SOLNP USERS' GUIDE

—A Nonlinear Optimization Program in MATLAB

By

Yinyu Ye

Department of Management Sciences

College of Business Administration

University of Iowa

Iowa City, Iowa 52242

August 1989

2. Table of Contents

1. Getting Started

This manual is a guide to solving various nonlinear optimization problems using a MATLAB optimization module. You will need a working knowledge of the MATLAB program language. You should feel comfortable with the following:

* Creating/Editing Text Files

* MATLAB Commands

* Saving and Loading Data

* Plotting.

Once familiar with MATLAB basics and SOLNP syntax, you should also review the benchmark examples in Section 5. Each example includes the following:

* Problem Formula

* Function Files

* Syntax Options

* Running Results.

2. SOLNP Algorithm

The SOLNP module in MATLAB solves the general nonlinear programming (NP) problem in the form

$$
\text{(NP)} \qquad \begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & g(x) = 0 \\ & l_h \le h(x) \le u_h \\ & l_x \le x \le u_x. \end{array}
$$

where $x \in R^n$, $f(x) : R^n \longrightarrow R$, $g(x) : R^n \longrightarrow R^{m_1}$, $h(x) : R^n \longrightarrow R^{m_2}$, $l_h,\ u_h \in R^{m_2}$ and $l_h < u_h$, and $l_x,\ u_x \in R^n$ and $l_x < u_x$. In general, f, g and h are any nonlinear smooth functions that may be specified and computed in MATLAB.

By adding slacks to the inequality constraints, the SOLNP algorithm converts the problem into

$$
\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & g(x) = 0 \\ & l_x \le x \le u_x. \end{array}
$$

The kth major iteration of SOLNP solves a linearly constrained optimization problem with an augmented Lagrangian objective function (Robinson [1]):

$$
(1) \qquad \begin{array}{ll} \text{minimize} & f(x) - y^k g(x) + (\rho/2)\|g(x)\|^2 \\ \text{subject to} & J^k(x - x^k) = -g(x^k) \\ & l_x \le x \le u_x. \end{array}
$$

where J^k is a numerical approximation to the Jacobian

$$
J^k = \frac{\partial g}{\partial x}|_{x^k}.
$$

and y^k is the vector of Lagrange multipliers at the kth major iteration ($y^0 = 0$ by default). Within the major iteration, the first step is to check to see if x^k is feasible for the linear equality constraints of (1). If it is not, an interior linear programming (LP) Phase 1 procedure is called to find an interior feasible (or near-feasible) solution.

Next, sequential quadratic programming (QP) is implemented to solve the linearly constrained problem (1), i.e., we calculate the gradient vector g and update the Hessian matrix H, using Broyden-Fletcher-Goldfarb-Shanno's technique, for the augmented Lagrangian objective function, and then we solve a quadratic problem

$$
(2) \qquad \begin{array}{ll} \text{minimize} & (1/2)(x - x^k)^T H (x - x^k) + g^T(x - x^k) \\ \text{subject to} & J^k(x - x^k) = -g(x^k) \\ & l_x \le x \le u_x. \end{array}
$$

Since there is no need to obtain highly accurate solution of (2), we use the interior QP algorithm to reach an approximate solution, which usually takes few steps. We refer to Ye [3] for technical details of an interior QP algorithm.

If the QP solution is both feasible and optimal for the linearly constrained problem (1), SOLNP starts the $(k+1)$th major iteration with x^{k+1} as the optimal solution and y^{k+1} as the optimal multiplier of the last QP problem during the kth major iteration; otherwise, it updates the gradient g and the Hessian H, and solves another QP problem that is referred as the minor iteration. Both major and minor processes repeat until the optimal solution is found or the user-specified maximum number of iterations is reached.

3. SOLNP Command Syntax

The complete syntax of the SOLNP command is:

$$[x, oh, y, h, ic] = solnp(xxb, icb, op, y0, h0)$$

Input

xxb: include the column vector of initial variable values (guesses) $x0$, or lower and upper variable bounds $[l_x, u_x]$, or all of them $[x0, l_x, u_x]$. If only $x0$ issued, we assume that there are no bounds for x; if $[l_x, u_x]$ is issued, we assume that the initial value of x is $x0 = (l_x + u_x)/2$. We also require that

$$l_x < x0 < u_x.$$

icb: include the column vectors of lower and upper bounds $[l_h, u_h]$, or $[i0, l_h, u_h]$, where l_h is the column vector of lower bounds, u_h is the column vector of upper bounds, and $i0$ is the column vector of the estimate values of the inequality constraints at the optimal solution. $i0$ should be strictly inside of the lower and upper bounds. If no inequality constraint is present, enter a zero in the icb position.

