User's Guide for SNOPT-MATLAB: Matlab interface for nonlinear optimizer SNOPT

Philip E. Gill* Elizabeth Wong†
Department of Mathematics
University of California, San Diego
La Jolla, CA 92093-0112, USA

April 8, 2017

Abstract

This article serves as documentation for the Matlab interface for SNOPT, a general-purpose system for constrained optimization. SNOPT minimizes a linear or nonlinear function subject to bounds on the variables and sparse linear or nonlinear constraints. It is suitable for large-scale linear and quadratic programming and for linearly constrained optimization, as well as for general nonlinear programs.

The Matlab m-files associated with the interface are available on github at:

http://github.com/snopt/snopt-matlab/releases.

The latest version is `2.3.0` and is fully compatible with SNOPT Version 7.5 and later.

For full details on SNOPT, please see the User's Guide for SNOPT.

Keywords: optimization, large-scale nonlinear programming, SQP methods, nonlinear constraints, Matlab.

*pgill@ucsd.edu (http://www.ccom.ucsd.edu/~peg)
†elwong@ucsd.edu (http://www.ccom.ucsd.edu/~elwong)

Contents

1. Introduction

`snOpt-matlab` is a Matlab interface for nonlinear optimization software SNOPT. (In particular, the Matlab interface utilizes `SNOPTA` interface of SNOPT). It minimizes a linear or nonlinear function subject to bounds on the variables and sparse linear or nonlinear constraints. It is suitable for large-scale linear and quadratic programming and for linearly constrained optimization, as well as for general nonlinear programs of the form

$$\text{NPA} \qquad \underset{x}{\text{minimize}} \quad F_{obj}(x) \qquad \text{subject to} \quad l_x \le x \le u_x, \qquad l_F \le F(x) \le u_F,$$

where x is an n-vector of variables, the upper and lower bounds are constant, $F(x)$ is a vector of smooth linear and nonlinear constraint functions $\{F_i(x)\}$, and $F_{obj}(x)$ is one of the components of F to be minimized, as specified by the input parameter `ObjRow`. (An optional parameter `Maximize` may specify that $F_{obj}(x)$ should be maximized instead of minimized.)

Ideally, the first derivatives (gradients) of F_i should be known and coded by the user. If only some gradients are known, then the missing ones are estimated by finite differences.

Note that upper and lower bounds are specified for all variables and functions. This form allows full generality in specifying various types of constraint. Special values are used to indicate absent bounds ($l_j = -\infty$ or $u_j = +\infty$ for appropriate j). Free variables and free constraints ("free rows") have both bounds infinite. Fixed variables and equality constraints have $l_j = u_j$.

In general, the components of F are *structured* in the sense that they are formed from sums of linear and nonlinear functions of just some of the variables. This structure can be exploited (see Section 3.1).

For more details on the algorithm implemented in SNOPT, please see

- P. E. Gill, W. Murray, M. A. Saunders, Elizabeth Wong. SNOPT 7.6 User's Manual. CCoM Technical Report 17-1, Center for Computational Mathematics, University of California, San Diego.
- P. E. Gill, W. Murray and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. SIAM Review 47 (2005), 99-131.

2. Compiling and Installing the Matlab interface

To compile a new Matlab mex-file, the SNOPT optimization package must be configured with the Matlab option:

```
>> ./configure --with-matlab=/path/to/matlab
```

with the appropriate path to Matlab set.

On a Linux system, the user must also configure with the option '–with-pic'.

```
>> ./configure --with-matlab=/path/to/Matlab --with-pic
```

By default, SNOPT uses its own BLAS subroutines. However, it is possible to compile the mex-file to utilize Matlab's own built-in BLAS library.

In addition to providing the location of Matlab, the user must also indicate that a third-party BLAS library will be used and also that SNOPT should be compiled with 64-bit integers. 64-bit integers are required as Matlab's built-in BLAS library only accepts 64-bit integers.

```
>> ./configure --with-matlab=/path/to/Matlab --with-blas=matlab  --with-64
```

After successful configuration, the mex-file can be built with the following command

```
>> make matlab
```

A new Matlab mex-file will be compiled and placed in the directory `${SNOPT}/matlab`.

2.1. Using precompiled Matlab libraries

The SNOPT optimization package comes with precompiled Matlab mex-files inside the directory: `${SNOPT}/matlab/precompiled`.

