  2
1   3  2
1   6  1
1  10  1
2   4  1
2   5  2
2   6  1
2   7  1
3   3  1
3   7  1
3   8  1
3   9  2
3  10  1
;

data bvec;
   input b @@;
   datalines;
2  1  1
;

data cvec;
   input c @@;
   datalines;
 -6.089 -17.164 -34.054  -5.914 -24.721
-14.986 -24.100 -10.708 -26.662 -22.179
;

proc optmodel;
   set ROWS;
   set COLS;
   set <num,num> IJPair;
   num c{COLS};
   num a{IJPair};
   num b{ROWS};
   var x{COLS} >= 1.e-6 init .1;
   min y = sum{i in COLS}( x[i]*(c[i] + log(x[i]/sum{j in COLS}x[j])) );
   con cons{i in ROWS}: sum{j in slice(<i,*>, IJPair)}a[i,j]*x[j] = b[i];

   read data bvec into ROWS=[_n_] b;
   read data cvec into COLS=[_n_] c;
   read data smat into IJPair=[i j] a = v;
   solve;
   print x;
quit;
```

In the CON statement, the SLICE expression for set operation takes the summation over the nonzero elements of the sparse matrix A only.

In PROC NLP you must use the MINQUAD or MAXQUAD statement or the INQUAD= option to explicitly indicate a quadratic programming problem, and PROC NLP does not recognize a linear programming problem at all. In PROC NLP you must make the constraints explicit by using the LINCON or NLINCON statement. In contrast, you can use the MIN or MAX statement in PROC OPTMODEL, and the procedure automatically recognizes the type of the objective function as a linear, quadratic, or general nonlinear function. The following PROC OPTMODEL program shows a simple linear programming problem and a simple quadratic programming problem:

```
/* Linear program */
proc optmodel;
   set S = 1..2;
   num l{S} = [2 -50];
   num u{S} = [50 50];
   var x{i in S} >= l[i] <= u[i] init 1;

   min f = 0.4*x[1] + 4*x[2];
   con c1: x[1] - 10*x[2] <= 5;
   con c2: x[1] + 2*x[2] >= 10;
   solve;
quit;
```

```
/* Quadratic program */
proc optmodel;
   set S = 1..2;
   num l{S} = [2 -50];
   num u{S} = [50 50];
   var x{i in S} >= l[i] <= u[i] init -1;

   min f = 0.5*(0.4*x[1]^2 + 4*x[2]^2);
   con c1: 10*x[1] - x[2] >= 10;
   solve;
quit;
```

You can mix linear and nonlinear constraints in the CONSTRAINT statement, and PROC OPTMODEL automatically distinguishes between linear and nonlinear constraints. The following PROC OPTMODEL program shows an optimization problem with a linear constraint and a nonlinear constraint:

```
proc optmodel;
   var x1 <= .5;
   var x2 <= .5;
   min f = .5*((10*(x2 - x1*x1))^2 + (1 - x1)^2);
   /* c1 is linear and c2 is nonlinear */
   con c1: x1 + x2 <= .6;
   con c2: x1*x1 - 2*x2 >= 0;
   solve;
quit;
```

The ability of PROC OPTMODEL to recognize the type of an objective function and constraints is not only a modeling convenience, but it also means that PROC OPTMODEL can identify the problem class and choose the most appropriate algorithm by default. In the preceding three examples, the SOLVE statements are used without specifying a specific solver. This allows PROC OPTMODEL to automatically select appropriate algorithms.

Figure 1 is the PROC OPTMODEL output for the preceding linear programming problem. This output shows that PROC OPTMODEL recognizes the problem as a linear programming problem.

Figure 1 Output of Linear Program

The OPTMODEL Procedure

Problem Summary

Objective Sense	Minimization
Objective Function	f
Objective Type	Linear
Number of Variables	2
Bounded Above	0
Bounded Below	0
Bounded Below and Above	2
Free	0
Fixed	0
Number of Constraints	2
Linear LE (<=)	1
Linear EQ (=)	0
Linear GE (>=)	1
Linear Range	0

In Figure 2, the problem summary for the quadratic programming problem shows that PROC OPTMODEL recognizes the problem as a quadratic programming problem.

Figure 2 Output of Quadratic Program