op: This is optional. op is a vector specifying the algorithm parameters defined as

$$[\rho, maj, min, \delta, \epsilon]$$

where

ρ the penalty parameter in the augmented Lagrangian objective function, default $= 1$.

maj the maximum number of major iterations, default $= 10$

min the maximum number of minor iterations, default = 10

δ the perturbation parameter for numerical gradient calculation, default = 10^{-5}

ϵ the relative tolerance on optimality and feasibility, default = 10^{-4}.

$y0$: the column vector of the estimate values of the optimal Lagrange multipliers for the constraints.

$h0$: the estimate Hessian matrix of the objective function at the optimal solution.

Output

x: the optimal solution.

oh: the history of the objective values over major iterations.

y : the optimal Lagrange multipliers.

h : the Hessian at the optimal solution.

ic: the optimal values of the inequality constraints.

Syntax variations include:

$$[x, oh] = solnp(xxb)$$

$$[x, oh, y, h, ic] = solnp(xxb, icb)$$

$$[x, oh, y, h, ic] = solnp(xxb, icb, op)$$

$$[x, oh, y, h, ic] = solnp(xxb, icb, op, y0, h0)$$

4. Running SOLNP

Running SOLNP is a straight forward process:

1. Create a function file named COST.M using any editor. Written in MATLAB command language, this function computes the objective and constraint function values.
2. Specify input variables and algorithm parameters and execute the SOLNP command in MATLAB. SOLNP calls COST.M many times.
3. Analyze the final results, and re-SOLNP if needed.

We now describe each of these steps in details.

4.1. Function File

This is a MATLAB user-defined function to return the values of the objective and constraint functions at an input x. It should be named COST.M.

The first line of the file is the MATLAB function declaration.

$function\ [f] = cost(x, par)$

x is the variable vector used to evaluate the function values, and par informs users that COST.M is called at a certain condition:

< 0 minor iteration completed,

$= 0$ gradient evaluation or line search,

> 0 major iteration completed.

This optional feature can be used to display the run time information on the screen.

The output f is a column vector: $f(1)$ is the value of the objective function evaluated at x, $f(2)$ through $f(m_1 + 1)$ are the values of the equality constraints, and $f(m_1 + 2)$ through $f(m_1 + m_2 + 1)$ are the values of the inequality constraint with the same order as icb in the SOLNP command syntax.

4.2. Inputs

Depending on the optimization problem to be solved, the SOLNP command varies.

1. *Unconstrained Optimization*

$$[x, oh] = solnp(x0)$$

where $x0$ is any starting values for x. However, a good guess will speed up the convergence.

2. *Equality Constrained Optimization*

$$[x, oh, y, h] = solnp(x0)$$

where $x0$ is any starting values for x. It need not to be feasible.

3. *Variable Bounded Optimization*

$$[x, oh, l, h] = solnp([x0, xb])$$

where xb is lower and upper bounds of x, and $x0$ is a starting values inside of xb.

4. *Inequality Constrained Optimization*

$$[x, oh, y, h, ic] = solnp(x0, icb)$$

where $x0$ is any starting values for x, and icb is the input of inequality constraints discussed in SOLNP Command Syntax.

5. *Equality and Inequality Constrained Optimization*

$$[x, oh, l, h, ic] = solnp(x0, icb)$$

where $x0$ is any starting values for x, and icb is the input of inequality constraints discussed in SOLNP Command Syntax. The equality constraints are implicitly inputed to SOLNP through COST.M file.

6. *General Constrained Optimization*

$$[x, oh, y, h, ic] = solnp([x0, xb], icb)$$

where $x0$ is any starting values inside xb, the lower and upper bounds for x. icb is the input of inequality constraints discussed in SOLNP Command Syntax. The equality constraints are implicitly inputed to SOLNP through COST.M file.

4.3. Parameter Specification

The algorithm parameters can be specified in the third input of SOLNP as shown in *op*, if users feel necessary. (In this case, if the optimization problem has no inequality constraints, issue 0 as the second input.)