Libraries for several types of machines are provided:

- Linux: *.mexa64
- Mac OS: *.mexmaci64
- Windows 64-bit Matlab: *.mexw64
- Windows 32-bit Matlab: *.mexw32

The directory containing the precompiled mex-file and the SNOPT Matlab m-files (usually located in `${SNOPT}/matlab`) need to be added to the Matlab path either by using the `setpath` command or by clicking on Home → Environment → Set Path.

3. Matlab subroutines for SNOPT

The Matlab interface for SNOPT is accessed via the following Matlab subroutines:

`snopt` (Section 3.2) `snopt` is the main subroutine called to solve the nonlinear problem NLP using SNOPT.

`snsolve` (Section 3.3) is the alternative method for calling SNOPT. It is based on the Matlab function *fmincon*.

`snJac` (Section 3.4) is used to define the derivative structures needed by SNOPT.

`snend` (Section 3.5) is called at the end of a program.

`snspec` (Section 3.6) allows the user to provide a file of option specifications.

`snget`, `sngeti`, `sngetr` (Section 3.7) obtain the current value of a given option.

`snset`, `snseti`, `snsetr` (Section 3.8) set the value of the given option.

`snscreen` (Section 3.9) controls the output of SNOPT to the Matlab command screen.

`snprint` (Section 3.10) allows the user to provide the name of an output print file.

`snsummary` (Section 3.11) allows the user to redirect the summary output to a file.

`snsetwork` (Section 3.12) is an advanced routine that lets the user define the sizes of the initial integer and real workspaces for SNOPT.

3.1. Exploiting problem structure

In many cases, the vector $F(x)$ is a sum of linear and nonlinear functions. `snOpt-matlab` allows these terms to be specified separately, so that the linear part is defined just once by the input arguments `iAfun`, `jAvar`, and `A`. Only the nonlinear part is recomputed at each x.

Suppose that each component of $F(x)$ is of the form

$$F_i(x) = f_i(x) + \sum_{j=1}^{n} A_{ij} x_j,$$

where $f_i(x)$ is a nonlinear function (possibly zero) and the elements A_{ij} are constant. The `nF`$\times$`n` Jacobian of $F(x)$ is the sum of two sparse matrices of the same size: $F'(x) = G(x) + A$, where $G(x) = f'(x)$ and A is the matrix with elements $\{A_{ij}\}$. The two matrices must be *non-overlapping* in the sense that every element of the Jacobian $F'(x) = G(x) + A$ is an element of $G(x)$ or an element of A, but *not both*.

For example, the function

$$F(x) = \begin{pmatrix} 3x_1 + e^{x_2} x_4 + x_2^2 + 4x_4 - x_3 + x_5 \\ x_2 + x_3^2 + \sin x_4 - 3x_5 \\ x_1 - x_3 \end{pmatrix}$$

can be written as

$$F(x) = f(x) + Ax = \begin{pmatrix} e^{x_2} x_4 + x_2^2 + 4x_4 \\ x_3^2 + \sin x_4 \\ 0 \end{pmatrix} + \begin{pmatrix} 3x_1 - x_3 + x_5 \\ x_2 - 3x_5 \\ x_1 - x_3 \end{pmatrix},$$

in which case

$$F'(x) = \begin{pmatrix} 3 & e^{x_2} x_4 + 2x_2 & -1 & e^{x_2} + 4 & 1 \\ 0 & 1 & 2x_3 & \cos x_4 & -3 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

can be written as $F'(x) = f'(x) + A = G(x) + A$, where

$$G(x) = \begin{pmatrix} 0 & e^{x_2} x_4 + 2x_2 & 0 & e^{x_2} + 4 & 0 \\ 0 & 0 & 2x_3 & \cos x_4 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad A = \begin{pmatrix} 3 & 0 & -1 & 0 & 1 \\ 0 & 1 & 0 & 0 & -3 \\ 1 & 0 & -1 & 0 & 0 \end{pmatrix}.$$

The nonzero elements of A and G are provided to `snOpt-matlab` in coordinate form. The elements of A are entered as triples (i, j, A_{ij}) in the arrays `iAfun`, `jAvar`, `A`. The sparsity pattern of G is entered as pairs (i, j) in the arrays `iGfun`, `jGvar`. The corresponding entries G_{ij} (any that are known) are assigned to appropriate array elements `G(k)` in the user's subroutine `userfun`.

