AIAA-2001-1620

DEVELOPMENT AND VERIFICATION OF A MATLAB DRIVER FOR THE SNOPT OPTIMIZATION SOFTWARE

Shawn E. Gano * **Victor M. Pérez** † **John E. Renaud** ‡

Department of Aerospace and Mechanical Engineering
University of Notre Dame
Notre Dame, Indiana
Email: John.E.Renaud.2@nd.edu

Abstract

The MATLAB program and computing language has seen increased usage both in industry and academia in recent years. This is due to the ease in which it handles matrices and numerical computations. This computing environment also has an array of toolboxes for different mathematical and engineering tasks (e.g., controls, optimization). These toolboxes provide a general suite of numerical tools within a specific discipline for the user. The toolbox codes are general tools and are not typically as robust or as efficient as state of the art numerical codes develop by advanced users in a given discipline. In this research a MATLAB driver which links an existing robust and efficient optimization program SNOPT is developed and tested. The resulting program and driver have proved to be more efficient than the existing MATLAB toolbox codes for optimization.

1 Introduction

SNOPT is a state of the art sparse nonlinear programming tool for optimizing problems consisting of large numbers of variables and constraints. It was developed by Stanford Business Software Inc., the developer of several other optimization programs such as MINOS and NPSOL. Written in FORTRAN 77, SNOPT employs a sparse SQP algorithm with limited-memory quasi-Newton approximations to the Hessian of Lagrangian. In this research a MATLAB driver for the SNOPT software is developed and tested.

In today's academic environment FORTRAN is less commonly used as an engineering programming language, and therefore SNOPT's problem solving capabilities are artificially restricted to FORTRAN users. The programming tool/language MATLAB has become a powerful and widely used tool in academic research. Researchers take advantage of MATLAB's powerful features and flexibility to write a wide variety of computational tools. MATLAB includes a range of specialty toolboxes for applications ranging from control systems, artificial neural networks and optimization. While the toolboxes provide a variety of capabilities, they are by no means state of the art in capability. In practice, researchers might prefer the efficient SNOPT algorithms over MATLAB's own optimization packages.

In a recent investigation by Gu, et al.[4] a decision based collaborative optimization framework was implemented exclusively in MATLAB's programming environment. In that investigation the use of MATLAB's SQP algorithm for optimization was observed to be unstable and performed poorly in managing the decision based collaborative optimization process. The SNOPT-MATLAB driver being developed in this research will facilitate the use of SNOPT as a best inclass SQP algorithm for driving decision

*Undergraduate Research Assistant, Student Member AIAA
†Graduate Research Assistant, Student Member AIAA
‡Associate Professor, Associate Fellow AIAA.

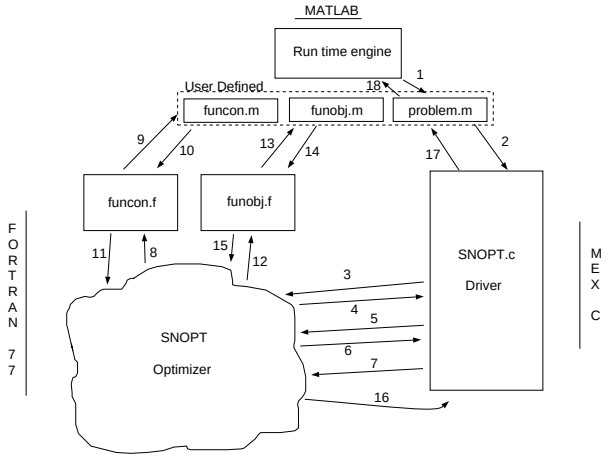

Figure 1. SNOPT-MATLAB Driver Flow Chart

based collaborative optimization algorithm of Gu, et al.[4].

The MATLAB driver for SNOPT is available to users via the Internet at the Design Automation Lab. webpage: *http://www.nd.edu/ nddal/index.htm*

2 Program Hierarchy

To run a problem in the stand-alone version of SNOPT, two subroutines and one general program have to be written in FORTRAN. The general program would define the problem and call the SNOPT routines. SNOPT would call the other two subroutines for the objective function, constraints, and their respective gradients (if known). This process is mimicked in the driver. The user creates a script (*.m) file in MATLAB that sets up the problem and calls the SNOPT driver. Also the user creates two MATLAB file functions that compute the objective function, constraints and respective gradients. The flow of the program is depicted in Figure 1.