ρ ρ is used as a penalty weighting scaler for infeasibility in the augmented objective function. The higher the value of ρ, the more emphasis will be put on bringing the solution into the feasible region. Be cautious when using large value of ρ, since it might make some numerically ill-conditioned or slow down the convergence.

maj SOLNP will terminate if the running number of major iterations reaches maj.

min Each major iteration will terminate if the running number of minor iterations reaches maj.

δ δ is used as the perturbation parameter for numerical gradient calculation. The gradient is evaluated as

$$\frac{\partial f}{\partial x_j} = \frac{f(x + \Delta e_j)}{\Delta}$$

where e_j is the jth unit vector, and

$$\Delta = \delta \max(|x_j|, 1).$$

ϵ The relative tolerance of feasibility and optimality.

4.4. Result Analysis

After the algorithm has completed the iterative process, you should be careful to check the final solution to see if it is feasible and optimal by common sense. There is no guarantee that the final solution is the global solution. The best way is to check to see if they meet the first and second order optimality conditions. Restart SOLNP if it terminates at the maximum number of major iterations.

4.5. Restart SOLNP

We recommend to run SOLNP first with default *op*, l^0 and h^0, and output l, h and ic (if exists) as well, i.e., issue

$$[x, oh, y, h, ic] = solnp([x0, l_x, u_x], [i0, l_h, u_h])$$

Then, if necessary, issue

$$[x, oh, y, h, ic] = solnp([x, l_x, u_x], [ic, l_h, u_h], op, y, h)$$

In the second run, the information of x, y, h and ic will be used in the following iterations. This gives SOLNP a warm start based on the first run. This essentially breaks a 20 iteration run into two 10 iteration runs without start from scratch in the second run.

5. Benchmark Examples

SOLNP has been tested for several benchmark problems in optimization area (Murtagh and Saunders [2]).

5.1. POWELL: this problem has only equality constraints

$$\begin{array}{ll} \text{min.} & \exp(x_1 x_2 x_3 x_4 x_5) \\ \text{s.t.} & x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 = 10 \\ & x_2 x_3 - 5 x_4 x_5 = 0 \\ & x_1^3 + x_2^3 = -1. \end{array}$$

The COST.M file is like:

$function\ [f] = cost(x, par)$
%
$f(1) = exp(x(1) * x(2) * x(3) * x(4) * x(5));$
$f(2) = x(1)^2 + x(2)^2 + x(3)^2 + x(4)^2 + x(5)^2 - 10;$
$f(3) = x(2) * x(3) - 5 * x(4) * x(5);$
$f(4) = x(1)^3 + x(2)^3 + 1;$
$f = f';$
$return$

The starting point is

$$x0 = (-2, 2, 2, -1, -1).$$

The SOLNP commands for $\rho = 1$ and $\rho = 0$, respectively, are

$\gg [x, oh, y] = solnp(x0);$

and

$\gg [x, oh, y] = solnp(x0, 0, 0);$

The convergence results of SOLNP vs MINOS:

	MINOS	SOLNP	
ρ	0	1	0
Major itns	4	5	5
Total itns	18	10	11

Table 1. POWELL Test Problem

ρ is the penalty parameter, Major itns is the number of major iterations used to converge, and Total itns is the number of total minor iterations used to converge.

5.2. WRIGHT4: this problem has only equality constraints

$$\begin{array}{ll} \text{min.} & (x_1-1)^2+(x_1-x_2)^2+(x_2-x_3)^3+(x_3-x_4)^4+(x_4-x_5)^4 \\ \text{s.t.} & x_1+x_2^2+x_3^3=2+3\sqrt{2} \\ & x_2-x_3^2+x_4=-2+2\sqrt{2} \\ & x_1x_5=2. \end{array}$$

The COST.M file is like:

$function\ [f]=cost(x,par)$
%
$f(1)=(x(1)-1)^2+(x(1)-x(2))^2+(x(2)-x(3))^3...$
$\quad +(x(3)-x(4))^4+(x(4)-x(5))^4;$
$f(2)=x(1)+x(2)^2+x(3)^3-2-3*sqrt(2);$
$f(3)=x(2)-x(3)^2+x(4)+2-2*sqrt(2);$
$f(4)=x(1)*x(5)-2;$
$f=f';$
$return$

