GAMS/SNOPT: AN SQP ALGORITHM FOR LARGE-SCALE CONSTRAINED OPTIMIZATION

PHILIP E. GILL[*] WALTER MURRAY[†]
MICHAEL A. SAUNDERS[†] ARNE DRUD[‡]
ERWIN KALVELAGEN[§]

May 10, 2002

1 Introduction

This section describes the GAMS interface to the general-purpose NLP solver SNOPT, (Sparse Nonlinear Optimizer) which implements a sequential quadratic programming (SQP) method for solving constrained optimization problems with smooth nonlinear functions in the objective and constraints. The optimization problem is assumed to be stated in the form

$$\boxed{\begin{array}{lll} \text{NP} & \underset{x}{\text{minimize}} \text{ or } \underset{x}{\text{maximize}}\, f(x) & \\ & & F(x) \sim b_1 \\ & \text{subject to} & Gx \sim b_2 \\ & & l \le x \le u, \end{array}} \tag{1}$$

where $x \in \Re^n$, $f(x)$ is a linear or nonlinear smooth objective function, l and u are constant lower and upper bounds, $F(x)$ is a set of nonlinear constraint functions, G is a sparse matrix, $\sim$ is a vector of relational operators ($\le$, $\ge$ or $=$), and b_1 and b_2 are right-hand side constants. $F(x) \sim b_1$ are the nonlinear constraints of the model and $Gx \sim b_2$ form the linear constraints.

The gradients of f and F_i are automatically provided by GAMS, using its automatic differentiation engine.

[*]Department of Mathematics, University of California, San Diego, La Jolla, CA 92093-0112.

[†]Department of EESOR, Stanford University, Stanford, CA 94305-4023

[‡]ARKI Consulting and Development, Bagsvaerd, Denmark

[§]GAMS Development Corp., Washington DC

The bounds may include special values `-INF` or `+INF` to indicate $l_j = -\infty$ or $u_j = +\infty$ for appropriate j. Free variables have both bounds infinite and fixed variables have $l_j = u_j$.

1.1 Problem Types

If the nonlinear functions are absent, the problem is a *linear program* (LP) and SNOPT applies the primal simplex method [2]. Sparse basis factors are maintained by LUSOL [11] as in MINOS [14].

If only the objective is nonlinear, the problem is *linearly constrained* (LC) and tends to solve more easily than the general case with nonlinear constraints (NC). Note that GAMS models have an objective variable instead of an objective function. The GAMS/SNOPT link will try to substitute out the objective variable and reformulate the model such that SNOPT will see a true objective function.

For both linearly and nonlinearly constrained problems SNOPT applies a sparse sequential quadratic programming (SQP) method [6], using limited-memory quasi-Newton approximations to the Hessian of the Lagrangian. The merit function for steplength control is an augmented Lagrangian, as in the dense SQP solver NPSOL [7, 9].

In general, SNOPT requires less matrix computation than NPSOL and fewer evaluations of the functions than the nonlinear algorithms in MINOS [12, 13]. It is suitable for nonlinear problems with thousands of constraints and variables, but not thousands of degrees of freedom. (Thus, for large problems there should be many constraints and bounds, and many of them should be active at a solution.)

1.2 Selecting the SNOPT Solver

The GAMS system can be instructed to use the SNOPT solver by incorporating the following option in the GAMS model:

```
option NLP=SNOPT;
```

If the model contains non-smooth functions like abs(x), or max(x, y) you can try to get it solved by SNOPT using

```
option DNLP=SNOPT;
```

These models have discontinuous derivatives however, and SNOPT was not designed for solving such models. Discontinuities in the gradients can sometimes be tolerated if they are not too close to an optimum.

It is also possible to specify `NLP=SNOPT` or `DNLP=SNOPT` on the command line, as in:

```
 > gamslib chem
 > gams chem nlp=snopt
```

2 Description of the method

Here we briefly describe the main features of the SQP algorithm used in SNOPT and introduce some terminology. The SQP algorithm is fully described in [6].

2.1 Objective function reconstruction

The first step GAMS/SNOPT performs is to try to reconstruct the objective function. In GAMS, optimization models minimize or maximize an objective variable. SNOPT however works with an objective function. One way of dealing with this is to add a dummy linear function with just the objective variable. Consider the following GAMS fragment:

```
     obj.. z =e= sum(i, sqr(resid(i)));

     model m /all/;
     solve m using nlp minimizing z;
```

This can be cast in form (1) by saying minimize z subject to $z = \sum_i resid_i^2$ and the other constraints in the model. Although simple, this approach is not always preferable. Especially when all constraints are linear it is important to minimize $\sum_i resid_i^2$ directly. This can be achieved by a simple reformulation: z can be substituted out. The substitution mechanism carries out the formulation if all of the following conditions hold:

- the objective variable z is a free continuous variable (no bounds are defined on z),
- z appears linearly in the objective function,
- the objective function is formulated as an equality constraint,
- z is only present in the objective function and not in other constraints.

For many models it is very important that the nonlinear objective function be used by SNOPT. For instance the model `chem.gms` from the model library solves in 16 iterations. When we add the bound

```
   energy.lo = 0;
```

on the objective variable `energy` and thus preventing it from being substituted out, SNOPT will not be able to find a feasible point for the given starting point.

This reformulation mechanism has been extended for substitutions along the diagonal. For example, the GAMS model

```
   variables x,y,z;
   equations e1,e2;
   e1..z =e= y;
   e2..y =e= sqr(1+x);
   model m /all/;
   option nlp=snopt;
   solve m using nlp minimizing z;
```

will be reformulated as an *unconstrained* optimization problem

$$\text{minimize}\, f(x) = (1+x)^2.$$

These additional reformulations can be turned off by using the statement `option reform = 0;` (see §4.1).

2.2 Constraints and slack variables

The m general constraints of the problem (1) are formed by $F(x) \sim b_1$ and $Gx \sim b_2$. SNOPT will add to each general constraint a slack variable s_i with appropriate bounds. The problem defined by (1) can therefore be rewritten in the following equivalent form:

$$\begin{array}{ll} \underset{x,s}{\text{minimize}} & f(x) \\ \text{subject to} & \begin{pmatrix} F(x) \\ Gx \end{pmatrix} - s = 0, \quad l \le \begin{pmatrix} x \\ s \end{pmatrix} \le u. \end{array}$$

where a maximization problem is cast into a minimization by multiplying the objective function by -1.

The linear and nonlinear general constraints become equalities of the form $F(x) - s_N = 0$ and $Gx - s_L = 0$, where s_L and s_N are known as the *linear* and *nonlinear* slacks.

2.3 Major iterations

The basic structure of SNOPT's solution algorithm involves *major* and *minor* iterations. The major iterations generate a sequence of iterates (x_k) that satisfy the linear constraints and converge to a point that satisfies the first-order conditions for optimality. At each iterate $\{x_k\}$ a QP subproblem is used to generate a search direction towards the next iterate $\{x_{k+1}\}$. The constraints of the subproblem are formed from the linear constraints $Gx - s_L = 0$ and the nonlinear constraint linearization

$$F(x_k) + F'(x_k)(x - x_k) - s_N = 0,$$

where $F'(x_k)$ denotes the *Jacobian*: a matrix whose rows are the first derivatives of $F(x)$ evaluated at x_k. The QP constraints therefore comprise the m linear constraints

$$\begin{array}{rccl} F'(x_k)x & -s_N & & = -F(x_k) + F'(x_k)x_k, \\ Gx & & -s_L & = \;0, \end{array}$$

where x and s are bounded by l and u as before. If the $m \times n$ matrix A and m-vector b are defined as

$$A = \begin{pmatrix} F'(x_k) \\ G \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} -F(x_k) + F'(x_k)x_k \\ 0 \end{pmatrix},$$

then the QP subproblem can be written as

$$\underset{x,s}{\text{minimize}} \quad q(x) \quad \text{subject to} \quad Ax - s = b, \quad l \le \begin{pmatrix} x \\ s \end{pmatrix} \le u, \tag{2}$$

where $q(x)$ is a quadratic approximation to a modified Lagrangian function [6].

2.4 Minor iterations

Solving the QP subproblem is itself an iterative procedure, with the *minor* iterations of an SQP method being the iterations of the QP method. At each minor iteration, the constraints $Ax - s = b$ are (conceptually) partitioned into the form

$$Bx_B + Sx_S + Nx_N = b,$$

where the *basis matrix* B is square and nonsingular. The elements of x_B, x_S and x_N are called the *basic*, *superbasic* and *nonbasic* variables respectively; they are a permutation of the elements of x and s. At a QP solution, the basic and superbasic variables will lie somewhere between their bounds, while the nonbasic variables will be equal to one of their upper or lower bounds. At each iteration, x_S is regarded as a set of independent variables that are free to move in any desired direction, namely one that will improve the value of the QP objective (or the sum of infeasibilities). The basic variables are then adjusted in order to ensure that (x, s) continues to satisfy $Ax - s = b$. The number of superbasic variables (n_S say) therefore indicates the number of degrees of freedom remaining after the constraints have been satisfied. In broad terms, n_S is a measure of *how nonlinear* the problem is. In particular, n_S will always be zero at a solution for LP problems.

If it appears that no improvement can be made with the current definition of B, S and N, a nonbasic variable is selected to be added to S, and the process is repeated with the value of n_S increased by one. At all stages, if a basic or superbasic variable encounters one of its bounds, the variable is made nonbasic and the value of n_S is decreased by one.

Associated with each of the m equality constraints in $Ax - s = b$ are the *dual variables* π_i. Similarly, each variable in (x, s) has an associated *reduced gradient* d_j. The reduced gradients for the variables x are the quantities $g - A^T\pi$, where g is the gradient of the QP objective, and the reduced gradients for the slacks are the dual variables π. The QP subproblem is optimal if $d_j \ge 0$ for all nonbasic variables at their lower bounds, $d_j \le 0$ for all nonbasic variables at their upper bounds, and $d_j = 0$ for other variables, including superbasics. In practice, an *approximate* QP solution is found by relaxing these conditions on d_j (see the `Minor optimality tolerance` in §4.3).