Every problem run using the driver goes through 18 basic steps. These steps are numbered in Figure 1 and are described by: (1) The problem .m script file is invoked in the MATLAB runtime engine. (2) After computing or assigning all needed parameters the problem file calls the SNOPT driver. (3) The SNOPT driver, written in MEX C,* imports the MATLAB variables, allocates the workspace memory, and makes a call to SNINIT which is the initial call to the SNOPT program. (4) SNOPT returns to the driver after it has initialized the optimization program. (5) The driver calls SNSPEC which tells SNOPT where to find the options file (if it exists). (6) SNOPT returns to the driver after it has read the options file (if possible). (7) The driver makes the final call to SNOPT for optimization of the problem. (8) SNOPT calls funcon.f for the constraints and possibly their gradients. (A call to funobj.f may occur first depending on initial variable conditions). (9) funcon.f calls the user defined MATLAB code for the constraints and gradients. (10) funcon.m (MATLAB) returns the constraints and the gradients. (11) funcon.f converts the MATLAB variables to FORTRAN and then returns to SNOPT. (12) SNOPT calls funobj.f for the objective function and its gradient. (A call to funcon.f may occur second depending on initial variable conditions). (13) funobj.f calls the user defined MATLAB code for the objective function and its gradient. (14) funobj.m (MATLAB) returns the objective function and possibly the gradient. (15) funobj.f converts the MATLAB variables to FORTRAN and then returns to SNOPT. (16) After repeating steps 8 through 12 as many times as needed to find the optimum solution SNOPT finishes and returns the final values to the driver. (17) The MEX C driver converts the return variables to MATLAB and exits. (18) The problem .m script finishes and returns to the MATLAB engine.

3 Problem Setup

Setting up a problem for running in MATLAB is identical to the procedure used in setting up the problem in FORTRAN, and is described in more detail in the SNOPT User's Guide[3].

3.1 Problem Definition

SNOPT solves problems which are assumed to be stated in the form

$$\begin{array}{l} \underset{x}{\text{minimize}}\ f(x) \\ \text{subject to } \mathbf{l} \leq \begin{pmatrix} \mathbf{x} \\ \mathbf{F}(x) \\ \mathbf{G}x \end{pmatrix} \leq \mathbf{u} \end{array}$$

where f is a smooth scalar objective function, $\mathbf{x}$ are design variables, $\mathbf{l}$ is a constant lower bound, $\mathbf{u}$ is a constant upper bound, $\mathbf{F}(x)$ is a vector of smooth

*Mex C: compiled C code for MATLAB

nonlinear constraint functions, and $\mathbf{G}x$ is the vector of linear constraints.

Note that all variables and constraints have their own unique upper and lower bounds. Free variables and free constraints are ones that have infinite bounds.

SNOPT converts the inequality constraints to equality constraints by means of slack variables such that the slack variables have the same lower and upper bounds as the constraints.

3.2 Jacobian and Sparcity Syntax

The full Jacobian matrix is defined as

$$A = \begin{pmatrix} \frac{\partial F(x)_i}{\partial x_j} \\ Gx_i \end{pmatrix}$$

The sparcity pattern of A is input column-wise via the array parameters a, ha, ka. While zero entries do not need to be included. For a more complete definition of the Jacobian matrix, refer to the *User's Guide for SNOPT*[3].

4 The program.m File

4.1 The SNOPT Call and Variable Definitions

The program .m file is a MATLAB script in which all of the problem variables are set and the SNOPT Driver is called. A few of the required parameters for running the FORTRAN optimizer have been hard coded into the driver minimizing the amount of parameters passed to the driver from MATLAB. The call to the driver should be in the form

```
[hs, xs, pi, rc, inform, mincw, miniw, minrw,
ns, ninf, sinf, obj] =snopt( m, n, ne, nncon,
nnobj, nnjac, iobj, objadd, prob, a, ha, ka,
bl, bu, xs, ru, op1, op2, op3, op4);
```