```
The OPTMODEL Procedure

   Problem Summary

Objective Sense            Minimization
Objective Function                    f
Objective Type                Quadratic

Number of Variables                   2
Bounded Above                         0
Bounded Below                         0
Bounded Below and Above               2
Free                                  0
Fixed                                 0

Number of Constraints                 1
Linear LE (<=)                        0
Linear EQ (=)                         0
Linear GE (>=)                        1
Linear Range                          0
```

The problem summary in Figure 3 for the problem that involves a linear constraint and a nonlinear constraint shows that PROC OPTMODEL correctly identifies the types of the constraints.

Figure 3 Output of General Constrained Program

```
The OPTMODEL Procedure

   Problem Summary

Objective Sense            Minimization
Objective Function                    f
Objective Type                Nonlinear

Number of Variables                   2
Bounded Above                         2
Bounded Below                         0
Bounded Below and Above               0
Free                                  0
Fixed                                 0

Number of Constraints                 2
Linear LE (<=)                        1
Linear EQ (=)                         0
Linear GE (>=)                        0
Linear Range                          0
Nonlinear LE (<=)                     0
Nonlinear EQ (=)                      0
Nonlinear GE (>=)                     1
Nonlinear Range                       0
```

In addition to recognizing the types of an objective function and constraints, PROC OPTMODEL extracts sparse structural information from an optimization problem and provides a nonlinear optimization algorithm with input of the following in sparse formats:

- coefficient matrix of the linear constraints
- Jacobian of the nonlinear constraints
- Hessian of the objective function
- Hessian of the nonlinear constraints

This feature enables the PROC OPTMODEL nonlinear optimization algorithms to exploit sparsity and solve large-scale optimization problems. This is a significant advantage PROC OPTMODEL has over PROC NLP which uses dense formats to represent Jacobian and Hessian matrices.

Thus far the discussions have been focused on the modeling aspects of PROC NLP and PROC OPTMODEL. Modeling is a crucial part of nonlinear optimization. You must be able to accurately and effectively express a nonlinear optimization problem before you invoke an algorithm to solve it. The previous examples show how easy it is to convert existing PROC NLP programs to PROC OPTMODEL programs. The PROC OPTMODEL modeling language also has many other useful

features that make PROC OPTMODEL an ideal replacement for PROC NLP in formulating a nonlinear optimization problem. The following section discusses the nonlinear optimization algorithms available in PROC NLP and PROC OPTMODEL.

NONLINEAR OPTIMIZATION ALGORITHMS IN PROC NLP

PROC NLP implements 11 nonlinear optimization algorithms. These algorithms are accessible via the TECH= option. Table 5 summarizes these algorithms.

Table 5 PROC NLP Algorithms

TECH=	Description	Class of Problems Intended	SLA
QUADAS	Active set algorithm	General quadratic programming	–
LICOMP	Linear complementarity problem	Convex quadratic programming	–
NEWRAP	Newton-Raphson algorithm	Linearly constrained nonlinear optimization	–
NRRIDG	Newton-Raphson with ridging		–
TRUREG	Trust-region algorithm		–
DBLDOG	Double-dogleg algorithm		–
CONGRA	Conjugate gradient algorithm		–
QUANEW	Quasi-Newton algorithm	General constrained nonlinear optimization	–
NMSIMP	Nelder-Mead simplex algorithm		–
LEVMAR	Levenberg-Marquardt algorithm	Linearly constrained nonlinear least squares	–
HYQUAN	Hybrid quasi-Newton algorithm		–