2.5 The merit function

After a QP subproblem has been solved, new estimates of the NP solution are computed using a linesearch on the augmented Lagrangian merit function

$$\mathcal{M}(x,s,\pi) = f(x) - \pi^T\big(F(x)-s_N\big) + \tfrac{1}{2}\big(F(x)-s_N\big)^T D\big(F(x)-s_N\big), \qquad (3)$$

where D is a diagonal matrix of penalty parameters. If (x_k, s_k, π_k) denotes the current solution estimate and $(\widehat{x}_k, \widehat{s}_k, \widehat{\pi}_k)$ denotes the optimal QP solution, the linesearch determines a step α_k $(0 < \alpha_k \le 1)$ such that the new point

$$\begin{pmatrix} x_{k+1} \\ s_{k+1} \\ \pi_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ s_k \\ \pi_k \end{pmatrix} + \alpha_k \begin{pmatrix} \widehat{x}_k - x_k \\ \widehat{s}_k - s_k \\ \widehat{\pi}_k - \pi_k \end{pmatrix} \qquad (4)$$

gives a *sufficient decrease* in the merit function. When necessary, the penalties in D are increased by the minimum-norm perturbation that ensures descent for $\mathcal{M}$ [9]. As in NPSOL, s_N is adjusted to minimize the merit function as a function of s prior to the solution of the QP subproblem. For more details, see [7, 3].

2.6 Treatment of constraint infeasibilities

SNOPT makes explicit allowance for infeasible constraints. Infeasible linear constraints are detected first by solving a problem of the form

$$\boxed{\begin{array}{lll} \text{FLP} & \underset{x,v,w}{\text{minimize}} & e^T(v+w) \\ & \text{subject to} & Gx - v + w \sim b_2, \quad l \le x \le u, \quad v,w \ge 0, \end{array}}$$

where e is a vector of ones. This is equivalent to minimizing the sum of the general linear constraint violations subject to the simple bounds. (In the linear programming literature, the approach is often called elastic programming. We also describe it as minimizing the ℓ_1 norm of the infeasibilities.)

If the linear constraints are infeasible ($v \ne 0$ or $w \ne 0$), SNOPT terminates without computing the nonlinear functions.

If the linear constraints are feasible, all subsequent iterates satisfy the linear constraints. (Such a strategy allows linear constraints to be used to define a region in which the functions can be safely evaluated.) SNOPT proceeds to solve NP as given, using search directions obtained from a sequence of quadratic programming subproblems (2).

If a QP subproblem proves to be infeasible or unbounded (or if the dual variables π for the nonlinear constraints become large), SNOPT enters "elastic" mode and solves the problem

$$\boxed{\begin{array}{lll} \text{NP}(\gamma) & \underset{x,v,w}{\text{minimize}} & f(x) + \gamma e^T(v+w) \\ & \text{subject to} & \begin{pmatrix} F(x) - v + w \\ Gx \end{pmatrix} \sim b, \quad l \le x \le u, \quad v,w \ge 0, \end{array}}$$

where γ is a nonnegative parameter (the *elastic weight*), and $f(x)+\gamma e^T(v+w)$ is called a *composite objective*. If γ is sufficiently large, this is equivalent to minimizing the sum of the nonlinear constraint violations subject to the linear constraints and bounds. A similar ℓ_1 formulation of NP is fundamental to the Sℓ_1QP algorithm of Fletcher [4]. See also Conn [1].

3 Starting points and advanced bases

A good starting point may be essential for solving nonlinear models. We show how such a starting point can be specified in a GAMS environment, and how SNOPT will use this information.

A related issue is the use of "restart" information in case a number of related models is solved in a row. Starting from an optimal point from a previous solve statement is in such situations often beneficial. In a GAMS environment this means reusing primal and dual information, which is stored in the `.L` and `.M` fields of variables and equations.

3.1 Starting points

To specify a starting point for SNOPT use the `.L` level values in GAMS. For example, to set all variables $x_{i,j} := 1$ use `x.l(i,j)=1;`. The default values for level values are zero.

Setting a good starting point can be crucial for getting good results. As an (artificial) example consider the problem where we want to find the smallest circle that contains a number of points (x_i, y_i):

$$\boxed{\text{Example} \qquad \begin{array}{ll} \underset{r,a,b}{\text{minimize}} & r \\ \text{subject to} & (x_i-a)^2+(y_i-b)^2 \le r^2, \quad r \ge 0. \end{array}}$$

This problem can be modeled in GAMS as follows.

```
set i points /p1*p10/;

parameters
    x(i)    x coordinates,
    y(i)    y coordinates;

* fill with random data
x(i) = uniform(1,10);
y(i) = uniform(1,10);

variables
    a       x coordinate of center of circle
    b       y coordinate of center of circle
    r       radius;

equations
    e(i)    points must be inside circle;


e(i).. sqr(x(i)-a) + sqr(y(i)-b) =l= sqr(r);
```

```
r.lo = 0;

model m /all/;
option nlp=snopt;
solve m using nlp minimizing r;
```

Without help, SNOPT will not be able to find an optimal solution. The problem will be declared infeasible. In this case, providing a good starting point is very easy. If we define

$$\begin{aligned} x_{\min} &= \min_i x_i, \\ y_{\min} &= \min_i y_i, \\ x_{\max} &= \max_i x_i, \\ y_{\max} &= \max_i y_i, \end{aligned}$$

then good estimates are

$$\begin{aligned} a &= (x_{\min} + x_{\max})/2, \\ b &= (y_{\min} + y_{\max})/2, \\ r &= \sqrt{(a - x_{\min})^2 + (b - y_{\min})^2}. \end{aligned}$$

Thus we include in our model:

```
parameters xmin,ymin,xmax,ymax;
xmin = smin(i, x(i));
ymin = smin(i, x(i));
xmax = smax(i, x(i));
ymax = smax(i, y(i));

* set starting point
a.l = (xmin+xmax)/2;
b.l = (ymin+ymax)/2;
r.l = sqrt( sqr(a.l-xmin) + sqr(b.l-ymin) );
```

and now the model solves very easily.

Level values can also be set implicitly as a result of assigning bounds. When a variable is bounded away from zero, for instance by the statement `Y.LO = 1;`, the `SOLVE` statement will override the default level of zero of such a variable in order to make it feasible.

Note: another way to formulate the model would be to minimize r^2 instead of r. This allows SNOPT to solve the problem even with the default starting point.

3.2 Advanced basis

GAMS automatically passes on level values and basis information from one solve to the next. Thus, when we have two solve statements in a row, with just a few changes in between SNOPT will typically need very few iterations to find an optimal solution in the second solve. For instance, when we add a second solve to the `chem.gms` model from the model library:

```
model mixer chemical mix for N2H4+O2 / all /;

solve mixer minimizing energy using nlp;
solve mixer minimizing energy using nlp;
```

we observe the following log:

```
[erwin@hamilton]$ gams chem nlp=snopt
GAMS 2.50A    Copyright (C) 1987-1999 GAMS Development. All rights reserved
Licensee: hamilton                                       G990622:1048CP-LNX
          GAMS Development
--- Starting compilation
--- chem.gms(48) 1 Mb
--- Starting execution
--- chem.gms(42) 1 Mb
--- Generating model MIXER
--- chem.gms(46) 1 Mb
---    5 rows, 12 columns, and 37 non-zeroes.
--- Executing SNOPT

    GAMS/SNOPT    X86/LINUX  version 5.3.4-007-035
    P. E. Gill, UC San Diego
    W. Murray and M. A. Saunders, Stanford University

 Work space allocated           --    0.02 Mb

 Major Minor   Step   nObj   Objective    Optimal    nS PD
     0     5 0.0E+00     1  3.292476E+01 2.1E-01    0 TF   r
     1     4 1.0E+00     2  4.517191E+01 2.2E-01    1 TF n r
     2    10 8.6E-02     5  4.533775E+01 1.7E-01    6 TF s
     3     3 1.0E+00     6  4.608439E+01 7.6E-02    6 TF
     4     2 1.0E+00     7  4.667813E+01 1.4E-01    5 TF
     5     2 1.0E+00     8  4.751149E+01 2.2E-02    6 TF
     6     3 1.0E+00     9  4.757024E+01 2.1E-02    4 TF
     7     2 7.1E-01    11  4.763634E+01 3.3E-02    5 TF
     8     3 1.0E+00    12  4.768395E+01 3.3E-02    7 TF
     9     3 4.6E-01    14  4.769958E+01 1.9E-02    5 TF
    10     2 1.0E+00    15  4.770539E+01 1.5E-02    6 TF

 Major Minor   Step   nObj   Objective    Optimal    nS PD
    11     1 1.0E+00    16  4.770639E+01 6.2E-04    6 TF
    12     1 1.0E+00    17  4.770650E+01 5.0E-03    6 TF
    13     1 1.0E+00    18  4.770651E+01 1.6E-04    6 TF
    14     1 1.0E+00    19  4.770651E+01 1.8E-05    6 TF
    15     1 1.0E+00    20  4.770651E+01 2.7E-05    6 TF
    16     1 1.0E+00    21  4.770651E+01 7.6E-07    6 TT

 EXIT - Optimal Solution found.

--- Restarting execution
--- chem.gms(46) 1 Mb
--- Reading solution for model MIXER
--- chem.gms(46) 1 Mb
--- Generating model MIXER
--- chem.gms(47) 1 Mb
---    5 rows, 12 columns, and 37 non-zeroes.
--- Executing SNOPT

    GAMS/SNOPT    X86/LINUX  version 5.3.4-007-035
    P. E. Gill, UC San Diego
    W. Murray and M. A. Saunders, Stanford University

 Work space allocated           --    0.02 Mb

 Major Minor   Step   nObj   Objective    Optimal    nS PD
```

```
     0     0 0.0E+00     1  4.770651E+01 7.4E-07    0 TT   r

 EXIT - Optimal Solution found.

--- Restarting execution
--- chem.gms(47) 1 Mb
--- Reading solution for model MIXER
--- chem.gms(47) 1 Mb
*** Status: Normal completion
[erwin@hamilton]$
```

The first `solve` takes 16 iterations, while the second `solve` needs exactly zero iterations.

Basis information is passed on using the marginals of the variables and equations. In general the rule is:

`X.M` $= 0$	basic
`X.M` $\neq 0$	nonbasic if level value is at bound, superbasic otherwise

A marginal value of `EPS` means that the numerical value of the marginal is zero, but that the status is nonbasic or superbasic. The user can specify a basis by assigning zero or nonzero values to the `.M` values. It is further noted that if too many `.M` values are zero, the basis is rejected. This happens for instance when two subsequent models are too different. This decision is made based on the value of the `bratio` option (see §4.1).

4 Options

In many cases NLP models can be solved with GAMS/SNOPT without using solver options. For special situations it is possible to specify non-standard values for some or all of the options.

4.1 GAMS options

The following GAMS options affect the behavior of SNOPT.

NLP
: This option selects the NLP solver. Example: `option NLP=SNOPT;`. See also §1.2.