The elements of A and G may be stored in any order. Duplicate entries are ignored. As mentioned, `iGfun` and `jGvar` may be defined automatically by subroutine `snJac` (p. 9) when `Derivative option` 0 is specified and `userfun` does not provide any gradients.

3.2. Subroutine snopt

This is the main subroutine to solve problem NLP using SNOPT. The following are valid call sequences for `snopt`:

```
[...] = snopt( x, xlow, xupp, xmul, xstate, ...
               Flow, Fupp, Fmul, Fstate, ...
               userfun [, options] )

[...] = snopt( x, xlow, xupp, xmul, xstate, ...
               Flow, Fupp, Fmul, Fstate, ...
               userfun, ObjAdd, ObjRow [, options] )

[...] = snopt( x, xlow, xupp, xmul, xstate, ...
               Flow, Fupp, Fmul, Fstate, ...
               userfun, ...
               A, iAfun, jAvar, iGfun, jGvar [, options] )

[...] = snopt( x, xlow, xupp, xmul, xstate, ...
               Flow, Fupp, Fmul, Fstate, ...
               userfun, ObjAdd, ObjRow, ...
               A, iAfun, jAvar, iGfun, jGvar [, options] )
```

The output from `snopt` is:

```
[x,F,inform,xmul,Fmul,xstate,Fstate,output] = snopt( ... )
```

On entry:

`x,xlow,xupp,xmul` define the initial value of the variable `x`, the lower and upper bounds on `x`, and the initial value of the multipliers for `x`. If nothing is known about `x`, set `x` and `xmul` to 0.0.

`Flow,Fupp,Fmul` define the lower and upper bounds on the problem functions $F(x)$, and the initial value of the multipliers for the problem functions. If nothing is known about `Fmul`, set `Fmul` to 0.0.

`xstate` defines the initial states of the variables `x`.

If there is no wish to provide special information, you may set `xstate`$(j) = 0$, `x`$(j) = 0.0$ for all $j = 1:$`n`. All variables will be eligible for the initial basis.

Less trivially, to say that the optimal value of x_j will probably be one of its bounds, set `xstate`$(j) = 4$ and `x`$(j) =$ `xlow`(j) or `xstate`$(j) = 5$ and `x`$(j) =$ `xupp`(j) as appropriate.

A `CRASH` procedure is used to select an initial basis. The initial basis matrix will be triangular (ignoring certain small entries in each column). The values `xstate`$(j) = 0, 1, 2, 3, 4, 5$ have the following meaning:

`xstate`(j)	State of variable j during `CRASH`
$\{0, 1, 3\}$	Eligible for the basis. 3 is given preference
$\{2, 4, 5\}$	Ignored

After `CRASH`, variables for which `xstate`$(j) = 2$ are made superbasic. Other variables not selected for the basis are made nonbasic at the value `x`(j) (or the closest value inside their bounds). See the description of `xstate` below (on exit).

`Fstate` defines the initial states of the problem functions `F`. See the description for `xstate`.

`userfun` is a user-defined function that computes the nonlinear portion $f(x)$ of the vector of problem functions $F(x) = f(x) + Ax$, and optionally, its Jacobian $G(x)$ for a given vector x. `userfun` is either a string containing the Matlab function name or a function handle. See Section 3.13.1.

`ObjAdd` is a constant that will be added to the objective row `F(Objrow)` for printing purposes. Typically, `ObjAdd` = `0.0d+0`.

`ObjRow` says which row of $F(x)$ is to act as the objective function. If there is no such row, set `ObjRow` $= 0$. Then SNOPT will seek a *feasible point* such that $l_F \le F(x) \le u_F$ and $l_x \le x \le u_x$. ($0 \le$ `ObjRow` $\le$ `nF`)

`A,iAfun,jAvar` define the coordinates (i,j) and values A_{ij} of the nonzero elements of the linear part A of the function $F(x) = f(x) + Ax$.

The entries may define the elements of A in any order.

`iGfun,jGvar` define the coordinates (i,j) of G_{ij}, the nonzero elements of the nonlinear part of the derivatives $G(x) + A$ of the function $F(x) = f(x) + Ax$.

The entries may define the elements of G in any order, but subroutine `userfun` must define the values of `G` in exactly the same order.