In Table 5, the column “Class of Problems Intended” describes which class of problems an algorithm is intended for. In the column, linearly constrained nonlinear optimization or linearly constrained nonlinear least squares problems include their corresponding unconstrained and bound-constrained problems as special cases; and general constrained nonlinear optimization problems include unconstrained, bound-constrained, linearly constrained, and nonlinearly constrained problems. The column SLA indicates whether sparse linear algebra is supported in an algorithm, and “–” means that sparse linear algebra is not supported.

All the PROC NLP algorithms except NMSIMP are for smooth optimization problems and hence require first or second derivatives. PROC NLP uses automatic differentiation or finite difference approximation to compute these derivatives as input for the algorithms. The NMSIMP algorithm does not use any derivative information, nor does it assume that the objective or constraint functions have continuous derivatives. However, the algorithm does require the functions themselves to be continuous.

These 11 nonlinear optimization algorithms in PROC NLP represent classical nonlinear optimization theory and their implementation reflects the status of computer hardware at the time when PROC NLP was initially designed. Since then, both areas have advanced considerably. Consequently PROC NLP has fallen short significantly in the following aspects:

- lack of robustness in solving highly nonlinear problems
- lack of full support of sparse linear algebra

The lack of support of sparse linear algebra prevents PROC NLP from solving large-scale nonlinear optimization problems.

NONLINEAR OPTIMIZATION ALGORITHMS IN PROC OPTMODEL

The nonlinear optimization algorithms in PROC OPTMODEL are grouped under the following four solvers:

- NLPU – two algorithms for unconstrained and bound-constrained nonlinear optimization problems
- NLPC – four classical nonlinear optimization algorithms that are also available in PROC NLP
- SQP – a sequential quadratic programming algorithm
- IPNLP – two primal-dual interior point nonlinear optimization algorithms

The following SOLVE statement is used to invoke an optimization algorithm in PROC OPTMODEL:

SOLVE WITH *solver [/ **TECH=**tech [other_options]]* **;**

solver is one of the four solvers in the previous list. For NLPU, NLPC, or IPNLP, the TECH= option can be used to select a particular algorithm that belongs to the solver. Since the SQP solver has only one algorithm, the TECH= option does not apply to it. Table 6 summarizes nine optimization algorithms available in PROC OPTMODEL as of SAS/OR 9.22.

Table 6 PROC OPTMODEL Algorithms

Solver	TECH=	Description	Class of Problems Intended	SLA
NLPU	LBFGS	Limited-memory BFGS algorithm	Unconstrained nonlinear optimization	X
	CGTR	Conjugate gradient trust-region algorithm	Bound-constrained nonlinear optimization	X
NLPC	NEWTYP	Newton-Raphson algorithm	Linearly constrained nonlinear optimization	–
	TRUREG	Trust-region algorithm		–
	CONGRA	Conjugate gradient algorithm		–
	QUANEW	Quasi-Newton algorithm	General constrained nonlinear optimization	–
SQP	—	Sequential quadratic programming algorithm		–
IPNLP	IPQN	Quasi-Newton interior point algorithm		–
	IPKRYLOV	Iterative interior point algorithm		X

In Table 6, the connotations of the columns “Class of Problems Intended” and SLA are similar to those of Table 5. In addition, bound-constrained nonlinear optimization problems also include unconstrained problems, and “X” in the column SLA means that sparse linear algebra is supported.