DNLP
: Selects the DNLP solver for models with discontinuous or non-differentiable functions. Example: `option DNLP=SNOPT;`. See also §1.2.

RMINLP
: Selects the Relaxed Non-linear Mixed-Integer (RMINLP) solver. By relaxing the integer conditions in an MINLP, the model becomes effectively an NLP. Example: `option RMINLP=SNOPT;`. See also §1.2.

iterlim

Sets the (minor) iteration limit. Example: `option iterlim=50000;`. The default is 10000. SNOPT will stop as soon as the number of *minor iterations* exceeds the iteration limit. In that case the current solution will be reported.

reslim

Sets the time limit or resource limit. Depending on the architecture this is wall clock time or CPU time. SNOPT will stop as soon as more than *reslim* seconds have elapsed since SNOPT started. The current solution will be reported in this case. Example: `option reslim = 600;`. The default is 1000 seconds.

domlim

Sets the domain violation limit. Domain errors are evaluation errors in the nonlinear functions. An example of a domain error is trying to evaluate $\sqrt{x}$ for $x < 0$. Other examples include taking logs of negative numbers, and evaluating x^y for $x < 0$ (x^y is evaluated as $\exp(y \log x)$). When such a situation occurs the number of domain errors is increased by one, and SNOPT will stop if this number exceeds the limit. If the limit has not been reached, a reasonable number is returned (e.g., in the case of $\sqrt{x}, x < 0$ a zero is passed back) and SNOPT is asked to continue. In many cases SNOPT will be able to recover from these domain errors, especially when they happen at some intermediate point. Nevertheless it is best to add appropriate bounds or linear constraints to ensure that these domain errors don't occur. For example, when an expression $\log(x)$ is present in the model, add a statement like `x.lo = 0.001;`. Example: `option domlim=100;`. The default value is 0.

bratio

Basis acceptance test. When several models are solved in a row, GAMS automatically passes dual information to SNOPT so that it can reconstruct an advanced basis. When too many new variables or constraints enter the model, it may be better not to use existing basis information, but to *crash* a new basis instead. The `bratio` determines how quickly an existing basis is discarded. A value of 1.0 will discard any basis, while a value of 0.0 will retain any basis. Example: `option bratio=1.0;`. Default: bratio = 0.25.

sysout

Debug listing. When turned on, extra information printed by SNOPT will be added to the listing file. Example: `option sysout=on;`. Default: sysout = off.

work

The `work` option sets the amount of memory SNOPT can use. By default an estimate is used based on the model statistics (number of (nonlinear) equations, number of (nonlinear) variables, number of (nonlinear) nonzeroes etc.). In most cases this is sufficient to solve the model. In some extreme cases SNOPT may need more memory, and the user can specify this with this option. For historical reasons `work` is specified in "double words" or 8 byte quantities. For example, `option work=100000;` will ask for 0.76 MB (a megabyte being defined as 1024×1024 bytes).

reform

This option will instruct the reformulation mechanism described in §2.1 to substitute out equality equations. The default value of 100 will cause the procedure to try further substitutions along the diagonal after the objective variable has been removed. Any other value will prohibit this diagonal procedure. Example: `option reform = 0;`. Default: reform = 100.

4.2 Model suffices

A model identifier in GAMS can have several suffices to which you can assign values. A small GAMS fragment illustrates this:

```
model m /all/;
m.iterlim = 3000;
solve m minimizing z using nlp;
```

Options set by assigning to the suffixed model identifier override the global options. For example,

```
model m /all/;
m.iterlim = 3000;
option iterlim = 100;
solve m minimizing z using nlp;
```

will use an iteration limit of 3000 for this solve.

m.iterlim

Sets the iteration limit. Overrides the global iteration limit. Example: `m.iterlim=50000;` The default is 10000. See also §4.1.

m.reslim

Sets the resource or time limit. Overrides the global resource limit. Example: `m.reslim=600;` The default is 1000 seconds. See also §4.1.

m.bratio

Sets the basis acceptance test parameter. Overrides the global setting. Example: `m.bratio=1.0;` The default is 0.25. See also §4.1.

m.scaleopt

Whether or not to scale the model using user-supplied scale factors. The user can provide scale factors using the `.scale` variable and equation suffix. For example, `x.scale(i,j) = 100;` will assign a scale factor of 100 to all $x_{i,j}$ variables. The variables SNOPT will see are scaled by a factor 1/*variable_scale*, so the modeler should use scale factors that represent the order of magnitude of the variable. In that case SNOPT will see variables that are scaled around 1.0. Similarly equation scales can be assigned to equations, which are scaled by a factor 1/*equation_scale*. Example: `m.scaleopt=1;` will turn scaling on. The default is not to use scaling, and the default scale factors are 1.0. Automatic scaling is provided by the SNOPT option `scale option`.

m.optfile

Sets whether or not to use a solver option file. Solver specific SNOPT options are specified in a file called `snopt.opt`, see §4.3. To tell SNOPT to use this file, add the statement: `option m.optfile=1;`. The default is not to use an option file.

m.workspace

The workspace option sets the amount of memory that SNOPT can use. By default an estimate is used based on the model statistics (number of (nonlinear) equations, number of (nonlinear) variables, number of (nonlinear) nonzeroes, etc.). In most cases this is sufficient to solve the model. In some extreme cases SNOPT may need more memory, and the user can specify this with this option. The amount of memory is specified in MB. Example: `m.workspace = 5;`.

4.3 SNOPT options

SNOPT options are specified in a file called `snopt.opt` which should reside in the working directory (this is the project directory when running models from the IDE). To tell SNOPT to read this file, use the statement `m.optfile = 1;` in the model (see §4.2).

An example of a valid `snopt.opt` is:

```
Hessian full memory
Hessian frequency 20
```

Users familiar with the SNOPT distribution from Stanford University will notice that the `begin` and `end` keywords are missing. These markers are optional in GAMS/SNOPT and will be ignored. Therefore, the following option file is also accepted:

```
begin
Hessian full memory
Hessian frequency 20
end
```

All options are case-insensitive. A line is a comment line if it starts with an asterisk, `*`, in column one.

Here follows a description of all SNOPT options that are possibly useful in a GAMS environment:

Check frequency i

Every ith minor iteration after the most recent basis factorization, a numerical test is made to see if the current solution x satisfies the general linear constraints (including linearized nonlinear constraints, if any). The constraints are of the form $Ax - s = b$, where s is the set of slack variables. To perform the numerical test, the residual vector $r = b - Ax + s$ is computed. If the largest component of r is judged to be too large, the current basis is refactorized and the basic variables are recomputed to satisfy the general constraints more accurately.

`Check frequency 1` is useful for debugging purposes, but otherwise this option should not be needed.

Default: `check frequency 60`.

Crash option i

Except on restarts, a CRASH procedure is used to select an initial basis from certain rows and columns of the constraint matrix $(A \quad -I)$. The `Crash option` i determines which rows and columns of A are eligible initially, and how many times CRASH is called. Columns of $-I$ are used to pad the basis where necessary.

`Crash option 0:` The initial basis contains only slack variables: $B = I$.

`Crash option 1:` CRASH is called once, looking for a triangular basis in all rows and columns of the matrix A.

`Crash option 2:` CRASH is called twice (if there are nonlinear constraints). The first call looks for a triangular basis in linear rows, and the iteration proceeds with simplex iterations until the linear constraints are satisfied. The Jacobian is then evaluated for the first major iteration and CRASH is called again to find a triangular basis in the nonlinear rows (retaining the current basis for linear rows).

`Crash option 3:` CRASH is called up to three times (if there are nonlinear constraints). The first two calls treat *linear equalities* and *linear inequalities* separately. As before, the last call treats nonlinear rows before the first major iteration.

If $i \geq 1$, certain slacks on inequality rows are selected for the basis first. (If $i \geq 2$, numerical values are used to exclude slacks that are close to a bound.) CRASH then makes several passes through the columns of A, searching for a basis matrix that is essentially triangular. A column is assigned to "pivot" on a particular row if the column contains a suitably large element in a row that has not yet been assigned. (The pivot elements ultimately form the diagonals of the triangular basis.) For remaining unassigned rows, slack variables are inserted to complete the basis.

Default: `Crash option 3` for linearly constrained problems and `Crash option 0;` for problems with nonlinear constraints.

Crash tolerance r

The `Crash tolerance` r allows the starting procedure CRASH to ignore certain "small" nonzeros in each column of A. If $a_{\max}$ is the largest element in column j, other nonzeros a_{ij} in the column are ignored if $|a_{ij}| \le a_{\max} \times r$. (To be meaningful, r should be in the range $0 \le r < 1$.)

When $r > 0.0$, the basis obtained by CRASH may not be strictly triangular, but it is likely to be nonsingular and almost triangular. The intention is to obtain a starting basis containing more columns of A and fewer (arbitrary) slacks. A feasible solution may be reached sooner on some problems.

For example, suppose the first m columns of A are the matrix shown under `LU factor tolerance`; i.e., a tridiagonal matrix with entries -1, 4, -1. To help CRASH choose all m columns for the initial basis, we would specify `Crash tolerance` r for some value of $r > 1/4$.

Default: `Crash tolerance 0.1`

Elastic weight ω

This parameter denoted by ω determines the initial weight γ associated with problem NP(γ).

At any given major iteration k, elastic mode is started if the QP subproblem is infeasible, or the QP dual variables are larger in magnitude than $\omega(1 + \|g(x_k)\|_2)$, where g is the objective gradient. In either case, the QP is re-solved in elastic mode with $\gamma = \omega(1 + \|g(x_k)\|_2)$.

Thereafter, γ is increased (subject to a maximum allowable value) at any point that is optimal for problem NP(γ), but not feasible for NP. After the rth increase, $\gamma = \omega 10^r (1 + \|g(x_{k1})\|_2)$, where x_{k1} is the iterate at which γ was first needed.

Default: `Elastic weight 10000.0`

Expand frequency i

This option is part of the EXPAND anti-cycling procedure [8] designed to make progress even on highly degenerate problems.

For linear models, the strategy is to force a positive step at every iteration, at the expense of violating the bounds on the variables by a small amount. Suppose that the `Minor feasibility tolerance` is δ. Over a period of i iterations, the tolerance actually used by SNOPT increases from 0.5δ to δ (in steps of $0.5\delta/i$).

For nonlinear models, the same procedure is used for iterations in which there is only one superbasic variable. (Cycling can occur only when the current solution is at a vertex of the feasible region.) Thus, zero steps are allowed if there is more than one superbasic variable, but otherwise positive steps are enforced.