The starting points:

$$x0=(1,1,1,1,1), \tag{a}$$

$$x0=(2,2,2,2,2), \tag{b}$$

$$x0=(-1,3,-1/2,-2,-3), \tag{c}$$

and

$$x0=(-1,2,1,-2,-2). \tag{d}$$

The SOLNP commands are

$\gg [x,oh,y]=solnp(x0);$

$\gg [x,oh,y]=solnp(x0,0,10);$

and

$\gg [x,oh,y]=solnp(x0,0,0);$

The local optimal points:

$$x^*=(1.1166,1.2205,1.5378,1.9727,1.7911), \tag{a}$$

$$x^*=(-2.7909,-3.0041,0.2054,3.8747,-0.7166), \tag{b}$$

$$x^*=(-1.273,2.4103,1.1949,-0.1542,-1.5710), \tag{c}$$

and

$$x^* = (-0.7034, 2.6357, -0.0964, -1.7980, -2.8434). \qquad (d)$$

The running results of SOLNP vs MINOS:

	SOLNP											
Start		(a)			(b)			(c)			(d)	
ρ	10	1	0	10	1	0	10	1	0	10	1	0
Major	6	6	6	4	4	4	6	6	4	8	4	5
Total	24	26	26	9	10	9	18	14	9	22	9	12
Solu.	(a)	(a)	(a)	(a)	(a)	(a)	(d)	(d)	(d)	(c)	(c)	(c)

Table 2. WRIGHT4 Test Problem

Start indicates where $x0$ is, Solu. indicates where x converges.

5.3. WRIGHT9: this problem has inequality constraints

$$\begin{array}{ll} \text{min.} & 10x_1x_4 - 6x_3x_2^2 + x_2x_1^3 + 9\sin(x_5 - x_3) + x_5^4x_4^2x_2^3 \\ \text{s.t.} & x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 \le 20 \\ & x_1^2x_3 + x_4x_5 \ge -2 \\ & x_2^2x_4 + 10x_1x_5 \ge 5. \end{array}$$

The COST.M file is like:

$function\ [f] = cost(x, par)$
%
$f(1) = 10 * x(1) * x(4) - 6 * x(3) * x(2)^2 + x(2) * x(1)^3 + ...$
$\quad 9 * sin(x(5) - x(3)) + (x(5)^4) * (x(4)^2) * (x(2)^3);$
$f(2) = norm(x(1:5))^2;$
$f(3) = x(3) * x(1)^2 + x(4) * x(5);$
$f(4) = x(4) * x(2)^2 + 10 * x(1) * x(5);$
$f = f';$
$return$

The starting points:

$$x0 = (1, 1, 1, 1, 1) \qquad (a)$$

and

$$x0 = (1.091, -3.174, 1.214, -1.614, 2.134). \qquad (b)$$

The SOLNP commands are

$\gg [x, oh, y] = solnp(x0, ib);$

and

$\gg [x, oh, y] = solnp(x0, ib, 100);$

where

$$ib = \begin{pmatrix} -100 & 20 \\ -2 & 100 \\ 5 & 100 \end{pmatrix}$$

The local optimal points:

$$x^* = (-0.0820, 3.6924, 2.4873, 0.3772, 0.1737) \qquad (a)$$

and

$$x^* = (1.4796, -2.6366, 1.0547, -1.6115, 2.6739). \qquad (b)$$

The running results of SOLNP vs MINOS:

	MINOS			SOLNP		
Start	(a)		(b)	(a)		(b)
ρ	100	10	100	100	1	100
Major	12	9	5	7	7	5
Total	92	71	32	45	32	18
Solu.	(a)	(a)	(b)	(a)	(a)	(b)

Table 3. WRIGHT9 Test Problem

5.4. BOX: this problem has one equality constraints and variable bounds

$$\begin{array}{ll} \text{min.} & -x_1x_2x_3 \\ \text{s.t.} & 4x_1x_2 + 2x_2x_3 + 2x_3x_1 = 100 \\ & 1 \le x_i \le 10 \text{ for } i = 1, 2, 3. \end{array}$$

The COST.M file is like:

$function\ [f] = cost(x, par)$
%
$f(1) = -x(1) * x(2) * x(3);$
$f(2) = 4 * x(1) * x(2) + 2 * x(2) * x(3) + 2 * x(3) * x(1) - 100;$
$f = f';$
$return$

The starting point:

$$x^0 = (1.1, 1.1, 9) \qquad (a)$$

and (the default)

$$x^0 = (5.5, 5.5, 5.5). \qquad (b)$$

The SOLNP command is

$\gg [x, oh, y] = solnp([x0 xb]);$

where

$$xb = \begin{pmatrix} 1 & 10 \\ 1 & 10 \\ 1 & 10 \\ 1 & 10 \end{pmatrix}.$$

The optimal solution is

$$x^* = (2.8856, 2.8856, 5.7779).$$

The convergence results of SOLNP :

	SOLNP	
Start	(a)	(b)
Major	8	4
Total	12	9

Table 4. BOX Test Problem

In general, starting from a "centered" solution (default) has a faster convergence due to the *interior* algorithm used in SOLNP.