Note: this should all be on one line

The input variables are: (**m**) Number of general constraints $(m > 0)$. (**n**) Number of variables, excluding slacks $(n > 0)$. (**ne**) Number of nonzero entries in A. (**nncon**) Number of nonlinear constraints $(nncon \geq 0)$. (**nnobj**) Number of nonlinear objective variables $(nnobj \geq 0)$. (**nnjac**) Number of nonlinear Jacobian variables. (**iobj**) Defines which row of A is a free row containing a linear objective vector **c**. (**objadd**) A constant added to the objective for purposes of printing. (**prob**) An eight character name for the problem. (**a(ne)**) The list of nonzero entries in the full Jacobian. (**ha(ne)**) The corresponding row number for each value of a. (**ka(n+1)**) Defines the beginning of each new column of the full Jacobian. (**bl(n+m)**) Contains the **lower** bounds of the variables and constraints (slacks) (x,s). (**bu(n+m)**) Contains the **upper** bounds of the variables and slacks (x,s). (**xs(n+m)**) Initial values for the variables and slacks. (**ru()**) Array of user workspace. Can pass any set of numbers to the function routines. (**op1**) Additional memory option for the character workspace. (**op2**) Additional memory option for the integer workspace. (**op3**) Additional memory option for the double (real) workspace. (**op4**) When $op4 = 1$ the variables passed from MATLAB to the driver are printed to the screen (useful in debugging).

The output variables are: (**hs(n+m)**) The final state vector. (**xs(n+m)**) Final variables and slacks (x,s). (**pi(m)**) The vector of dual variables π (a set of Lagrange multipliers for the general constraints). (**rc**) Vector of reduced costs. (**inform**) Reports the result of the SNOPT call. (**mincw**) Reports how much character storage is needed to solve the problem. (**miniw**) Reports how much integer storage is needed to solve the problem. (**minrw**) Reports how much double (real) storage is needed to solve the problem. (**ns**) The final number of superbasic variables. (**ninf**) The number of constraints that lie outside their bounds by more than the feasibility tolerance (violated constraints). (**sinf**) The sum of violated constraints. (**obj**) Value of the objective function, including the constant **objadd**.

4.2 Building Variables a(ne), ha(ne), and ka(n+1)

The sparcity of the full Jacobian matrix A is represented in the 3 arrays $a(ne)$, $ha(ne)$, and $ka(n+1)$. Only the nonzero terms of the matrix are defined.

The variable $a(ne)$ contains a list of all the nonzero entries. These entries are just given by their constant gradient values if linear, or a dummy value, like 0, if they are not linear. Entries are typed column wise starting in the upper left hand corner and working down and to the right. Figure 2 shows how the a(ne) array is constructed. If the ith entry is a constant then $a(i)$ is that constant otherwise $a(i) = 0$ (or any other dummy value).

The vector $ha(ne)$ contains the row number for

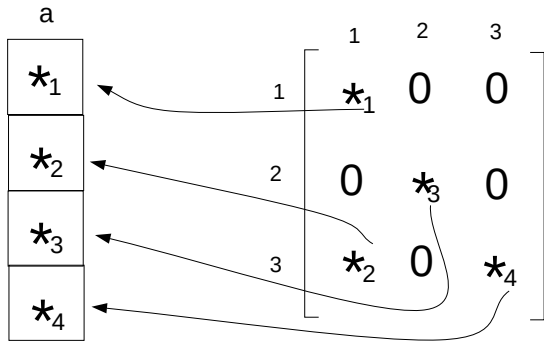

Figure 2. Variable a(ne) Structure

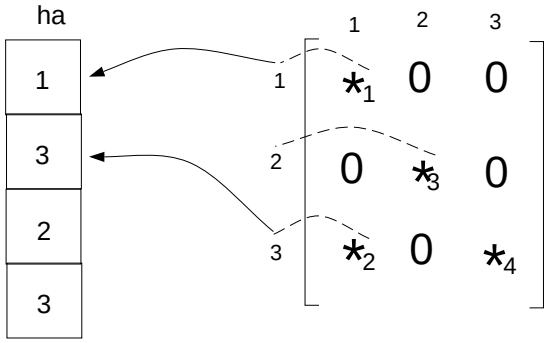

Figure 3. Variable ha(ne) Structure

each nonzero entry. Such that $ha(i) =$ row $a(i)$ where $i = 1 \ldots ne$. This process is depicted in Figure 3.

Finally, $ka(n+1)$ contains the index number for the first nonzero entry in each column and the last number of the vector must be $ne+1$. This results in $ka(1) = 1$ and $ka(n+1) = ne+1$ always. See Figure 4 for a visualization of this process.

For a more concise definitions of $a(ne)$, $ha(ne)$, and $ka(n+1)$ refer to the *User's Guide for SNOPT*[3].