All the nine PROC OPTMODEL algorithms are for smooth optimization problems and require first or second derivatives. PROC OPTMODEL uses automatic differentiation or finite difference approximation to compute these derivatives. The four algorithms in the NLPC solver (NEWTYP, TRUREG, CONGRA, and QUANEW) implement the same optimization algorithms as in PROC NLP. The CGTR algorithm in the NLPU solver, the SQP algorithm, and the IPQN and IPKRYLOV algorithms in the IPNLP solver have demonstrated robustness in solving highly nonlinear optimization problems. Among these four algorithms, the CGTR and IPKRYLOV algorithms use trust-region frameworks and use the true Hessian (second derivatives), and hence are well-suited for difficult nonlinear optimization problems. In addition, sparse linear algebra is fully supported in the CGTR and IPKRYLOV algorithms, and thus these two algorithms are capable of solving large-scale problems efficiently.

PERFORMANCE GAINS FROM USING ALGORITHMS IN PROC OPTMODEL

PROC OPTMODEL has the following advantages over PROC NLP:

- robustness in solving difficult highly nonlinear problems
- full support of sparse linear algebra, enabling it to solve large-scale problems efficiently

The robustness of the PROC OPTMODEL algorithms can be easily observed when PROC OPTMODEL is used to solve a nonlinear optimization problem with a large number of nonlinear constraints. Even for a nonlinear optimization problem that has no nonlinear constraint but whose objective function exhibits high nonlinearity, PROC OPTMODEL shows improved robustness in solving the problem. Table 7 shows the numerical results of the IPKRYLOV algorithm on 20 test problems selected from the well-known CUTEr testing library (Bongartz et al. 1995 and Gould, Orban, and Toint 2003).

Table 7 Numerical Results of IPKRYLOV on Selected CUTEr Test Problems

Name	Variables	Linear Constraints	Nonlinear Constraints	Iterations	CPU Time
aug3d	3873	1000	0	22	0:00:03.81
aug3dc	3873	1000	0	27	0:00:05.71
aug3dcqp	3873	1000	0	92	0:02:02.93
aug3dqp	3873	1000	0	80	0:01:43.93
broydn3d	10000	0	10000	11	0:00:08.20
broydn7d	1000	0	0	49	0:00:00.96
broydnbd	5000	0	5000	11	0:00:02.50
brybnd	5000	0	0	8	0:00:00.65
dtoc2	5998	0	3996	20	0:00:41.43
dtoc3	14999	9998	0	67	0:02:15.95
dtoc4	14999	4999	4999	22	0:01:40.42
dtoc5	9999	0	4999	25	0:00:59.14
hager1	10001	5001	0	10	0:00:14.39
hager2	10001	5000	0	21	0:00:22.62
hager3	10001	5000	0	18	0:00:22.01
hager4	10001	5000	0	38	0:01:45.96
sosqp1	20000	10001	0	1	0:00:34.35
sosqp2	20000	10001	0	166	0:39:03.95
sreadin3	10002	1	5000	17	0:03:48.72
srosenbr	10000	0	0	20	0:00:00.95

In Table 7 the CPU time was based on an Intel 2.4 GHz processor running under 32-bit Windows with 2 GB of RAM. As you can easily see, these are nontrivial instances of nonlinear optimization with the number of variables ranging from 1,000 to 20,000, and the number of linear or nonlinear constraints (or both) up to 10,000. The IPKRYLOV algorithm takes less than 200 iterations or a CPU time of less than 40 minutes in solving any of these instances. Most of these instances are solved in less than 50 iterations or within a few minutes. If you attempt to solve these test problems with PROC NLP, PROC NLP either runs out of memory or runs prohibitively slowly.

PROGRAMMING CAPABILITIES OF THE PROC OPTMODEL MODELING LANGUAGE

The rich syntax of the PROC OPTMODEL modeling language enables a SAS user to formulate a nonlinear optimization problem with ease and flexibility. The numerous expressions and statements available in PROC OPTMODEL also facilitate building customized solutions in which optimization algorithms in PROC OPTMODEL are called to solve a series of subproblems that are embedded in a larger solution context. For a complete list and description of these expressions and statements in PROC OPTMODEL, refer to the chapter "The OPTMODEL Procedure" in the *SAS/OR User's Guide: Mathematical Programming*. In the previous sections, you have seen many examples that use some of these expressions and statements. This section outlines additional useful PROC OPTMODEL expressions and statements.

The following lists four groups of PROC OPTMODEL expressions:

- *index-set* expression, of which several examples were shown in the section "HOW TO FORMULATE AN NLP IN PROC OPTMODEL"
- IF-THEN/ELSE expression; for example,