`options` is an optional parameter. It is a struct containing options for `snOpt-matlab`. Currently, only two options exist:

- `options.name` provides the name of the problem
- `options.stop` is an advanced feature that allows the user to specify the "snSTOP" function called at every major iteration. `options.stop` is either a string containing the name of the Matlab function or a function handle. (See Section 3.14)
- `options.start` can be set to 'Cold' or 'Warm' to specify how a starting point will be obtained. For Cold starts, SNOPT starts "from scratch" to find an initial starting point. For Warm starts, the `xstate` and `Fstate` inputs should define a valid starting point (usually from a previous run).

On exit:

`x,F` are the values of the variables and constraints at the solution.

`inform` is the exit code returned by SNOPT. Please see the SNOPT documentation for a full description of the codes.

`xmul,Fmul` are the values of the multipliers for the variables and constraints at the solution.

`xstate,Fstate` are the final states of the variables and constraints.

`output` is a struct containing the following components:

- output.info is the same as `inform`
- output.iterations returns the total number of minor iterations for the problem
- output.majors is the total number of major iterations for the problem

3.3. Subroutine snsolve

This is the alternative *fmincon-style* subroutine to solve the following nonlinear problem using SNOPT:

fmincon-NP
$$\begin{array}{lll} \underset{x}{\text{minimize}} & f(x) & \\ \text{subject to} & c(x) \le 0 & c_{eq}(x) = 0 \\ & Ax \le b & A_{eq}(x) = b_{eq} \\ & x_{low} \le x \le x_{upp} & \end{array}$$

The following are valid call sequences for `snsolve`:

```
[...] = snsolve( myobj, x0, A, b [, options] )
[...] = snsolve( myobj, x0, A, b, Aeq, beq [, options] )
[...] = snsolve( myobj, x0, A, b, Aeq, beq, xlow, xupp [, options] )
[...] = snsolve( myobj, x0, A, b, Aeq, beq, xlow, xupp, ...
                 nonlcon [, options] )
```

The output from `snsolve` is:

```
[x,fval,exitflag,output,lambda,states] = snsolve( ... )
```

On entry:

`myobj` is a user-defined function that computes the objective function $f(x)$ at a given point x. `myobj` should be a string containing the name of the Matlab function or a function handle. See Section 3.13.2.

`x0` is the initial value of the variables.

`A,b` define the linear inequality constraints of the problem.

`Aeq,beq` define the linear equality constraints of the problem.

`xlow,xupp` define the lower and upper bounds on the variables.

`nonlcon` is a user-defined function that computes the nonlinear equality and inequality constraints of the problem, and optionally, the Jacobian matrix of these functions at a given point. `nonlcon` is either a string containing the name of a Matlab function or a function handle. See Section 3.13.2.

`options` is an optional parameter. It is a struct containing options for `snOpt-matlab`. Currently, the only valid options are:

- `options.name` provides the name of the problem
- `options.stop` is an advanced feature that allows the user to specify the "snSTOP" function called at every major iteration. `options.stop` is either a string containing the name of the Matlab function or a function handle. (See Section 3.14)

On exit:

`x` contains the values of the variables at the solution.

`fval` is the objective value at `x`.

`exitflag` is the exit code returned by SNOPT. Please see the SNOPT documentation for a full description of the codes.

`lambda` is a struct containing the multipliers at the solution.

- `lambda.lower` are the final multipliers for lower bounds
- `lambda.upper` are the final multipliers for upper bounds
- `lambda.ineqnonlin` are the final multipliers for nonlinear inequalities
- `lambda.eqnonlin` are the final multipliers for nonlinear equalities
- `lambda.ineqlin` are the final multipliers for linear inequalities
- `lambda.eqlin` are the final multipliers for linear equalities

`states` are the final states of the variables and constraints.