Increasing the expand frequency helps reduce the number of slightly infeasible nonbasic basic variables (most of which are eliminated during a resetting procedure). However, it also diminishes the freedom to choose a large pivot element (see `Pivot tolerance`).

Default: `Expand frequency 10000`

Factorization frequency k

At most k basis changes will occur between factorizations of the basis matrix.

- With linear programs, the basis factors are usually updated every iteration. The default k is reasonable for typical problems. Smaller values (say $k = 75$ or $k = 50$) may be more efficient on problems that are rather dense or poorly scaled.
- When the problem is nonlinear, fewer basis updates will occur as an optimum is approached. The number of iterations between basis factorizations will therefore increase. During these iterations a test is made regularly (according to the `Check frequency`) to ensure that the general constraints are satisfied. If necessary the basis will be refactorized before the limit of k updates is reached.

Default: `Factorization frequency 100` for linear programs and `Factorization frequency 50` for nonlinear models.

Feasibility tolerance t

See `Minor feasibility tolerance`.
Default: `Feasibility tolerance 1.0e-6`

Feasible point only

This options means "Ignore the objective function" while finding a feasible point for the linear and nonlinear constraints. It can be used to check that the nonlinear constraints are feasible.
Default: turned off.

Feasible exit

If SNOPT is about to terminate with nonlinear constraint violations, the option `Feasible exit` requests further effort to satisfy the nonlinear constraints while ignoring the objective function.
Default: turned off.

Hessian Full memory

This option selects the full storage method for storing and updating the approximate Hessian. (SNOPT uses a quasi-Newton approximation to the Hessian of the Lagrangian. A BFGS update is applied after each major iteration.)
If `Hessian Full memory` is specified, the approximate Hessian is treated as a dense matrix and the BFGS updates are applied explicitly. This option is most efficient when the number of nonlinear variables n_1 is not too large (say, less than 75). In this case, the storage requirement is fixed and one can expect 1-step Q-superlinear convergence to the solution.
Default: turned on when the number of nonlinear variables $n_1 \leq 75$.

Hessian Limited memory

This option selects the limited memory storage method for storing and updating the approximate Hessian. (SNOPT uses a quasi-Newton approximation to the Hessian of the Lagrangian. A BFGS update is applied after each major iteration.)

`Hessian Limited memory` should be used on problems where the number of nonlinear variables n_1 is very large. In this case a limited-memory procedure is used to update a diagonal Hessian approximation H_r a limited number of times. (Updates are accumulated as a list of vector pairs. They are discarded at regular intervals after H_r has been reset to their diagonal.)

Default: turned on when the number of nonlinear variables $n_1 > 75$.

Hessian frequency i

If `Hessian Full` is selected and i BFGS updates have already been carried out, the Hessian approximation is reset to the identity matrix. (For certain problems, occasional resets may improve convergence, but in general they should not be necessary.)

`Hessian Full memory` and `Hessian frequency = 20` have a similar effect to `Hessian Limited memory` and `Hessian updates = 20` (except that the latter retains the current diagonal during resets).

Default: `Hessian frequency 99999999` (i.e. never).

Hessian updates i

If `Hessian Limited memory` is selected and i BFGS updates have already been carried out, all but the diagonal elements of the accumulated updates are discarded and the updating process starts again.

Broadly speaking, the more updates stored, the better the quality of the approximate Hessian. However, the more vectors stored, the greater the cost of each QP iteration. The default value is likely to give a robust algorithm without significant expense, but faster convergence can sometimes be obtained with significantly fewer updates (e.g., $i = 5$).

Default: `Hessian updates 20` (only when limited memory storage model is used).

Infeasible exit ok

If SNOPT is about to terminate with nonlinear constraint violations, the companion option `Feasible exit` requests further effort to satisfy the nonlinear constraints while ignoring the objective function. `Infeasible exit ok` does not do this extra effort.

Default: turned on.

Iterations limit k

This is the maximum number of minor iterations allowed (i.e., iterations of the simplex method or the QP algorithm), summed over all major iterations. This option overrides the GAMS iterlim options.

Default: specified by GAMS.

Linesearch tolerance t

This controls the accuracy with which a steplength will be located along the direction of search each iteration. At the start of each linesearch a target directional derivative for the merit function is identified. This parameter determines the accuracy to which this target value is approximated.

- t must be a real value in the range $0.0 \leq t \leq 1.0$.
- The default value $t = 0.9$ requests just moderate accuracy in the linesearch.
- If the nonlinear functions are cheap to evaluate (this is usually the case for GAMS models), a more accurate search may be appropriate; try $t = 0.1$, 0.01 or 0.001. The number of major iterations might decrease.
- If the nonlinear functions are expensive to evaluate, a less accurate search may be appropriate. In the case of running under GAMS where all gradients are known, try $t = 0.99$. (The number of major iterations might increase, but the total number of function evaluations may decrease enough to compensate.)

Default: `Linesearch tolerance 0.9`.

Log frequency k
See `Print frequency`.
Default: `Log frequency 1`

LU factor tolerance r_1
LU update tolerance r_2
These tolerances affect the stability and sparsity of the basis factorization $B = LU$ during refactorization and updating, respectively. They must satisfy r_1, $r_2 \geq 1.0$. The matrix L is a product of matrices of the form

$$\begin{pmatrix} 1 & \\ \mu & 1 \end{pmatrix},$$

where the multipliers μ satisfy $|\mu| \leq r_i$. Smaller values of r_i favor stability, while larger values favor sparsity.

- For large and relatively dense problems, $r_1 = 5.0$ (say) may give a useful improvement in stability without impairing sparsity to a serious degree.
- For certain very regular structures (e.g., band matrices) it may be necessary to reduce r_1 and/or r_2 in order to achieve stability. For

example, if the columns of A include a submatrix of the form

$$\begin{pmatrix} 4 & -1 & & & & \\ -1 & 4 & -1 & & & \\ & -1 & 4 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 4 & -1 \\ & & & & -1 & 4 \end{pmatrix},$$

both r_1 and r_2 should be in the range $1.0 \le r_i < 4.0$.

Defaults for linear models: `LU factor tolerance 100.0` and `LU update tolerance 10.0`.
The defaults for nonlinear models are `LU factor tolerance 5.0` and `LU update tolerance 5.0`.

LU density tolerance r_1

The density tolerance r_1 is used during LU factorization of the basis matrix. Columns of L and rows of U are formed one at a time, and the remaining rows and columns of the basis are altered appropriately. At any stage, if the density of the remaining matrix exceeds r_1, the Markowitz strategy for choosing pivots is terminated. The remaining matrix is factored by a dense LU procedure. Raising the density tolerance towards 1.0 may give slightly sparser LU factors, with a slight increase in factorization time.
Default: `LU density tolerance 0.6`

LU singularity tolerance r_2

The singularity tolerance r_2 helps guard against ill-conditioned basis matrices. When the basis is refactorized, the diagonal elements of U are tested as follows: if $|U_{jj}| \le r_2$ or $|U_{jj}| < r_2 \max_i |U_{ij}|$, the jth column of the basis is replaced by the corresponding slack variable. (This is most likely to occur after a restart, or at the start of a major iteration.)
In some cases, the Jacobian may converge to values that make the basis exactly singular. (For example, a whole row of the Jacobian could be zero at an optimal solution.) Before exact singularity occurs, the basis could become very ill-conditioned and the optimization could progress very slowly (if at all). Setting a larger tolerance r_2 = `1.0e-5`, say, may help cause a judicious change of basis.
Default: `LU singularity tolerance 3.7e-11` for most machines. This value corresponds to $\epsilon^{2/3}$, where ϵ is the relative machine precision.

Major feasibility tolerance ϵ_r

This specifies how accurately the nonlinear constraints should be satisfied. The default value of `1.0e-6` is appropriate when the linear and nonlinear constraints contain data to about that accuracy.

Let `rowerr` be the maximum nonlinear constraint violation, normalized by the size of the solution. It is required to satisfy

$$\texttt{rowerr} = \max_i \ \texttt{viol}_i/\|x\| \ \le \ \epsilon_r, \tag{5}$$

where $\texttt{viol}_i$ is the violation of the ith nonlinear constraint ($i = 1 : \texttt{nnCon}$, `nnCon` being the number of nonlinear constraints).
In the GAMS/SNOPT iteration log, `rowerr` appears as the quantity labeled "`Feasibl`". If some of the problem functions are known to be of low accuracy, a larger `Major feasibility tolerance` may be appropriate.
Default: `Major feasibility tolerance 1.0e-6`.

Major optimality tolerance ϵ_d
This specifies the final accuracy of the dual variables. On successful termination, SNOPT will have computed a solution (x, s, π) such that

$$\texttt{maxgap} = \max_j \ \texttt{gap}_j/\|\pi\| \ \le \ \epsilon_d, \tag{6}$$

where $\texttt{gap}_j$ is an estimate of the complementarity gap for variable j ($j = 1 : n + m$). The gaps are computed from the final QP solution using the reduced gradients $d_j = g_j - \pi^T a_j$ (where g_j is the jth component of the objective gradient, a_j is the associated column of the constraint matrix $(\,A \quad -I\,)$, and π is the set of QP dual variables):

$$\texttt{gap}_j = \begin{cases} d_j \min\{x_j - l_j, 1\} & \text{if } d_j \ge 0; \\ -d_j \min\{u_j - x_j, 1\} & \text{if } d_j < 0. \end{cases}$$

In the GAMS/SNOPT iteration log, `maxgap` appears as the quantity labeled "`Optimal`".
Default: `Major optimality tolerance 1.0e-6`.

Major iterations limit k
This is the maximum number of major iterations allowed. It is intended to guard against an excessive number of linearizations of the constraints.
Default: `Major iterations limit` $\max\{1000, 3\max\{m, n\}\}$.

Major print level p
This controls the amount of output to the GAMS listing file each major iteration. This output is only visible if the `sysout` option is turned on (see §4.1). `Major print level 1` gives normal output for linear and nonlinear problems, and `Major print level 11` gives additional details of the Jacobian factorization that commences each major iteration.
In general, the value being specified may be thought of as a binary number of the form

```
Major print level JFDXbs
```

where each letter stands for a digit that is either `0` or `1` as follows:

`s` a single line that gives a summary of each major iteration. (This entry in `JFDXbs` is not strictly binary since the summary line is printed whenever `JFDXbs` ≥ 1).

`b` BASIS statistics, i.e., information relating to the basis matrix whenever it is refactorized. (This output is always provided if `JFDXbs` $\geq$ 10).

`X` x_k, the nonlinear variables involved in the objective function or the constraints.