```
proc optmodel;
   var x{1..10};
   min f = sum{i in 1..10} if i<=5 then i*x[i] else x[i]^2;
   . . .
```

- expressions for set operations such as subsetting, UNION, and SLICE:

```
proc optmodel;
   . . .
   put ({i in 1..5 : i NE 3}); /* outputs {1,2,4,5} */
   put ({1,3} union {2,3}); /* outputs {1,3,2} */
   put (slice(<*,2,*>, {<1,2,3>, <2,4,3>, <2,2,5>})); /* outputs {<1,3>,<2,5>} */
```

- SUM and PROD expressions described in the section "HOW TO FORMULATE AN NLP IN PROC OPTMODEL"

In addition to the statements in Table 3 that describe the basic elements of an optimization problem, PROC OPTMODEL has the following enhanced statements that describe and select multiple optimization problems:

- PROBLEM *problem* associates an objective function, decision variables, constraints, and some status information with *problem*.
- USE PROBLEM *problem* makes *problem* the current problem for the solver.

PROC OPTMODEL has the following flow control statements:

- DO loop
- DO UNTIL loop
- DO WHILE loop
- FOR loop
- IF-THEN/ELSE block
- CONTINUE, LEAVE, and STOP for loop modification

To modify an existing optimization model, the following statements can be used:

- DROP *constraint* ignores the constraint.
- RESTORE *constraint* adds back the constraint that was dropped by the DROP statement.
- FIX *variable* treats the variable as fixed in value.
- UNFIX *variable* reverses the effect of the FIX statement.

The following statements are used to read or create a SAS data set:

- READ DATA
- CREATE DATA

In this section a number of commonly used expressions and statements in PROC OPTMODEL have been listed to show the versatility of the PROC OPTMODEL modeling language. The next section uses two examples to illustrate how to build customized solutions with some of these powerful expressions and statements.

BUILDING CUSTOMIZED SOLUTIONS WITH PROC OPTMODEL

This section contains two examples that illustrate how to build customized solutions with PROC OPTMODEL. The first example uses a simple multiple-start scheme to seek the global optimum of a maximization problem with multiple local maxima. The second example implements an algorithm for finding a feasible solution to a convex mixed integer nonlinear programming problem.

Example C.1. The following is the maximization problem:

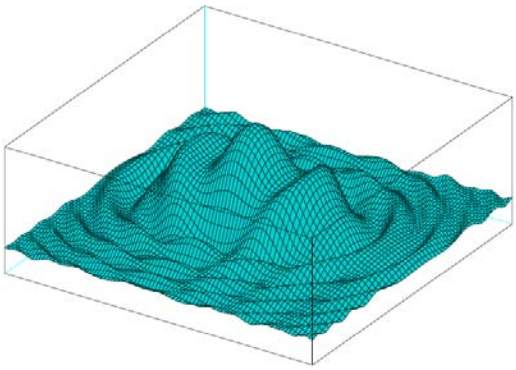
$$\begin{array}{ll} \text{maximize} & f(x,y) = \text{sinc}\,(x^2+y^2) + \text{sinc}\,((x-2)^2+y^2) \\ \text{subject to} & -10 \le x, y \le 10 \end{array}$$

where sinc(·), sometimes called the *sampling function*, is a function that frequently arises in signal processing. The canonical definition of the sampling function is as follows:

$$\text{sinc}(x) \equiv \begin{cases} 1 & \text{for } x = 0 \\ \dfrac{\sin x}{x} & \text{otherwise} \end{cases}$$

Figure 4 plots $f(x,y)$ on the vertical axis against x and y on the horizontal axes.

Figure 4 An Example of Multiple Local Minima



In order to formulate and solve this problem in PROC OPTMODEL, the following equivalent form of sinc(·) is used:

$$\text{sinc}(x) \equiv \begin{cases} 1 - \dfrac{x^2}{6} & \text{for } x = 0 \\ \dfrac{\sin x}{x} & \text{otherwise} \end{cases}$$

Note that in the previous form the value of the expression $1 - \frac{x^2}{6}$ for $x = 0$ and its first and second derivatives at $x = 0$ all match those in the canonical definition. The following PROC OPTMODEL program shows a customized solution that uses a simple multiple-start scheme to find the global maximum of $f(x, y)$ for $-10 \leq x, y \leq 10$:

```
proc optmodel printlevel=0;
   /* Set up for the multi-start algorithm */
   set grids = -10..10 by 20/6;
   num cnt init 0;
   num fb init -1.0e+30;
   num xb, yb;
   /* Define the optimization problem */
   var x >= -10 <= 10 init 0;
   var y >= -10 <= 10 init 0;
   max f = (if (x^2+y^2) ~= 0 then sin(x^2+y^2)/(x^2+y^2)
            else 1-(x^2+y^2)^2/6) +
           (if ((x-2)^2+y^2) ~= 0 then sin((x-2)^2+y^2)/((x-2)^2+y^2)
            else 1-((x-2)^2+y^2)^2/6);

   /* Call the default NLPC algorithm from different starting points */
   for {xs in grids, ys in grids} do;
      cnt = cnt + 1;
      x = xs;
      y = ys;
      solve with nlpc;
      if index(symget("_OROPTMODEL_"),'SOLUTION_STATUS=OPTIMAL')
            and f > fb then do;
         put "**** A better solution was found with f = " (f);
         fb = f;
         xb = x;
         yb = y;
      end;
   end;
   /* Print out summary information */
   print "A total of " (cnt) "different starting points were used";
   print "The best solution found is:";
   print "    f = " (fb);
   print "    x = " (xb) ",  y = " (yb);
quit;
```

The preceding program involves initially setting up for the multiple-start algorithm and defining the optimization problem. In the definition of the objective function, the IF-THEN/ELSE expressions are used to model the two cases of the sinc(·) function for $x = 0$ or otherwise. In the FOR loop, the default algorithm in the NLPC solver is called to solve for local maxima of the optimization problem, starting from different points. The IF-THEN block makes sure that a new solution is saved if the solution is optimal and its objective value is strictly better than that of the current solution. Finally summary

information is printed out as shown in Figure 5. The PRINTLEVEL=0 option of the PROC OPTMODEL statement is used to suppress printing of the default listing output by PROC OPTMODEL.

Figure 5 Output of Multiple-Start Scheme for Global Maximization

```
                    The OPTMODEL Procedure

          A total of    49    different starting points were used

                    The best solution found is:

                              f =    1.6829

                          x =    1    ,  y =    0
```

If you want to implement the multiple-start scheme using PROC NLP to solve the optimization subproblems, you have to use much more complicated programming logic and techniques to accomplish what the preceding PROC OPTMODEL program does.

Example C.2. This example illustrates a solution that implements an algorithm by Bonami et al. (2009), called the Feasibility Pump, for finding a feasible solution to a convex mixed integer nonlinear program. The mixed integer nonlinear program (MINLP) to be considered has the following constraints:

$$
\begin{aligned}
&(y_1 - \tfrac{1}{2})^2 + \tfrac{1}{4}(y_2 - 1)^2 \le \tfrac{1}{4} \\
&x - y_1 \le 0 \\
&y_2 \le \tfrac{1}{2} \\
&x \in \{0, 1\}
\end{aligned}
$$

Given an initial feasible solution $(\bar{x}^0, \bar{y}_1^0, \bar{y}_2^0) = (\frac{3}{4}, \frac{3}{4}, \frac{1}{2})$ to the relaxation of MINLP, at iteration $i \ge 1$, the algorithm finds a point $(\hat{x}^i, \hat{y}^i)$ by solving the following mixed integer linear program (SFOA):

$$
\begin{aligned}
\text{minimize} \quad & r + s \\
\text{subject to} \quad & (\bar{y}_1^k - \tfrac{1}{2})^2 + \tfrac{1}{4}(\bar{y}_2^k - 1)^2 + (2\bar{y}_1^k - 1)(y_1 - \bar{y}_1^k) + \tfrac{1}{2}(\bar{y}_2^k - 1)(y_2 - \bar{y}_2^k) \le \tfrac{1}{4}, \quad k = 0, \ldots, i-1 \\
& x - y_1 \le 0 \\
& (\bar{x}^k - \hat{x}^k)(x - \bar{x}^k) \ge 0, \quad k = 1, \ldots, i-1 \\
& x - \bar{x}^{i-1} = r - s \\
& y_2 \le \tfrac{1}{2} \\
& r, s \ge 0 \\
& x \in \{0, 1\}
\end{aligned}
$$

Then it solves the following nonlinear program (FP-NLP) for $(\bar{x}^i, \bar{y}^i)$:

$$
\begin{aligned}
\text{minimize} \quad & (x - \hat{x}^i)^2 \\
\text{subject to} \quad & (y_1 - \tfrac{1}{2})^2 + \tfrac{1}{4}(y_2 - 1)^2 \le \tfrac{1}{4} \\
& x - y_1 \le 0 \\
& 0 \le x \le 1 \\
& y_2 \le \tfrac{1}{2}
\end{aligned}
$$

The algorithm iterates between solving SFOA and FP-NLP until either a feasible solution to MINLP is found or SFOA becomes infeasible. The following SAS program shows how the Feasibility Pump is implemented in PROC OPTMODEL:

```
proc optmodel printlevel=0;
   set OASet init {0};
   num maxIters = 100;
   num Iter init 0;
   num xBar{OASet};
   num yBar{OASet, 1..2};
   num xHat{i in OASet: i > 0};
   var x binary;
   var y1, y2 <= 1/2;
   con lincon: x - y1 <= 0;
   /* Define the FP-NLP problem */
   min nlpObj = (x - xHat[Iter])^2;
```

```
   con nlncon: (y1-1/2)^2 + 1/4*(y2-1)^2 <= 1/4;
   problem fpnlp include x y1 y2 nlpObj lincon nlncon;
   /* Define the SFOA problem */
   var r >= 0, s >= 0;
   min oaObj = r + s;
   con oacons{i in OASet}: (yBar[i,1]-1/2)^2 + 1/4*(yBar[i,2]-1)^2 +
      (2*yBar[i,1]-1)*(y1-yBar[i,1]) + 1/2*(yBar[i,2]-1)*(y2-yBar[i,2]) <= 1/4;
   con cuts{i in  OASet: i > 0}: (xBar[i]-xHat[i])*(x-xBar[i]) >= 0;
   con auxcon: x - xBar[Iter-1] = r - s;
   problem sfoa include x y1 y2 r s oaObj lincon oacons cuts auxcon;

   xBar[Iter] = 3/4; yBar[Iter,1] = 3/4; yBar[Iter,2] = 1/2;
   do while (Iter < maxIters);
      Iter = Iter + 1;
      put "**** Iter " Iter ": Solving SFOA problem . . .";
      use problem sfoa;
      solve with milp / printfreq=0;
      /* Terminate if the problem is infeasible */
      if index(symget("_OROPTMODEL_"),'SOLUTION_STATUS=INFEASIBLE') then do;
         put "**** The problem is infeasible";
         leave;
      end;
      OASet = OASet union {Iter};
      xHat[Iter] = x;
      put "**** Iter " Iter ": Solving FP-NLP problem . . .";
      use problem fpnlp;
      solve with sqp relaxint / printfreq=0;
      /* Terminate if an integer solution was found */
      if index(symget("_OROPTMODEL_"),'SOLUTION_STATUS=OPTIMAL')
            and (abs(x) <= 1e-6 or abs(x-1) <= 1e-6) then do;
         put "**** An integer solution was found";
         x = round(x);
         leave;
      end;
      xBar[Iter] = x; yBar[Iter,1] = y1; yBar[Iter,2] = y2;
   end;
   print "The solution found is";
   print x y1 y2;
quit;
```

A number of features in the preceding implementation demonstrate the power of the PROC OPTMODEL modeling language in building sophisticated solution methods. The PROBLEM statement is used to declare the two optimization subproblems SFOA and FP-NLP. In the DO WHILE loop, the USE PROBLEM statements enable the algorithm to alternate between solving the two subproblems with ease. Furthermore, the index set OASET is initialized to a single element of 0 and expanded via the UNION operator on index sets. This allows the indexed constraints OACONS and CUTS to grow dynamically as the algorithm proceeds. Finally, SFOA is solved by the PROC OPTMODEL MILP solver and FP-NLP solved by the PROC OPTMODEL SQP solver. Since X is declared as a binary variable but FP-NLP is a continuous optimization problem, the RELAXINT keyword in the SOLVE statement is used to relax the integrality requirement. Figure 6 shows the output of the preceding PROC OPTMODEL program. As you can see, the algorithm finds an integer feasible solution with $x = 0$.

Figure 6 Output of Feasibility Pump for Mixed Integer Nonlinear Programming