5 The funobj.m File

The funobj.m is a MATLAB function file. It computes the objective function and possibly it's gradients. The first line of the file must have the form

```
function [fobj,gobj,mode,grad] =
funobj(x,mode,nstate,ru)
```

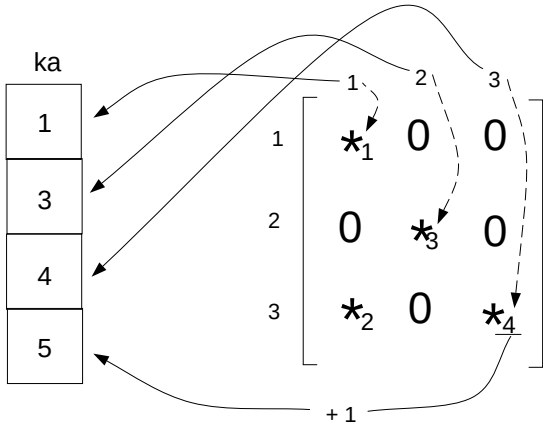

Figure 4. Variable ka(n+1) Structure

The input variables are: (**x(nnobj)**) Contains the nonlinear objective variables x. (**mode**) Indicates whether $fobj$ or $gobj$ or both need to be assigned during the present call. (**nstate**) Indicates the first and last calls to the function. (**ru()**) The workspace array of user defined values.

The output variables are: (**obj**) Returns the computed value of $f(x)$. (**gobj(nnobj)**) Returns the gradient vector $g(x)$. (**mode**) May be used to indicate that you are unwilling to evaluate the objective function at the current x. (**grad**) Specifies whether or not the user is supplying the gradients (gobj).

6 The funcon.m File

The funcon.m file is a MATLAB function file. It computes the nonlinear constraints and possibly their gradients. The Gradients are stored column wise in the array *gcon*. *gcon* is a one dimensional array and must store the Jacobian values in the same order as the corresponding parts of *a*, *ha*, and *ka*. The file's first line must have the form

```
function [fcon, gcon, mode, grad] =
funcon(x,mode,nstate,ru)
```

Where the input variables are: (**x(nnjac)**) Contains the nonlinear Jacobian variables x. (**mode**) Indicates whether $fcon$ or $gcon$ or both need to be assigned during the present call. (**nstate**) Indicates the first and last calls to the function. (**ru()**) The workspace array of user

defined values.

Where the output variables are: (**fcon(nncon)**) Vector containing the values of the nonlinear constraints. (**gcon(nejac)**) Where $nejac = nncon * nnjac$. Returns the computed Jacobian of the nonlinear constraints. (**mode**) May be used to indicate that you are unwilling to evaluate the objective function at the current x. (**grad**) Specifies whether or not the user is supplying the gradients (gcon).

7 Test Problems

To test the SNOPT-MATLAB driver, two MDO problems are used. The first is a trivial problem given as an example of how to set a problem up for running the driver. The second problem is a more costly structural problem used to compare MATLAB's optimization tool box to that of the driver developed in this paper.

7.1 An Academic Example Problem

This is a small example on how to setup and run the SNOPT driver for MATLAB. The problem was taken from *Optimization Concepts and Applications in Engineering* by Ashok D. Belegundu and Tirupathi R. Chandrupatla[1].

$$\begin{array}{l} \underset{x_1,x_2}{minimize}\ f(x_1,x_2) = 0.01x_1^2 + x_2^2 \\ \text{subject to: } 0 \leq \begin{pmatrix} g_1 = x_1x_2 - 25 \\ g_2 = x_1 - 2 \\ x_1 \\ x_2 \end{pmatrix} \leq \infty \end{array}$$

This example is in the right form already so we can start computing each needed variable from section 3.1. Since there are two general constraints, g_1 and g_2, thus $m = 2$. This problem has two variables, x_1 and x_2, so $n = 2$. Next we have to compute the full Jacobian

$$A = \begin{pmatrix} \frac{\partial g_1(x_1,x_2)}{\partial x_1} & \frac{\partial g_1(x_1,x_2)}{\partial x_2} \\ \frac{\partial g_2(x_1,x_2)}{\partial x_1} & \frac{\partial g_2(x_1,x_2)}{\partial x_2} \end{pmatrix} = \begin{pmatrix} x_2 & x_1 \\ 1 & 0 \end{pmatrix}$$