`D` π_k, the dual variables for the nonlinear constraints.

`F` $F(x_k)$, the values of the nonlinear constraint functions.

`J` $J(x_k)$, the Jacobian.

To obtain output of any items `JFDXbs`, set the corresponding digit to `1`, otherwise to `0`.
If `J=1`, the Jacobian will be output column-wise at the start of each major iteration. Column j will be preceded by the value of the corresponding variable x_j and a key to indicate whether the variable is basic, superbasic or nonbasic. (Hence if `J=1`, there is no reason to specify `X=1` unless the objective contains more nonlinear variables than the Jacobian.) A typical line of output is

```
    3  1.250000D+01 BS      1  1.00000E+00      4  2.00000E+00
```

which would mean that x_3 is basic at value 12.5, and the third column of the Jacobian has elements of 1.0 and 2.0 in rows 1 and 4.
`Major print level 0` suppresses most output, except for error messages.
Default: `Major print level 00001`

Major step limit r

This parameter limits the change in x during a linesearch. It applies to all nonlinear problems, once a "feasible solution" or "feasible subproblem" has been found.

1. A linesearch determines a step α over the range $0 < \alpha \leq \beta$, where β is 1 if there are nonlinear constraints, or the step to the nearest upper or lower bound on x if all the constraints are linear. Normally, the first steplength tried is $\alpha_1 = \min(1, \beta)$.
2. In some cases, such as $f(x) = ae^{bx}$ or $f(x) = ax^b$, even a moderate change in the components of x can lead to floating-point overflow. The parameter r is therefore used to define a limit $\bar{\beta} = r(1+\|x\|)/\|p\|$ (where p is the search direction), and the first evaluation of $f(x)$ is at the potentially smaller steplength $\alpha_1 = \min(1, \bar{\beta}, \beta)$.
3. Wherever possible, upper and lower bounds on x should be used to prevent evaluation of nonlinear functions at meaningless points. The `Major step limit` provides an additional safeguard. The default

value $r = 2.0$ should not affect progress on well behaved problems, but setting $r = 0.1$ or 0.01 may be helpful when rapidly varying functions are present. A "good" starting point may be required. An important application is to the class of nonlinear least-squares problems.

4. In cases where several local optima exist, specifying a small value for r may help locate an optimum near the starting point.

Default: `Major step limit 2.0`.

Minor iterations limit k

This is the maximum number of minor iterations allowed for each QP subproblem in the SQP algorithm. Current experience is that the major iterations converge more reliably if the QP subproblems are allowed to solve accurately. Thus, k should be a large value.
In the major iteration log, a `t` at the end of a line indicates that the corresponding QP was terminated by the limit k.
Note that the SNOPT option `Iterations limit` or the GAMS `iterlim` option defines an independent limit on the *total* number of minor iterations (summed over all QP subproblems).
Default: `Minor iterations limit` $\max\{1000, 5\max\{m, n\}\}$

Minor feasibility tolerance t

SNOPT tries to ensure that all variables eventually satisfy their upper and lower bounds to within the tolerance t. This includes slack variables. Hence, general linear constraints should also be satisfied to within t.
Feasibility with respect to nonlinear constraints is judged by the `Major feasibility tolerance` (not by t).

- If the bounds and linear constraints cannot be satisfied to within t, the problem is declared *infeasible.* Let `sInf` be the corresponding sum of infeasibilities. If `sInf` is quite small, it may be appropriate to raise t by a factor of 10 or 100. Otherwise, some error in the data should be suspected.
- Nonlinear functions will be evaluated only at points that satisfy the bounds and linear constraints. If there are regions where a function is undefined, every attempt should be made to eliminate these regions from the problem. For example, if $f(x) = \sqrt{x_1} + \log x_2$, it is essential to place lower bounds on both variables. If $t = 1.0\text{e-}6$, the bounds $x_1 \geq 10^{-5}$ and $x_2 \geq 10^{-4}$ might be appropriate. (The log singularity is more serious. In general, keep x as far away from singularities as possible.)
- If `Scale option` ≥ 1, feasibility is defined in terms of the *scaled* problem (since it is then more likely to be meaningful).
- In reality, SNOPT uses t as a feasibility tolerance for satisfying the bounds on x and s in each QP subproblem. If the sum of infeasibilities

cannot be reduced to zero, the QP subproblem is declared infeasible. SNOPT is then in *elastic mode* thereafter (with only the linearized nonlinear constraints defined to be elastic). See the `Elastic` options.

Default: `Minor feasilibility tolerance 1.0e-6`.

Minor optimality tolerance t

This is used to judge optimality for each QP subproblem. Let the QP reduced gradients be $d_j = g_j - \pi^T a_j$, where g_j is the jth component of the QP gradient, a_j is the associated column of the QP constraint matrix, and π is the set of QP dual variables.

- By construction, the reduced gradients for basic variables are always zero. The QP subproblem will be declared optimal if the reduced gradients for nonbasic variables at their lower or upper bounds satisfy

$$d_j/\|\pi\| \geq -t \quad \text{or} \quad d_j/\|\pi\| \leq t$$

respectively, and if $|d_j|/\|\pi\| \leq t$ for superbasic variables.

- In the above tests, $\|\pi\|$ is a measure of the size of the dual variables. It is included to make the tests independent of a large scale factor on the objective function. The quantity actually used is defined by

$$\|\pi\| = \max\{\sigma/\sqrt{m}, 1\}, \quad \text{where} \quad \sigma = \sum_{i=1}^{m} |\pi_i|.$$

- If the objective is scaled down to be very *small*, the optimality test reduces to comparing d_j against t.

Default: `Minor optimality tolerance 1.0e-6`.

Minor print level k

This controls the amount of output to the GAMS listing file during solution of the QP subproblems. This option is only useful if the `sysout` option is turned on (see §4.1). The value of k has the following effect:

0	No minor iteration output except error messages.
≥ 1	A single line of output each minor iteration (controlled by `Print frequency`).
≥ 10	Basis factorization statistics generated during the periodic refactorization of the basis (see `Factorization frequency`). Statistics for the *first factorization* each major iteration are controlled by the `Major print level`.

Default: `Minor print level 0`.

Optimality tolerance t

See `Minor optimality tolerance`.
Default: `Optimality tolerance 1.0e-6`.

Partial Price i

This parameter is recommended for large problems that have significantly more variables than constraints. It reduces the work required for each "pricing" operation (when a nonbasic variable is selected to become superbasic).

- When $i = 1$, all columns of the constraint matrix $(\, A \quad -I \,)$ are searched.
- Otherwise, A and I are partitioned to give i roughly equal segments A_j, I_j ($j = 1$ to i). If the previous pricing search was successful on A_j, I_j, the next search begins on the segments A_{j+1}, I_{j+1}. (All subscripts here are modulo i.)
- If a reduced gradient is found that is larger than some dynamic tolerance, the variable with the largest such reduced gradient (of appropriate sign) is selected to become superbasic. If nothing is found, the search continues on the next segments A_{j+2}, I_{j+2}, and so on.
- `Partial price` t (or $t/2$ or $t/3$) may be appropriate for time-stage models having t time periods.

Default: `Partial price 10` for linear models and `Partial price 1` for nonlinear models.

Pivot tolerance r During solution of QP subproblems, the pivot tolerance is used to prevent columns entering the basis if they would cause the basis to become almost singular.

- When x changes to $x + \alpha p$ for some search direction p, a "ratio test" is used to determine which component of x reaches an upper or lower bound first. The corresponding element of p is called the pivot element.
- Elements of p are ignored (and therefore cannot be pivot elements) if they are smaller than the pivot tolerance r.
- It is common for two or more variables to reach a bound at essentially the same time. In such cases, the `Minor Feasibility tolerance` (say t) provides some freedom to maximize the pivot element and thereby improve numerical stability. Excessively small values of t should therefore not be specified.
- To a lesser extent, the `Expand frequency` (say f) also provides some freedom to maximize the pivot element. Excessively *large* values of f should therefore not be specified.

Default: `Pivot tolerance 3.7e-11` on most machines. This corresponds to $\epsilon^{2/3}$ where ϵ is the machine precision.

Print frequency k

When `sysout` is turned on (see §4.1) and `Minor print level` ≥ 1, a line

of the QP iteration log will be printed on the listing file every kth minor iteration.
Default: `Print frequency 1`.

Scale option i
Three scale options are available as follows:

`Scale option 0`: No scaling. This is recommended if it is known that x and the constraint matrix (and Jacobian) never have very large elements (say, larger than 1000).

`Scale option 1`: Linear constraints and variables are scaled by an iterative procedure that attempts to make the matrix coefficients as close as possible to 1.0 (see Fourer [5]). This will sometimes improve the performance of the solution procedures.

`Scale option 2`: All constraints and variables are scaled by the iterative procedure. Also, an additional scaling is performed that takes into account columns of $(A \quad -I)$ that are fixed or have positive lower bounds or negative upper bounds.

If nonlinear constraints are present, the scales depend on the Jacobian at the first point that satisfies the linear constraints. `Scale option 2` should therefore be used only if (a) a good starting point is provided, and (b) the problem is not highly nonlinear.

Default: `Scale option 2` for linear models and `Scale option 1` for NLP's.

Scale tolerance r
This parameter affects how many passes might be needed through the constraint matrix. On each pass, the scaling procedure computes the ratio of the largest and smallest nonzero coefficients in each column:

$$\rho_j = \max_i |a_{ij}| / \min_i |a_{ij}| \qquad (a_{ij} \neq 0).$$

If $\max_j \rho_j$ is less than r times its previous value, another scaling pass is performed to adjust the row and column scales. Raising r from 0.9 to 0.99 (say) usually increases the number of scaling passes through A. At most 10 passes are made.
Default: `Scale tolerance 0.9`.

Scale Print
This option causes the row-scales $r(i)$ and column-scales $c(j)$ to be printed. The scaled matrix coefficients are $\bar{a}_{ij} = a_{ij}c(j)/r(i)$, and the scaled bounds on the variables and slacks are $\bar{l}_j = l_j/c(j)$, $\bar{u}_j = u_j/c(j)$, where $c(j) \equiv r(j-n)$ if $j > n$.
The listing file will only show these values if the `sysout` option is turned on (see §4.1).
Default: turned off.

Solution Yes

This option causes the SNOPT solution file to be printed to the GAMS listing file. It is only visible if the `sysout` option is turned on (see §4.1).
Default: turned off.