```
                    The OPTMODEL Procedure

                    The solution found is

                  x         y1         y2

                  0    0.08311    0.49823
```

The Feasibility Pump algorithm requires solution of mixed integer linear programming subproblems. Hence it is impossible to implement the algorithm using PROC NLP to solve the optimization subproblems.

CONCLUSION

PROC OPTMODEL provides a versatile modeling environment in which its powerful programming expressions and statements make modeling complex optimization problems intuitive and efficient. In this paper, a number of examples illustrate how to migrate from PROC NLP to PROC OPTMODEL. In addition, PROC OPTMODEL enables SAS users to effectively model complex nonlinear optimization problems that are otherwise difficult to formulate in PROC NLP. The PROC OPTMODEL modeling language enables the important structures of a nonlinear optimization problem to be captured. The optimization algorithms in PROC OPTMODEL take advantage of this structural information and hence are capable of solving more difficult nonlinear optimization problems.

Furthermore, SAS users can take advantage of the programming capability of the PROC OPTMODEL modeling language and build solutions that are customized for their particular applications. Two examples demonstrate such customized solution methods. In conclusion, with its powerful modeling language and state-of-the-art optimization algorithms, PROC OPTMODEL is an ideal replacement for PROC NLP.

REFERENCES

Bonami, P., Cornuéjols, G., Lodi, A., and Margot, F. (2009), "A Feasibility Pump for Mixed Integer Nonlinear Programs," *Mathematical Programming*, 119:2, 331–352.

Bongartz, I., Conn, A. R., Gould, N. I. M., and Toint, Ph. L. (1995), "CUTE: Constrained and Unconstrained Testing Environment," *ACM Transactions on Mathematical Software*, 21:1, 123–160.

Bracken, J. and McCormick, G. P. (1968), *Selected Applications of Nonlinear Programming*, New York: John Wiley & Sons.

Gould, N. I. M., Orban, D., and Toint, Ph. L. (2003), "CUTEr and SifDec: A Constrained and Unconstrained Testing Environment, revisited," *ACM Transactions on Mathematical Software*, 29:4, 373–394.

Hock, W. and Schittkowski, K. (1981), *Test Examples for Nonlinear Programming Codes*, volume 187 of *Lecture Notes in Economics and Mathematical Systems*, Berlin-Heidelberg-New York: Springer-Verlag.

Moré, J. J., Garbow, B. S., and Hillstrom, K. E. (1981), "Testing Unconstrained Optimization Software," *ACM Transactions on Mathematical Software*, 7, 17–41.

Nocedal, J. and Wright, S. J. (1999), *Numerical Optimization*, New York: Springer-Verlag.

Schittkowski, K. (1987), *More Test Examples for Nonlinear Programming Codes*, volume 282 of *Lecture Notes in Economics and Mathematical Systems*, Berlin-Heidelberg-New York: Springer-Verlag.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Tao Huang
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
E-mail: Tao.Huang@sas.com

Ed Hughes
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
E-mail: Ed.Hughes@sas.com