Identifying three nonzero entries in the Jacobian yields, $ne = 3$. g_1 is the only nonlinear constraint so $nncon = 1$. In the objective function, $f(x_1,x_2)$, both variables are nonlinear thus $nnobj = 2$. There are two linear variables in the constraints therefore $nnjac = 2$. As there is no linear objective vector **c** added to the Jacobian, $iobj = 0$. There is no bias of the function thus $objadd = 0.0$. Giving a name to the problem we set $prob =$ 'Example'. Using the method from section 3.2; $a(1) = 0$ since the first nonzero element of A, x_2, is not constant, $a(2) = 1$ since that is the value of the second nonzero element, and $a(3) = 0$ since again the third nonzero entry, x_1 is not constant, $a^T = (0,1,0)$. Also, $ha(1) = 1$ because the first nonzero entry (x_2) is in the first row, $ha(2) = 2$ because the second nonzero entry (1), and $ha(3) = 1$ because x_1 is in the row 1, $ha^T = (1,2,1)$. Likewise, $ka(1) = 1$ since the first nonzero entry in column 1 is the first nonzero entry in the matrix, $ka(2) = 3$ since the third nonzero entry in the matrix is the first nonzero entry in row 2, and $ka(3) = 4$ because $ka(n+1) = ne+1$ always, $ka^T = (1,3,4)$. The first to entries of $bl(n+m)$ are 0 since both x_1 and x_2 are greater than or equal to 0. The third entry of bl is also 0 because g_1's slack is 0 (though we could rearrange the equation to yield a slack of 25 and not include this value in funcon.m). $bl(4) = 2$ because g_2 is linear and by rearranging we get

$$2 \leq g_2 = x_2 \leq \infty$$

so the slack of g_2 is 2. The upper bounds for all variables and slacks is ∞ thus $bu = [1e100, 1e100, 1e100, 1e100]$ or any comparable relatively high values. Next set the initial points for the variables and slacks $xs = [0,0,0,0]$ (As we do not know anything about the slacks we use 0 as the starting point, any other desired starting point is acceptable). This problem doesn't require any user variables so $ru = []$ and none of the options are needed. The last step is to create the MATLAB .m file which might look like:

```
% example.m an example file
% for running the SNOPT driver

m       = 2;
n       = 2;
ne      = 3;
nncon   = 1;
nnobj   = 2;
nnjac   = 2;
iobj    = 0;
objadd  = 0.0d+0;
prob    = 'Example';
a       = [0,1,0];
ha      = [1,2,1];
```

```
ka     = [1,3,4];
bl     = [0,0,0,2];
bu     = [1e100,1e100,1e100,1e100];
xs     = [0,0,0,0];
ru     = [];

[hs,xs,pi,rc,inform,mincw,miniw,...
minrw,ns,ninf,sinf,obj]=...
snopt(m,n,ne,nncon,nnobj,nnjac,...
iobj,objadd,prob,a,ha, ka,bl,..
bu,xs,ru);
```

The next step is to create the funobj.m file. First we calculate all the gradients of the objective function $f(x_1,x_2)$,

$$\frac{\partial f(x_1,x_2)}{\partial x_1} = 0.02x_1$$

and

$$\frac{\partial f(x_1,x_2)}{\partial x_1} = 2x_2$$

Then we are set to create the MATLAB code, which might look like:

```
function [fobj,gobj,mode, grad]...
 = funobj(x,mode,nstate,ru)
% funobj.m function file for
% SNOPT driver problem example.m

fobj = 0.01*x(1)^2 + x(2)\^ 2;

gobj(1) = 0.02 * x(1);
gobj(2) = 2*x(2);

grad = 1;
% Suppling the gradients.
```

We continue the example by creating the funcon.m file. In setting up the function the first thing we must do is specify what constraints are nonlinear, in this case just g_1. Then we compute the Jacobian of the nonlinear constraints,

$$A = \left(\frac{\partial g_1(x_1,x_2)}{\partial x_1} \;\; \frac{\partial g_1(x_1,x_2)}{\partial x_2} \right) = \left(x_2 \;\; x_1 \right)$$

There are two non-constant terms in the Jacobian therefore, $gcon(1) = x_2$ and $gcon(2) = x_1$. We are set to create the MATLAB code, which might look like:

```
function [fcon, gcon, mode, grad]...
 = funcon(x,mode,nstate,ru)
```