- `states.x` are the final states of the variables
- `states.ineqnonlin` are the final states for nonlinear inequalities
- `states.eqnonlin` are the final states for nonlinear equalities
- `states.ineqlin` are the final states for linear inequalities
- `states.eqlin` are the final states for linear equalities

`output` is a struct containing the following components:

- output.info is the same as `inform`
- output.iterations returns the total number of minor iterations for the problem
- output.majors is the total number of major iterations for the problem

---

3.4. Subroutine snJac

If the "Derivative option" is set to 0 and your user-defined routine for the objective and constraints do not provide any derivatives, you can call `snJac` to determine the input arrays `iAfun`, `jAvar`, `A`, `iGfun`, and `jGvar`. The call sequence to `snJac` is:

```
[A,iAfun,jAvar,iGfun,jGvar] = snJac(usrfun,x0,xlow,xupp,nF)
```

On entry:

`userfun` is a user-defined function that computes the nonlinear portion $f(x)$ of the vector of problem functions $F(x) = f(x) + Ax$, and optionally, its Jacobian $G(x)$ for a given vector x. `userfun` is either a string containing the Matlab function name or a function handle. See Section 3.13.1.

`x0` is the initial point x. If nothing is known, `x0` can be set to zero.

`xlow,xupp` are the lower and upper bounds on the variables.

`nF` is the number of constraints in the problem.

On exit:

`A,iAfun,jAvar` define the coordinates (i, j) of the nonzero elements of the linear part A of the function $F(x) = f(x) + Ax$.

`iGfun,jGvar` define the coordinates (i, j) of G, the nonzero elements of the nonlinear part of the derivatives $G(x) + A$ of the function $F(x) = f(x) + Ax$.

3.5. Subroutine snend

The routine `snend` should be called at the end to deallocate any memory used by SNOPT.

3.6. Subroutine snspec

`snspec` lets the user provide the name of a file with option specifications for SNOPT.

```
[inform] = snspec( filename )
```

On entry:

`filename` is a string containing the name of the specifications file to be read by SNOPT.

On exit:

`inform` reports the result of calling `snspec`. Here is a summary of possible values.

	Finished successfully
101	Specs file read.
	Errors while reading Specs file
131	No Specs file specified (`iSpecs` ≤ 0 or `iSpecs` > 99).
132	End-of-file encountered while looking for Specs file. `snspec` encountered end-of-file or `Endrun` before finding `Begin` (see the Options section of the SNOPT Manual). The Specs file may not be properly assigned.
133	End-of-file encountered before finding `End`. Lines containing `Skip` or `Endrun` may imply that all options should be ignored.
134	`Endrun` found before any valid sets of options.
>134	There were $i =$ `INFO` $- 134$ errors while reading the Specs file.

3.7. Subroutines snget, sngeti, sngetr

`snget`, `sngeti`, and `sngetr` let the user retrieve the current value of a given option. The following are the valid call sequences:

```
[option] = snget ( keyword )
[option] = sngeti( keyword )
[option] = sngetr( keyword )
```

On entry:

`keyword` is a string containing the keyword of the option to be retrived.

On exit:

`option` is the value (integer or real) of the option specified by `keyword`.

3.8. Subroutines snset, snseti, snsetr

```
snset ( keyword )
snseti( keyword, ivalue )
snsetr( keyword, rvalue )
```

On entry:

`keyword` is a string containing the keyword of the option to be retrived. For `snset`, the keyword should contain both the option name and the desired option value.

`ivalue/rvalue` is the (integer or real) value that the specified option will be set to.

3.9. Subroutine snscreen

`snscreen` controls the output of SNOPT to the Matlab command screen.

```
snscreen on
snscreen off
```

3.10. Subroutine snprint

`snprint` lets the user provide a filename for the print output from SNOPT.

```
snprint( filename )
```

On entry:

`filename` should be a string specifying the filename to be used.

3.11. Subroutine snsummary

`snsummary` controls the summary output of SNOPT.

```
snsummary off
snsummary( filename )
```

On entry:

`off` turns the summary output off.

`filename` is the name of the file that the summary output is redirected to.

3.12. Subroutine snsetwork

`snsetwork` is an advanced subroutine that allows the user to define the initial sizes of the integer and real workspaces for SNOPT. By default, workspace size is initialized to 5000. However, for larger problems or dense problems, a much larger value may be required. This subroutine should be called before any other subroutines.

```
snsetwork( leniw, lenrw )
```

On entry:

`leniw,lenrw` are integers specifying the size of the integer and real workspace for SNOPT. These values should be at least 500.

3.13. User-defined subroutines

Both `snopt` and `snsolve` require user-defined functions as arguments. These functions compute the objective function and constraints, and if necessary, the gradients of these functions, at a given point.