Superbasics limit i

This places a limit on the storage allocated for superbasic variables. Ideally, i should be set slightly larger than the "number of degrees of freedom" expected at an optimal solution.
For linear programs, an optimum is normally a basic solution with no degrees of freedom. (The number of variables lying strictly between their bounds is no more than m, the number of general constraints.) The default value of i is therefore 1.
For nonlinear problems, the number of degrees of freedom is often called the "number of independent variables".
Normally, i need not be greater than $n_1 + 1$, where n_1 is the number of nonlinear variables. For many problems, i may be considerably smaller than n_1. This will save storage if n_1 is very large.

Unbounded objective value $f_{\max}$
Unbounded step size $\alpha_{\max}$

These parameters are intended to detect unboundedness in nonlinear problems. (They may not achieve that purpose!) During a line search, f is evaluated at points of the form $x + \alpha p$, where x and p are fixed and α varies. if $|f|$ exceeds $f_{\max}$ or α exceeds $\alpha_{\max}$, iterations are terminated with the exit message `Problem is unbounded (or badly scaled)`.
In a GAMS environment no floating-point overflow errors should occur when singularities are present during the evaluation of $f(x + \alpha p)$ before the test can be made.
Defaults: `Unbounded objective value 1.0e+15` and `Unbounded step size 1.0e+18`.

Violation limit τ

This keyword defines an absolute limit on the magnitude of the maximum constraint violation after the line search. On completion of the line search, the new iterate x_{k+1} satisfies the condition

$$v_i(x_{k+1}) \leq \tau \max\{1, v_i(x_0)\},$$

where x_0 is the point at which the nonlinear constraints are first evaluated and $v_i(x)$ is the ith nonlinear constraint violation $v_i(x) = \max(0, l_i - F_i(x), F_i(x) - u_i)$.
The effect of this violation limit is to restrict the iterates to lie in an *expanded* feasible region whose size depends on the magnitude of τ. This makes it possible to keep the iterates within a region where the objective is expected to be well-defined and bounded below. If the objective is bounded below for all values of the variables, then τ may be any large

positive value.
Default: `Violation limit 10`.

5 The SNOPT log

When GAMS/SNOPT solves a linearly constrained problem the following log is visible on the screen:

```
[erwin@hamilton]$ gamslib chem
Model chem.gms retrieved
[erwin@hamilton]$ gams chem nlp=snopt
GAMS 2.50A    Copyright (C) 1987-1999 GAMS Development. All rights reserved
Licensee: hamilton                                       G990622:1048CP-LNX
          GAMS Development
--- Starting compilation
--- chem.gms(47) 1 Mb
--- Starting execution
--- chem.gms(42) 1 Mb
--- Generating model MIXER
--- chem.gms(46) 1 Mb
---    5 rows, 12 columns, and 37 non-zeroes.
--- Executing SNOPT

    GAMS/SNOPT    X86/LINUX  version 5.3.4-007-035
    P. E. Gill, UC San Diego
    W. Murray and M. A. Saunders, Stanford University

 Work space allocated           --    0.02 Mb

 Major Minor   Step   nObj  Objective   Optimal    nS PD
     0     5 0.0E+00     1  3.292476E+01 2.1E-01    0 TF   r
     1     4 1.0E+00     2  4.517191E+01 2.2E-01    1 TF n r
     2    10 8.6E-02     5  4.533775E+01 1.7E-01    6 TF s
     3     3 1.0E+00     6  4.608439E+01 7.6E-02    6 TF
     4     2 1.0E+00     7  4.667813E+01 1.4E-01    5 TF
     5     2 1.0E+00     8  4.751149E+01 2.2E-02    6 TF
     6     3 1.0E+00     9  4.757024E+01 2.1E-02    4 TF
     7     2 7.1E-01    11  4.763634E+01 3.3E-02    5 TF
     8     3 1.0E+00    12  4.768395E+01 3.3E-02    7 TF
     9     3 4.6E-01    14  4.769958E+01 1.9E-02    5 TF
    10     2 1.0E+00    15  4.770539E+01 1.5E-02    6 TF

 Major Minor   Step   nObj  Objective   Optimal    nS PD
    11     1 1.0E+00    16  4.770639E+01 6.2E-04    6 TF
    12     1 1.0E+00    17  4.770650E+01 5.0E-03    6 TF
    13     1 1.0E+00    18  4.770651E+01 1.6E-04    6 TF
    14     1 1.0E+00    19  4.770651E+01 1.8E-05    6 TF
    15     1 1.0E+00    20  4.770651E+01 2.7E-05    6 TF
    16     1 1.0E+00    21  4.770651E+01 7.6E-07    6 TT

 EXIT - Optimal Solution found.

--- Restarting execution
--- chem.gms(46) 1 Mb
--- Reading solution for model MIXER
--- chem.gms(46) 1 Mb
*** Status: Normal completion
[erwin@hamilton]$
```

For a nonlinearly constrained problem, the log is somewhat different:

```
[erwin@hamilton]$ gamslib chenery
Model chenery.gms retrieved
```

```
[erwin@hamilton]$ gams chenery nlp=snopt
GAMS 2.50A    Copyright (C) 1987-1999 GAMS Development. All rights reserved
Licensee: hamilton                                      G990622:1048CP-LNX
          GAMS Development
--- Starting compilation
--- chenery.gms(240) 1 Mb
--- Starting execution
--- chenery.gms(222) 1 Mb
--- Generating model CHENRAD
--- chenery.gms(225) 1 Mb
---    39 rows, 44 columns, and 133 non-zeroes.
--- Executing SNOPT

    GAMS/SNOPT    X86/LINUX  version 5.3.4-007-035
    P. E. Gill, UC San Diego
    W. Murray and M. A. Saunders, Stanford University

 Work space allocated           --    0.07 Mb

 Major Minor   Step   nCon    Merit      Feasibl Optimal   nS Penalty PD
     0    39 0.0E+00     1 -1.653933E+07 1.4E+00 1.4E+00    6 0.0E+00 FF   r i
     1     6 1.0E+00     2  6.215366E+03 9.8E-01 1.4E+00    6 0.0E+00 FF n r
     2     2 1.0E+00     3 -4.424844E+03 5.6E-01 7.8E-01    7 2.8E-01 FF s
     3     1 1.0E+00     4  2.756620E+02 1.1E-01 2.1E-01    7 2.8E-01 FF
     4     1 1.0E+00     6  5.640617E+02 5.6E-03 2.2E-01    7 2.8E-01 FF  m
     5     6 1.0E+00     8  6.188177E+02 9.9E-03 2.6E-01    5 2.8E-01 FF  m
     6     2 7.2E-01    11  6.827737E+02 2.7E-02 2.0E-01    4 2.8E-01 FF  m
     7     4 5.9E-01    14  7.516259E+02 3.4E-02 9.4E-02    6 2.8E-01 FF  m
     8     1 1.0E+00    15  8.437315E+02 6.7E-03 1.7E+00    6 2.8E-01 FF
     9     3 1.0E+00    17  8.756771E+02 7.1E-03 3.5E-01    4 2.8E-01 FF  m
    10     5 3.1E-01    21  9.010440E+02 2.4E-02 1.1E+00    6 2.8E-01 FF  m

 Major Minor   Step   nCon    Merit      Feasibl Optimal   nS Penalty PD
    11     2 2.6E-01    24  9.168958E+02 2.5E-02 6.9E-01    5 2.8E-01 FF
    12     1 4.8E-01    26  9.404851E+02 2.9E-02 5.0E-01    5 2.8E-01 FF
    13     3 1.0E+00    27  9.983802E+02 1.6E-02 1.3E-01    6 2.8E-01 FF
    14     1 1.0E+00    28  1.013533E+03 6.2E-04 4.8E-02    6 2.8E-01 FF
    15     2 1.0E+00    29  1.021295E+03 1.1E-02 1.2E-02    5 2.8E-01 FF
    16     1 1.0E+00    30  1.032156E+03 5.2E-03 1.1E-02    5 2.8E-01 FF
    17     2 1.0E+00    31  1.033938E+03 6.7E-05 1.4E-02    4 2.8E-01 FF
    18     2 1.0E+00    32  1.036764E+03 4.5E-04 1.0E-02    3 2.8E-01 FF
    19     2 1.0E+00    33  1.037592E+03 6.5E-05 3.0E-02    2 2.8E-01 FF
    20     1 1.0E+00    34  1.039922E+03 4.6E-04 4.4E-02    2 2.8E-01 FF
    21     2 1.0E+00    35  1.040566E+03 1.4E-05 7.8E-02    3 2.8E-01 FF

 Major Minor   Step   nCon    Merit      Feasibl Optimal   nS Penalty PD
    22     4 1.0E+00    36  1.056256E+03 1.3E-02 5.6E-02    4 2.8E-01 FF
    23     2 1.0E+00    37  1.053213E+03 1.9E-04 3.1E-03    3 3.9E-01 FF
    24     2 1.0E+00    38  1.053464E+03 2.2E-05 4.7E-03    2 3.9E-01 FF
    25     1 1.0E+00    39  1.053811E+03 3.7E-05 1.8E-02    2 3.9E-01 FF
    26     1 1.0E+00    40  1.055352E+03 1.1E-03 3.9E-02    2 3.9E-01 FF
    27     1 1.0E+00    41  1.055950E+03 1.3E-03 1.5E-02    2 3.9E-01 FF
    28     1 1.0E+00    42  1.056047E+03 1.5E-06 1.3E-02    2 3.9E-01 FF
    29     1 1.0E+00    43  1.056991E+03 3.2E-04 2.2E-02    2 3.9E-01 FF
    30     1 1.0E+00    44  1.058439E+03 2.7E-03 1.8E-02    2 3.9E-01 FF
    31     3 1.0E+00    45  1.058885E+03 2.2E-04 9.3E-03    2 3.9E-01 FF
    32     1 1.0E+00    46  1.058918E+03 3.3E-05 1.6E-03    2 3.9E-01 FF

 Major Minor   Step   nCon    Merit      Feasibl Optimal   nS Penalty PD
    33     1 1.0E+00    47  1.058920E+03 4.6E-06 1.1E-05    2 3.9E-01 FF
    34     1 1.0E+00    48  1.058920E+03 5.1E-10 5.3E-07    2 3.9E-01 TT

 EXIT - Optimal Solution found.

--- Restarting execution
--- chenery.gms(225) 1 Mb
--- Reading solution for model CHENRAD
--- chenery.gms(239) 1 Mb
```

```
*** Status: Normal completion
[erwin@hamilton]$
```

GAMS prints the number of equations, variables and non-zero elements of the model it generated. This gives an indication of the size of the model. SNOPT then says how much memory it allocated to solve the model, based on an estimate. If the user had specified a different amount using the `work` option or the `workspace` model suffix, there would be a message like

```
   Work space requested by user   --    0.76 Mb
   Work space requested by solver --    0.02 Mb
```

The SNOPT log shows the following columns:

Major The current major iteration number.