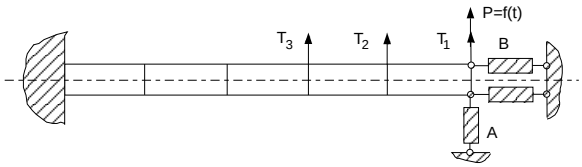

Figure 5. Cantilever beam with actuators.

```
% funcon.m function file for SNOPT
% driver problem example.m

fcon(1) = x(1)*x(2) - 25;

gcon(1,1) = x(2);
gcon(1,2) = x(1);

grad = 1;
% Suppling the gradients.
```

Running the example problem yields the following results:

```
xs =

   15.8114
    1.5811
         0
   15.8114

obj =

    5.0000
```

7.2 Control-Augmented Structure Problem

The control-augmented structure design problem shown in Figure 5 was introduced by Sobieszczanski-Sobieski *et al.*[13]. The problem comprises a total of 11 design variables and 43 states. The physical problem consists of a cantilever beam subjected to static loads along the beam and to a dynamic excitation force applied at the free end. Two sets of actuators are placed at the free end of the beam to control both the lateral and rotational displacement.

The system analysis is comprised of two coupled contributing analysis as shown in Figure 6. The structures subsystem, CA_s consists of a finite element model of the beam where the natural frequencies and modes of the cantilever beam are computed. CA_s requires, in

addition to the characteristics of the beam, the weight of the control system as input. The weight of the control system is calculated at the controls CA, CA_c. The weight of the control system is a function of the dynamic displacements and rotations of the free end of the beam. These dynamic displacements and rotations are functions of the natural frequencies and modes obtained in the structures CA, thus subjecting these CAs to coupling.

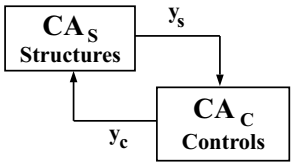

Figure 6. Dependency diagram of the Control-Augmented Structure design problem.

The objective of the optimization is to minimize the total weight of the system W_t, composed of the weight of the beam W_s plus the weight of the control system W_c. The minimization is subjected to seven constraints on the static stresses, lateral and rotational displacements, natural frequencies and dynamic lateral and rotational displacements at the free end of the beam. The problem is posed as:

$$\min W_t = W_s + W_c$$

subject to

$$g_1 = 1 - \frac{dl}{dl_a} \geq 0,$$
$$g_2 = 1 - \frac{dr}{dr_a} \geq 0,$$
$$g_3 = \frac{\omega_1}{\omega_{1_a}} - 1 \geq 0,$$
$$g_4 = \frac{\omega_2}{\omega_{2_a}} - 1 \geq 0,$$
$$g_5 = 1 - \frac{\sigma}{\sigma_a} \geq 0,$$
$$g_6 = 1 - \frac{ddl}{ddl_a} \geq 0,$$
$$g_7 = 1 - \frac{ddr}{ddr_a} \geq 0,$$

where dl is the static lateral displacement, dr is the static rotational displacement, ddl is the dynamic lateral displacement, ddr is the dynamic rotational displacement, ω_1 is the first natural frequency, ω_2 is the second natural frequency, and σ is the static stress. The subscript a stands for the allowed value. The optimum for this problem is depicted in Table 1. The minimum weight, $W = 1493.6\ lbs$ occurs where 6 design variables are at their bounds.

Design Variable	Starting Design	Optimum Design
b_1 (in)	10.0	3.0
b_2 (in)	10.0	3.0
b_3 (in)	10.0	3.0
b_4 (in)	10.0	3.0
b_5 (in)	10.0	3.0
h_1 (in)	10.0	13.85
h_2 (in)	10.0	11.96
h_3 (in)	10.0	9.78
h_4 (in)	10.0	7.06
h_5 (in)	10.0	3.75
c	0.01	0.06

Table 1. Starting and optimum designs for the Constrol-Augmented Structure problem.

8 Testing Methodology

To test the SNOPT driver and underlying program against the MATLAB optimization toolbox (version 2.0), we ran control-augmented structure problem on similar computers using both optimizers.

Comparison between the driver proposed in this paper and the MATLAB optimizer is done in two ways. First, by plotting each design variable as a function of iteration and by plotting the objective function versus iteration for each optimizer.