5.5. ENTROPY: this is a nonconvex problem:

$$\begin{array}{ll} \text{min.} & -\sum(\ln x_i) - \ln(\|x - e\| + 0.1) \\ \text{s.t.} & e^T x = m \\ & x \geq 0 \end{array}$$

where $x \in R^m$ and e is the vector of all ones.

The COST.M file is like

```
function [f] = cost(x, par)
%
[m, n] = size(x);
f(1) = 0;
for i = 1 : m,
    f(1) = f(1) - log(x(i));
```

$end;$
$f(1) = f(1) - log(norm(x - ones(x)) + .1);$
$f(2) = sum(x) - m;$
$f = f';$
$return$

The SOLNP command is like

$\gg [x, oh, y] = solnp([x0, xb]);$

where x^0 is randomly generated between 0 and 1, and $m = 10$:

$$x0 = \begin{pmatrix} 0.8474 \\ 0.4524 \\ 0.8075 \\ 0.4832 \\ 0.6135 \\ 0.2749 \\ 0.8807 \\ 0.6538 \\ 0.4899 \\ 0.7741 \end{pmatrix}$$

and

$$xb = \begin{pmatrix} 0e & 10e \end{pmatrix}.$$

SOLNP converges to the following solution in 3 major iterations and 14 total minor iterations.

$$x^* = \begin{pmatrix} 0.8584 \\ 0.8568 \\ 0.8620 \\ 0.8566 \\ 0.8560 \\ 0.8606 \\ 2.2781 \\ 0.8556 \\ 0.8566 \\ 0.8594 \end{pmatrix}.$$

5.6. ALKYLA: this problem has both equality and inequality constraints, as well as the parameter bounds.

$$
\begin{array}{ll}
\text{min.} & -0.63 * x_4 * x_7 + 50.4 * x_1 + 3.5 * x_2 + x_3 + 33.6 * x_5 \\
\text{s.t.} & 98 * x_3 - 0.1 * x_4 * x_6 * x_9 - x_3 * x_6 = 0 \\
 & 1000 * x_2 + 100 * x_5 - 100 * x_1 * x_8 = 0 \\
 & 122 * x_4 - 100 * x_1 - 100 * x_5 = 0 \\
 & .99 \le (1.12 * x_1 + 0.13167 * x_1 * x_8 - 0.00667 * x_1 * x_8^2)/x_4 \le 100/99 \\
 & .99 \le (1.098 * x_8 - 0.038 * x_8^2 + 0.325 * x_6 + 57.25)/x_7 \le 100/99 \\
 & .9 \le (-0.222 * x_10 + 35.82)/x_9 \le 10/9 \\
 & .99 \le (3 * x_7 - 133)/x_{10} \le 100/99 \\
 & l_x \le x \le u_x
\end{array}
$$

where

$$l_x = (\, 0 \quad 0 \quad 0 \quad 10 \quad 0 \quad 85 \quad 10 \quad 3 \quad 1 \quad 145\,)^T$$

and

$$u_x = (\, 20 \quad 16 \quad 120 \quad 50 \quad 20 \quad 93 \quad 95 \quad 12 \quad 4 \quad 162\,)^T.$$

The COST.M file is like:

$function\ [f] = cost(x, par)$
%
$f(1) = -0.63 * x(4) * x(7) + 50.4 * x(1) + 3.5 * x(2) + x(3) + 33.6 * x(5);$
$f(2) = 98 * x(3) - 0.1 * x(4) * x(6) * x(9) - x(3) * x(6);$
$f(3) = 1000 * x(2) + 100 * x(5) - 100 * x(1) * x(8);$
$f(4) = 122 * x(4) - 100 * x(1) - 100 * x(5);$
$f(5) = (1.12 * x(1) + 0.13167 * x(1) * x(8) - 0.00667 * x(1) * x(8)^2)/x(4);$
$f(6) = (1.098 * x(8) - 0.038 * x(8)^2 + 0.325 * x(6) + 57.25)/x(7);$
$f(7) = (-0.222 * x(10) + 35.82)/x(9);$
$f(8) = (3 * x(7) - 133)/x(10);$
$f = f';$
$return$