3.13.1. Function `userfun`

For `snopt`, the user provides a function `userfun` that computes the nonlinear portion $f(x)$ of the vector of problem functions $F(x) = f(x) + Ax$, and optionally, its Jacobian $G(x)$ for a given vector x.

The valid function declarations for `userfun` are:

```
[F]   = myFun(x)
[F,G] = myFun(x)
```

If the user does not provide any derivative information for F, then G can be omitted from the output. In this case, `snOpt-matlab` will estimate the derivatives by finite differences. The user can also provide *some* of the derivatives of F. Any unknown derivatives should be marked by a `NaN`.

In addition, the user can check the variable `nargout` to determine if computation of the derivatives is necessary. In particular, if `nargout` is greater than 1, then G is requested and should be computed if possible. Otherwise, only F is required.

3.13.2. Functions `myobj` and `nonlcon`

For `snsolve`, the user provides `myobj`, which computes the objective function $f(x)$ at a given point x, and (possibly) `nonlcon`, which computes the nonlinear equality and inequality constraints of the problem, and optionally, the Jacobian matrix of these functions at a given point.

The valid function declarations for `myobj` and `nonlcon` are:

```
[f]   = myObj(x)
[f,g] = myObj(x)
```

The valid function declarations for `nonlcon` are:

```
[c,ceq]          = myCon(x)
[c,ceq,dc,dceq] = myCon(x)
```

where `c` and `ceq` are the nonlinear inequality and equality constraints, and `dc` and `dceq` are the gradients of the nonlinear inequality and equality constraints.

If no derivative information is provided, then the user can omit `g`, `dc`, and `dceq` from the output. In this case, `snOpt-matlab` will estimate the derivatives by finite differences.

For both `myobj` and `nonlcon`, the user can provide *some* of the derivatives of the objective and constraints. Any unknown derivatives should be marked by a `NaN`.

In addition, the user can check the variable `nargout` to determine if computation of the derivatives is necessary. In particular, for `myobj`, if `nargout` is greater than 1, then `g` is requested and should be computed if possible; for `nonlcon`, if `nargout` is greater than 2, then `dc` and `dceq` are requested and should be computed if possible.

3.14. snSTOP Feature

`snSTOP` is an advanced feature for SNOPT. The user can provide a function with the following declaration:

```
[iAbort] = toySTOP( itn, nMajor, nMinor, condZHZ, obj, merit, step, ...
                    primalInf, dualInf, maxViol, maxViolRel, ...
                    x, xlow, xupp, xmul, xstate, ...
                    F, Flow, Fupp, Fmul, Fstate )
```

This function will be called at the end of *every* major iteration. Current values relevant to the run are passed to the STOP function. If `iAbort` is set to a nonzero value, then SNOPT will terminate the current run.

4. Frequently Asked Questions

- Why is the Matlab interface to SNOPT so slow compared with (`fmincon` | the Fortran version of SNOPT/NPSOL| etc.)?

  There could be a number of things affecting the speed of `snOpt-matlab`:

  1. There is a lot of overhead in the Matlab interface. The interface works by taking the Matlab input, and transforming and converting it into Fortran input for the SNOPT library. In particular, for every callback to the user-defined functions (`myobj`, `nonlcon`, or `userfun`), the interface has to allocate memory for input and output, and then copy that data between the Fortran library and Matlab. We try to minimize the work for these callbacks as much as possible.
  2. Are you providing derivative information to SNOPT? The more information you give SNOPT, the better. If no derivative information is provided, then SNOPT uses finite-differencing to estimate the values, which can take more time. Even partial derivative information can help speed things up. Providing the derivatives in sparse format would also speed things up.
  3. Is your problem very dense? SNOPT is intended for *sparse* problems. In particular, if the Jacobian matrix is very dense, then the interface has to copy a dense matrix back and forth between Fortran and Matlab with every callback to the user-defined functions. It would be ideal if the Jacobian of the constraints be provided in a sparse format (and not as a dense matrix).

- I tried to compile the Matlab mex-files on my Linux machine, but I get the error:

```
/usr/bin/ld: ./lib/libsnmex7.a(snopta.o): relocation R\_X86\_64\_32
against '.rodata' can not be used when making a shared object;
recompile with -fPIC
```

How do I fix this?

Reconfigure the software package with "–with-pic" and then recompile. Also make sure to do a "make veryclean" before you compile again.