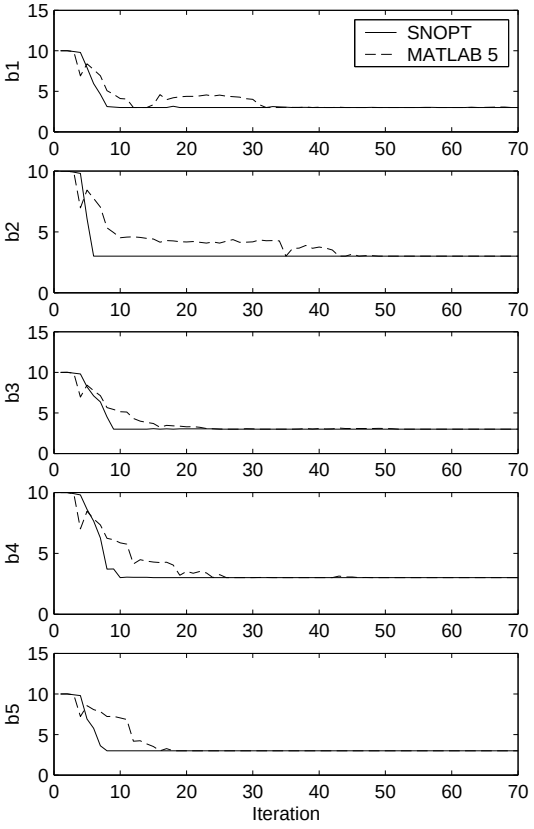

Figure 7. Convergence of the b_i design variables (width)

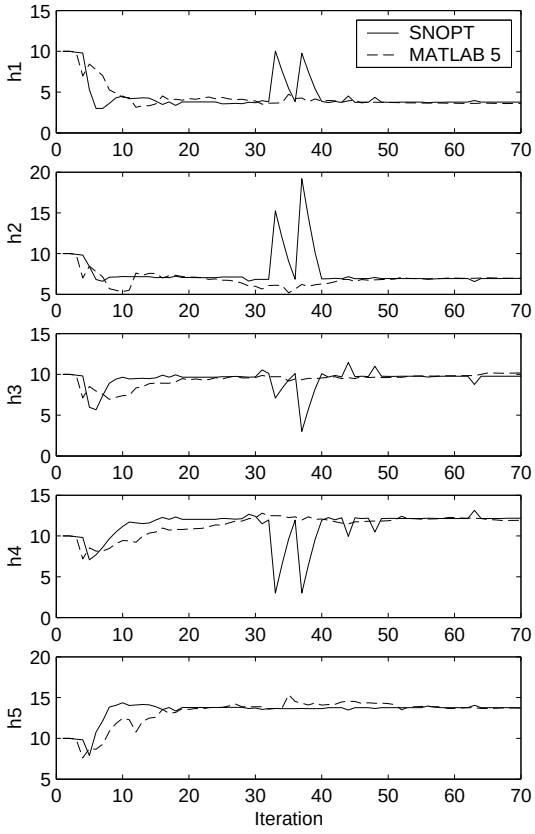

Figure 8. Convergence of the h_i design variables (height)

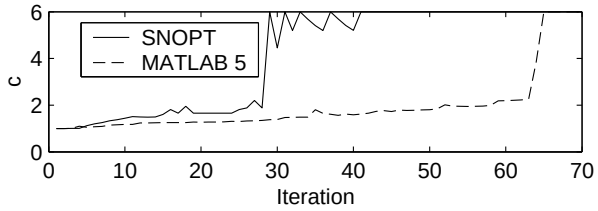

Figure 9. Convergence of the c design variable

9 Results

In Figures 7 - 9 each design variable is plotted as a function of iteration. These plots compare the convergence of the SNOPT and MATLAB optimizers for each variable. Figure 7 shows the 5 b design variables, likewise Figure 8 shows the 5 h variables. Finally, Figure 9 specifically shows the single c variable.

The convergence of the objective function of the two optimizers is shown in Figure 10. The optimal value of the function is $W = 1493.6\ lbs$ which is depicted as 1.4936 in the vertical axis of the plot.