Minor is the number of iterations required by both the feasibility and optimality phases of the QP subproblem. Generally, `Minor` will be 1 in the later iterations, since theoretical analysis predicts that the correct active set will be identified near the solution (see §2).

Step The step length α taken along the current search direction p. The variables x have just been changed to $x + \alpha p$. On reasonably well-behaved problems, the unit step will be taken as the solution is approached.

nObj The number of times the nonlinear objective function has been evaluated. `nObj` is printed as a guide to the amount of work required for the linesearch.

nCon The number of times SNOPT evaluated the nonlinear constraint functions.

Merit is the value of the augmented Lagrangian merit function (3). This function will decrease at each iteration unless it was necessary to increase the penalty parameters (see §2). As the solution is approached, `Merit` will converge to the value of the objective at the solution.
In elastic mode, the merit function is a composite function involving the constraint violations weighted by the elastic weight.
If the constraints are linear, this item is labeled `Objective`, the value of the objective function. It will decrease monotonically to its optimal value.

Feasibl is the value of `rowerr`, the maximum component of the scaled nonlinear constraint residual (5). The solution is regarded as acceptably feasible if `Feasibl` is less than the `Major feasibility tolerance`.
If the constraints are linear, all iterates are feasible and this entry is not printed.

Optimal is the value of `maxgap`, the maximum complementarity gap (6). It is an estimate of the degree of nonoptimality of the reduced costs. Both `Feasibl` and `Optimal` are small in the neighborhood of a solution.

nS The current number of superbasic variables.

Penalty is the Euclidean norm of the vector of penalty parameters used in the augmented Lagrangian merit function (not printed if the constraints are linear).

PD is a two-letter indication of the status of the convergence tests involving primal and dual feasibility of the iterates (see (5) and (6) in the description of `Major feasibility tolerance` and `Major optimality tolerance`). Each letter is `T` if the test is satisfied, and `F` otherwise.
If either of the indicators is `F` when SNOPT terminates with `0 EXIT -- optimal solution found`, the user should check the solution carefully.

The summary line may include additional code characters that indicate what happened during the course of the iteration.

`c` Central differences have been used to compute the unknown components of the objective and constraint gradients. This should not happen in a GAMS environment.

`d` During the linesearch it was necessary to decrease the step in order to obtain a maximum constraint violation conforming to the value of `Violation limit`.

`l` The norm-wise change in the variables was limited by the value of the `Major step limit`. If this output occurs repeatedly during later iterations, it may be worthwhile increasing the value of `Major step limit`.

`i` If SNOPT is not in elastic mode, an "`i`" signifies that the QP subproblem is infeasible. This event triggers the start of nonlinear elastic mode, which remains in effect for all subsequent iterations. Once in elastic mode, the QP subproblems are associated with the elastic problem NP(γ).
If SNOPT is already in elastic mode, an "`i`" indicates that the minimizer of the elastic subproblem does not satisfy the linearized constraints. (In this case, a feasible point for the usual QP subproblem may or may not exist.)

`M` An extra evaluation of the problem functions was needed to define an acceptable positive-definite quasi-Newton update to the Lagrangian Hessian. This modification is only done when there are nonlinear constraints.

`m` This is the same as "`M`" except that it was also necessary to modify the update to include an augmented Lagrangian term.

`R` The approximate Hessian has been reset by discarding all but the diagonal elements. This reset will be forced periodically by the `Hessian frequency` and `Hessian updates` keywords. However, it may also be necessary to reset an ill-conditioned Hessian from time to time.

`r` The approximate Hessian was reset after ten consecutive major iterations in which no BFGS update could be made. The diagonals of the approximate Hessian are retained if at least one update has been done since the last reset. Otherwise, the approximate Hessian is reset to the identity matrix.

`s` A self-scaled BFGS update was performed. This update is always used when the Hessian approximation is diagonal, and hence always follows a Hessian reset.

`S` This is the same as a "`s`" except that it was necessary to modify the self-scaled update to maintain positive definiteness.

`n` No positive-definite BFGS update could be found. The approximate Hessian is unchanged from the previous iteration.

`t` The minor iterations were terminated at the `Minor iteration limit`.

`u` The QP subproblem was unbounded.

`w` A weak solution of the QP subproblem was found.

Finally SNOPT prints an exit message. See §5.1.

5.1 EXIT conditions

When the solution procedure terminates, an `EXIT --` message is printed to summarize the final result. Here we describe each message and suggest possible courses of action.

```
 0  EXIT -- optimal solution found
```

This is the message we all hope to see! It is certainly preferable to every other message, and we naturally want to believe what it says, because this is surely one situation where *the computer knows best*.

In all cases, a distinct level of caution is in order, even if it can wait until next morning. For example, if the objective value is much better than expected, we may have obtained an optimal solution to the wrong problem! Almost any item of data could have that effect if it has the wrong value. Verifying that the problem has been defined correctly is one of the more difficult tasks for a model builder.

If nonlinearities exist, one must always ask the question: could there be more than one local optimum? When the constraints are linear and the objective is known to be convex (e.g., a sum of squares) then all will be well if we are *minimizing* the objective: a local minimum is a global minimum in the sense that no other point has a lower function value. (However, many points could have the *same* objective value, particularly if the objective is largely linear.) Conversely, if we are *maximizing* a convex function, a local maximum cannot be expected to be global, unless there are sufficient constraints to confine the feasible region.

Similar statements could be made about nonlinear constraints defining convex or concave regions. However, the functions of a problem are more likely to be neither convex nor concave. Our advice is always to specify a starting point that is as good an estimate as possible, and to include reasonable upper and lower bounds on all variables, in order to confine the solution to the specific region of interest. We expect modelers to *know something about their problem*, and to make use of that knowledge as they themselves know best.

One other caution about "`Optimal solution`"s. Some of the variables or slacks may lie outside their bounds more than desired, especially if scaling was requested. `Max Primal infeas` refers to the largest bound infeasibility and which variable is involved. If it is too large, consider restarting with a smaller `Minor feasibility tolerance` (say 10 times smaller) and perhaps `Scale option 0`.

Similarly, `Max Dual infeas` indicates which variable is most likely to be at a non-optimal value. Broadly speaking, if

$$\text{Max Dual infeas/Norm of pi} = 10^{-d},$$

then the objective function would probably change in the dth significant digit if optimization could be continued. If d seems too large, consider restarting with smaller `Major` and `Minor optimality tolerance`s.

Finally, `Nonlinear constraint violn` shows the maximum infeasibility for nonlinear rows. If it seems too large, consider restarting with a smaller `Major feasibility tolerance`.

` 1  EXIT -- the problem is infeasible`

When the constraints are linear, this message can probably be trusted. Feasibility is measured with respect to the upper and lower bounds on the variables and slacks. Among all the points satisfying the general constraints $Ax - s = 0$, there is apparently no point that satisfies the bounds on x and s. Violations as small as the `Minor feasibility tolerance` are ignored, but at least one component of x or s violates a bound by more than the tolerance.

When nonlinear constraints are present, infeasibility is *much* harder to recognize correctly. Even if a feasible solution exists, the current linearization of the constraints may not contain a feasible point. In an attempt to deal with this situation, when solving each QP subproblem, SNOPT is prepared to relax the bounds on the slacks associated with nonlinear rows.

If a QP subproblem proves to be infeasible or unbounded (or if the Lagrange multiplier estimates for the nonlinear constraints become large), SNOPT enters so-called "nonlinear elastic" mode. The subproblem includes the original QP objective and the sum of the infeasibilities—suitably weighted using the `Elastic weight` parameter. In elastic mode, the nonlinear rows are made "elastic"—i.e., they are allowed to violate their specified bounds. Variables subject to elastic bounds are known as *elastic variables*. An elastic variable is free to violate one or both of its original upper or lower bounds. If the original problem has a feasible solution and the elastic weight is sufficiently large, a feasible point eventually will be obtained for the perturbed constraints, and optimization can continue on the subproblem. If the nonlinear problem has no feasible solution, SNOPT will tend to determine a "good" infeasible point if the elastic weight is sufficiently large. (If the elastic weight were infinite, SNOPT would locally minimize the nonlinear constraint violations subject to the linear constraints and bounds.)

Unfortunately, even though SNOPT locally minimizes the nonlinear constraint violations, there may still exist other regions in which the nonlinear constraints are satisfied. Wherever possible, nonlinear constraints should be defined in such a way that feasible points are known to exist when the constraints are linearized.

```
 2  EXIT -- the problem is unbounded  (or badly scaled)
    EXIT -- violation limit exceeded -- the problem may be unbounded
```

For linear problems, unboundedness is detected by the simplex method when a nonbasic variable can apparently be increased or decreased by an arbitrary amount without causing a basic variable to violate a bound. A message prior to the EXIT message will give the index of the nonbasic variable. Consider adding an upper or lower bound to the variable. Also, examine the constraints that have nonzeros in the associated column, to see if they have been formulated as intended.

Very rarely, the scaling of the problem could be so poor that numerical error will give an erroneous indication of unboundedness. Consider using the `Scale` option.

For nonlinear problems, SNOPT monitors both the size of the current objective function and the size of the change in the variables at each step. If either of these is very large (as judged by the `Unbounded` parameters—see §4), the problem is terminated and declared UNBOUNDED. To avoid large function values, it may be necessary to impose bounds on some of the variables in order to keep them away from singularities in the nonlinear functions.

The second message indicates an abnormal termination while enforcing the limit on the constraint violations. This exit implies that the objective is not bounded below in the feasible region defined by expanding the bounds by the value of the `Violation limit`.

```
 3  EXIT -- major iteration limit exceeded
    EXIT -- minor iteration limit exceeded
    EXIT -- too many iterations
```

Either the `Iterations limit` or the `Major iterations limit` was exceeded before the required solution could be found. Check the iteration log to be sure that progress was being made. If so, repeat the run with higher limits. If not, consider specifying new initial values for some of the nonlinear variables.

```
 4  EXIT -- requested accuracy could not be achieved
```

A feasible solution has been found, but the requested accuracy in the dual infeasibilities could not be achieved. An abnormal termination has occurred, but SNOPT is within 10^{-2} of satisfying the `Major optimality tolerance`. Check that the `Major optimality tolerance` is not too small.