The starting point:

$$x0 = (\, 17.45 \quad 12 \quad 110 \quad 30 \quad 19.74 \quad 89.2 \quad 92.8 \quad 8 \quad 3.6 \quad 155\,)^T.$$

The SOLNP command is

$\gg [x, oh, y] = solnp([x0, xb], ib, 0);$

where

$$xb = (l_x, u_x)$$

and

$$ib = \begin{pmatrix} .99 & 100/99 \\ .99 & 100/99 \\ .9 & 10/9 \\ .99 & 100/99 \end{pmatrix}.$$

The optimal point:

$$x^* = \begin{pmatrix} 16.9964 \\ 15.9994 \\ 57.6885 \\ 30.3249 \\ 20.0000 \\ 90.5654 \\ 95.0000 \\ 10.5901 \\ 1.5616 \\ 153.5353 \end{pmatrix}.$$

We used $\rho = 0$ to test the program. It converges to the solution in 6 major iterations and 25 total iterations.

6. SOLNP Diagnostics

This section explains the warning messages and error diagnostics in the SOLNP module. The messages should provide the necessary information to the user.

1. *Variable input must have three columns or less.*

Here, the input of the variables, xxb, has been specified with too many columns. See the SOLNP command syntax.

2. *Inequality constraint input must have three columns or less.*

Here, the input of the inequality constraints, icb, has been specified with too many columns. See the SOLNP command syntax.

3. *The lower bounds must be strictly less than the upper bounds.*

Here, one or more of the values in the lower bound vector is greater than or equal to the corresponding value in the upper bound vector.

4. *Initial variable values must be within the lower and upper bounds.*

Here, the initial values of $x0$ or $i0$ are not in the interior of their lower and upper bounds.

5. *COST function must return a column vector.*

SOLNP requires that the output of COST function in COST.M file must be a column vector. See the SOLNP command syntax.

6. *Algorithm parameter must be a vector.*

Here, the input of the algorithm parameters *op* is a matrix. It should be a row or column vector. See the SOLNP command syntax.

7. *The number of constraints returned from COST function does not match the number specified in the call to SOLNP.*

SOLNP found that the number of inequality constraints returned from COST function is no the same as the number of inequality constraints specified in *icb*.

8. *Redundant constraints were found. Poor intermediate result will occur. Remove redundant constraints and re-SOLNP.*

When this message occurs, the linearized Jacobian matrix of constraints is singular or near singular. This means that the matrix does not have full row- rank. The program will continue, but the final answers may be suspect. The user should review the constraint equations, remove any redundant constraints and re-SOLNP.

9. *SOLNP completed in # major iterations.*

The SOLNP algorithm has successfully solved the optimization problem in the indicated number of major iterations, satisfying the tolerance for optimality and feasibility.

10. *Exiting after maximum number of major iterations. Tolerance not achieved.*

After completing the specified maximum number of major iterations, SOLNP was not able to achieve required tolerance for optimality and feasibility. You may rerun SOLNP from the latest outputs. See Restart SOLNP.

11. *Suboptimization routine did not converge in specified number of minor iterations.*

This message tells you that SOLNP was not able to solve the linearly constrained (sub)optimization problem in the specified number of minor iterations. In most cases, this does not compromise the final solution. In fact, the limit of the number of minor iterations can be used to improve the overall efficiency. However, if SOLNP does not perform well, you should rerun SOLNP with a larger number of minor iterations in *op*.

12. *The linearized problem has no feasible solution, The problem may not be feasible.*

This message informs you that the linearized suboptimization problem has no feasible solution. This does not necessarily imply the original problem has no feasible solution. However, if this message appears repeatedly, the original problem may be infeasible.

7. References

[1] S. M. Robinson, "A quadratically convergent algorithm for general nonlinear programming problems," *Mathematical Programming* **3** (1972) 145-156.

[2] B. A. Murtagh and M. A. Saunders, "MINOS 5.1 user's guide," Technical Report SOL 83-20R, Department of OR, Stanford University (Stanford, CA, 1987).

[3] Y. Ye, "Interior algorithms for linear, quadratic, and linearly constrained nonlinear programming," Ph.D. Thesis, Department of EES, Stanford University (Stanford, CA, 1987).