10 Discussion

The SNOPT optimizer used through the driver developed in this paper does indeed converge faster in this example than MATLAB's optimization toolbox. Figure 7 depicts all five variables converged faster when run with the SNOPT optimizer. In Figure 8 three of the variables converged fastest using MATLAB's toolbox. The last variable converged much faster with the SNOPT

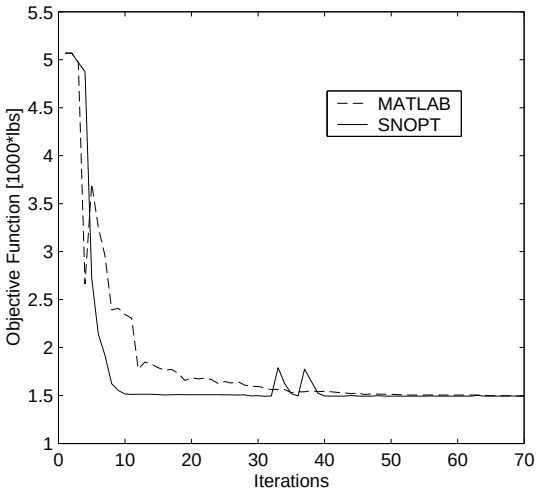

Figure 10. Objective Function Convergence of the SNOPT and MATLAB Optimizers

program, as seen in Figure 9.

By looking at the convergence of the objective function we can get a better quantitative look at the convergence rate difference between the two methods. In the first few iterations the MATLAB optimizer seemed to converge faster, but then after about 7 iterations the SNOPT optimizer converged considerably faster, with exception to a few small bases changes in the 30 to 40 iterations range. However on the larger view, the SNOPT optimizer converged to 99.9% of the minimum objective function value in 29 iterations. It took MATLAB 69 iterations to reach the same criteria.

11 Conclusions

A driver which facilitates the use of the state of-the-art optimization software, SNOPT, within a MATLAB environment was developed in this research. The driver was successfully tested and run using both academic and real engineering problems. The convergence of the optimization software was compared to MATLAB's own optimization toolbox. These tests showed that SNOPT minimized the objective function faster and thus less expensively. The MATLAB SNOPT capability developed in this research provides a new tool for engineers or others who want to run optimization problems within MATLAB.

Acknowledgments

This multidisciplinary research effort is supported in part by the following grants and contracts; NSF grant DMI98-12857 (NSF REU supplement), NASA grant NAG1-2240 and CONACyT, Mexico.

References

1.- Belegundu, Ashok D., Chandrupatla, Tirupathi R.: *Optimization Concepts and Applications in Engineering*, Prentice Hall, 1999.

2.- Deitel, H.M., Deitel, P.J.: *C How to Program*, Prentice Hall, 1994.

3.- Gill, Philip E., Murray, Walter, Saunders, Michael A.: *User's Guide to SNOPT 5.3: A Fortran Package for Large-Scale Nonlinear Programming*, December 1998.

4.- Gu, Xiaoyu, Renaud, John E., Ashe, Leah M., Batill, Stephen M., Budhiraja, Amarjit S., Krajewski, Lee J. : *Decision-Based Collaborative Optimization under Uncertainty*, DETC2000/DAC-14297 2000.

5.- Hanselman, Duane C., Littlefield, Bruce C.: *Mastering MATLAB 5: A Comprehensive Tutorial and Reference*, Prentice Hall, 1997.

6.- Hillier, F.S., Lieberman, G.J.: *Introduction to Operations Research*, McGraw Hill, 2001.

7.- Kopka, Helmut, Daly, Patrick W.: *A Guide to LaTeX*, Addison-Wesley, third edition 1999.

8.- Mashaw, Bijan: *Programming Byte by Byte: Structured FORTRAN 77*, Little, Brown and Company, 1983.

9.- Mathworks, Inc. *MATLAB: External Interface Guide*, 1992.

10.- Perez, V.M., Renaud, J.E., Gano, S.E., 2000, *Constructing Variable Fidelity Response Surface Approximations on the Usable Feasible Region*, Proceedings of the 8th AIAA/USAF/NASA/ISSMO Symposium on Multidsciplinary Analysis & Optimization, AIAA 2000-4888, Long Beach, CA, September 6-8.

11.- Rao, Singiresu S., *Engineering Optimization*, Wiley, 1996.

12.- Reklaitis, G.V., Ravindran, A., Ragsdell, K.M., *Engineering Optimization Methods and Applications*, Wiley, 1983.

13.- Sobieszczanski-Sobieski, J.; Bloebaum, C. L. ; Hajela, P.1991: Sensitivity of Control-Augmented Structure Obtained by a System Decomposition Method. *AIAA Journal*. Vol. 29, No. 2, February, pp. 264–270.