`5 EXIT -- the superbasics limit is too small: nnn`
The problem appears to be more nonlinear than anticipated. The current set of basic and superbasic variables have been optimized as much as possible and a PRICE operation is necessary to continue, but there are already `nnn` superbasics (and no room for any more).

In general, raise the `Superbasics limit` s by a reasonable amount, bearing in mind the storage needed for the reduced Hessian (about $\frac{1}{2}s^2$ double words).

`6 EXIT -- constraint and objective values could not be calculated`
This exit should not occur in a GAMS environment.

`7 EXIT -- subroutine funobj seems to be giving incorrect gradients`
This exit should not occur in a GAMS environment.

`8 EXIT -- subroutine funcon seems to be giving incorrect gradients`
This exit should not occur in a GAMS environment.

`9 EXIT -- the current point cannot be improved upon`
The algorithm could not find a better solution although optimality was not achieved within the optimality tolerance. Possibly scaling can lead to better function values and derivatives. Raising the `optimality tolerance` will probably make this message go away.

`10 EXIT -- cannot satisfy the general constraints`
An LU factorization of the basis has just been obtained and used to recompute the basic variables x_B, given the present values of the superbasic and nonbasic variables. A step of "iterative refinement" has also been applied to increase the accuracy of x_B. However, a row check has revealed that the resulting solution does not satisfy the current constraints $Ax - s = 0$ sufficiently well.

This probably means that the current basis is very ill-conditioned. Try `Scale option 1` if scaling has not yet been used and there are some linear constraints and variables.

For certain highly structured basis matrices (notably those with band structure), a systematic growth may occur in the factor U. Try setting the `LU factor tolerance` to 2.0 (or possibly even smaller, but not less than 1.0).

`12 EXIT -- terminated from subroutine s1User`
This message appears when the *resource limit* was reached (see §4.1) or when the solver was interrupted by a Ctrl-C keyboard signal.

`20 EXIT -- not enough integer/real storage for the basis factors`
Increase the workspace by using the `work` option or the `workspace` model suffix.

`21 EXIT -- error in basis package`
A preceding message will describe the error in more detail. This error should not happen easily in a GAMS environment.

`22 EXIT -- singular basis after nnn factorization attempts`
This exit is highly unlikely to occur. The first factorization attempt will have found the basis to be structurally or numerically singular. (Some diagonals

of the triangular matrix U were respectively zero or smaller than a certain tolerance.) The associated variables are replaced by slacks and the modified basis is refactorized, but singularity persists. This must mean that the problem is badly scaled, or the `LU factor tolerance` is too much larger than 1.0.

`30 EXIT -- the basis file dimensions do not match this problem`
This exit should not occur in a GAMS environment.

`31 EXIT -- the basis file state vector does not match this problem`
This exit should not occur in a GAMS environment.

`32 EXIT -- system error.  Wrong no. of basic variables:  nnn`
This exit should never happen, and may indicate a configuration problem with your GAMS/SNOPT version.

`42 EXIT -- not enough 8-character storage to start solving the problem`
Increase the workspace by using the `work` option or the `workspace` model suffix.

`43 EXIT -- not enough integer storage to start solving the problem`
Increase the work space by using the `work` option or the `workspace` model suffix.

`44 EXIT -- not enough real storage to start solving the problem`
Increase the work space by using the `work` option or the `workspace` model suffix.

`45 EXIT -- Function evaluation error limit exceeded`
The `function evaluation error limit` was reached. When evaluating a nonlinear function either in the objective or in the constraints, an evaluation error occurred and the allowed number of such errors was exceeded. Either increase the `domlim` option (see §4.1) or preferably add bounds or linear constraints such that these errors cannot happen. The errors are most often caused by declaring x to be a free variable while the model contains functions like $\sqrt{x}$ or $\log(x)$. Overflows in exponentiation x^y are also a common cause for this exit. Inspect the listing file for messages like

```
**** ERROR(S) IN EQUATION PRODF
     2 INSTANCES OF - UNDEFINED LOG OPERATION (RETURNED -0.1E5)
```

The equation name mentioned in this message gives a good indication where to look in the model.

6 Listing file messages

The listing file (`.lst` file) also contains feedback on how the SNOPT solver performed on a particular model. For the `chem.gms` model, the solve summary looks like the following:

```
               S O L V E      S U M M A R Y

     MODEL   MIXER                OBJECTIVE  ENERGY
     TYPE    NLP                  DIRECTION  MINIMIZE
     SOLVER  SNOPT                FROM LINE  46
```

```
**** SOLVER STATUS     1 NORMAL COMPLETION
**** MODEL STATUS      2 LOCALLY OPTIMAL
**** OBJECTIVE VALUE              -47.7065

 RESOURCE USAGE, LIMIT          0.010     1000.000
 ITERATION COUNT, LIMIT        45        10000
 EVALUATION ERRORS              0            0

    GAMS/SNOPT    X86/LINUX  version 5.3.4-007-035
    P. E. Gill, UC San Diego
    W. Murray and M. A. Saunders, Stanford University


 Work space allocated           --    0.02 Mb

 EXIT - Optimal Solution found.

 Major, Minor Iterations    16      45
 Funobj, Funcon calls       22       0
 Superbasics                 6
 Aggregations                0
 Interpreter Usage        0.01   100.0%

 Work space used by solver      --    0.02 Mb
```

The solver completed normally at a local (or possibly global) optimum. A complete list of possible solver status and model status values is in Tables 1 and 2.

The resource usage (time used), iteration count and evaluation errors during nonlinear function and gradient evaluation are all within their limits. These limits can be increased by the option `reslim`, `iterlim` and `domlim` (see §4.1).

The possible `EXIT` messages are listed in §5.1.

The statistics following the `EXIT` message are as follows.

Major, minor iterations. The number of major and minor iterations for this optimization task. Note that the number of minor iterations is the same as reported by `ITERATION COUNT`.

Funobj, Funcon calls. The number of times SNOPT evaluated the objective function $f(x)$ or the constraint functions $F_i(x)$ and their gradients. For a linearly constrained problem the number of `funcon` calls should be zero.

Superbasics. This is number of superbasic variables in the reported solution. See §2.4.

Aggregations. The number of equations removed from the model by the objective function recovery algorithm (see §2.1).

Interpreter usage. This line refers to how much time was spent evaluating functions and gradients. Due to the low resolution of the clock and the small size of this model, it was concluded that 100% of the time was spent inside the routines that do function and gradient evaluations. For larger models these numbers are more accurate.

Model status	**Remarks**
`1 OPTIMAL`	Applies only to linear models.
`2 LOCALLY OPTIMAL`	A local optimum in an NLP was found. It may or may not be a global optimum.
`3 UNBOUNDED`	For LP's this message is reliable. A badly scaled NLP can also cause this message to appear.
`4 INFEASIBLE`	Applies to LP's: the model is infeasible.
`5 LOCALLY INFEASIBLE`	Applies to NLP's: Given the starting point, no feasible solution could be found although feasible points may exist.
`6 INTERMEDIATE INFEASIBLE`	The search was stopped (e.g., because of an iteration or time limit) and the current point violates some constraints or bounds.
`7 INTERMEDIATE NONOPTIMAL`	The search was stopped (e.g., because of an iteration or time limit) and the current point is feasible but violates the optimality conditions.
`8 INTEGER SOLUTION`	Does not apply to SNOPT.
`9 INTERMEDIATE NON-INTEGER`	Does not apply to SNOPT.
`10 INTEGER INFEASIBLE`	Does not apply to SNOPT.
`ERROR UKNOWN`	Check listing file for error messages.
`ERROR NO SOLUTION`	Check listing file for error messages.

Table 1: Model status values

Solver status	**Remarks**
`1 NORMAL COMPLETION`	SNOPT completed the optimization task.
`2 ITERATION INTERRUPT`	Iteration limit was hit. Increase the `iterlim` option (see §4.1).
`3 RESOURCE INTERRUPT`	Time limit was hit. Increase the `reslim` option (see §4.1).
`4 TERMINATED BY SOLVER`	Check the listing file.
`5 EVALUATION ERROR LIMIT`	`domlim` error limit was exceeded. See §4.1.
`6 UNKNOWN ERROR`	Check the listing file for error messages.
`ERROR SETUP FAILURE`	Id.
`ERROR SOLVER FAILURE`	Id.
`ERROR INTERNAL SOLVER FAILURE`	Id.
`ERROR SYSTEM FAILURE`	Id.

Table 2: Solver status values

References

[1] A. R. Conn, *Constrained optimization using a nondifferentiable penalty function*, SIAM J. Numer. Anal., 10 (1973), pp. 760–779.

[2] G. B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey, 1963.

[3] S. K. Eldersveld, *Large-scale sequential quadratic programming algorithms*, PhD thesis, Department of Operations Research, Stanford University, Stanford, CA, 1991.

[4] R. Fletcher, *An ℓ_1 penalty method for nonlinear constraints*, in Numerical Optimization 1984, P. T. Boggs, R. H. Byrd, and R. B. Schnabel, eds., Philadelphia, 1985, SIAM, pp. 26–40.

[5] R. Fourer, *Solving staircase linear programs by the simplex method. 1: Inversion*, Math. Prog., 23 (1982), pp. 274–313.

[6] P. E. Gill, W. Murray, and M. A. Saunders, *SNOPT: An SQP algorithm for large-scale constrained optimization*, Numerical Analysis Report 97-2, Department of Mathematics, University of California, San Diego, La Jolla, CA, 1997.

[7] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright, *User's guide for NPSOL (Version 4.0): a Fortran package for nonlinear programming*, Report SOL 86-2, Department of Operations Research, Stanford University, Stanford, CA, 1986.

[8] ——, *A practical anti-cycling procedure for linearly constrained optimization*, Math. Prog., 45 (1989), pp. 437–474.

[9] ——, *Some theoretical properties of an augmented Lagrangian merit function*, in Advances in Optimization and Parallel Computing, P. M. Pardalos, ed., North Holland, North Holland, 1992, pp. 101–128.

[10] ——, *Sparse matrix methods in optimization*, SIAM J. on Scientific and Statistical Computing, 5 (1984), pp. 562–589.

[11] ——, *Maintaining LU factors of a general sparse matrix*, Linear Algebra and its Applications, 88/89 (1987), pp. 239–270.

[12] B. A. Murtagh and M. A. Saunders, *Large-scale linearly constrained optimization*, Math. Prog., 14 (1978), pp. 41–72.

[13] ——, *A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints*, Math. Prog. Study, 16 (1982), pp. 84–117.

[14] ——, *MINOS 5.5 User's Guide*, Report SOL 83-20R, Department of Operations Research, Stanford University, Stanford, CA, Revised 1998.