ALGORITMI R&D CENTER
Systems Engineering Group
Universidade do Minho
Campus de Gualtar
4710-057 Braga, Portugal

SIPAMPL: Semi-Infinite Programming with AMPL, v2.0

A. Ismael F. Vaz
Edite M.G.P. Fernandes
M. Paula S.F. Gomes

Technical Report
ALG/EF/4-2002

December 2002
SIPAMPL: Semi-Infinite Programming with AMPL, v2.0

A. Ismael F. Vaz,
Edite M.G.P. Fernandes\footnote*{Departamento de Produção e Sistemas, Escola de Engenharia, Universidade do Minho Campus de Gualtar, 4710 Braga, Portugal; Email: \{aivaz,emgpf\}@dps.uminho.pt} and
M. Paula S.F. Gomes\footnote{Mechanical Engineering Department, Mechatronics in Medicine Laboratory, Imperial College of Science, Technology and Medicine, London SW7 2BX, UK; Email:p.gomes@ic.ac.uk}

April 2000

Abstract

This report describes an environment for coding semi-infinite programming (SIP) problems. This environment is mainly an interface wrapper for AMPL, which we have coined as SIPAMPL. A large set of problems has been collected from the literature and has been codified using AMPL. We trust this development will be of aid to researchers benchmarking SIP algorithms. As a concept demonstration, we show how a commercial SIP solver (MATLAB) can be interfaced with SIPAMPL. Version 2.0 proposes another approach to interface MATLAB with SIPAMPL. The Linux and Microsoft Windows versions, and the database of codified problems are freely available via the web.

Keywords: Semi-infinite programming, test problems, evaluation tools.

1 Change Log

Version 2.0 includes the \texttt{select} tool with new features, compiles with Microsoft Visual C/C++ and has a new MATLAB interface.
2 Introduction

In this report, we describe an environment for coding semi-infinite programming (SIP) problems. This environment is mainly an interface wrapper for AMPL (see http://www.ampl.com), which we have coined as SIPAMPL. AMPL does not directly support SIP problems. We provide an extension to enable the codification of SIP problems in AMPL. The resolution of SIP problems is outside the scope of this report. We start in section 3 by giving a description of a SIP problem and present a short SIP example coded in AMPL. A brief description of AMPL is done in section 4. The interface between AMPL and a SIP solver is described in section 5. In section 6 we describe the collection of coded SIP problems. An interface to MATLAB Optimization Toolbox is also available and it is described in section 7. Section 8 describes the new approach for using SIPAMPL with MATLAB. Some discussions and the conclusions are presented in the last two sections. The Appendix contains a brief description on how to hook a solver to AMPL and the SIPAMPL directory structure.

3 Semi-Infinite Programming

A SIP problem is a mathematical program of the form:

\[
\begin{align*}
\min_{x \in \mathbb{R}^n} & \quad f(x) \\
\text{s.t.} & \quad g_i(x, t) \leq 0, \quad i = 1, \ldots, m \\
& \quad h_i(x) \leq 0, \quad i = 1, \ldots, o \\
& \quad h_i(x) = 0, \quad i = o + 1, \ldots, q \\
& \quad \forall t \in T,
\end{align*}
\]

(1)

where \( f(x) \) is the objective function, \( g_i(x, t), \quad i = 1, \ldots, m, \) are the infinite and \( h_i(x), \quad i = 1, \ldots, q, \) the finite constraint functions. \( T \subset \mathbb{R}^p \) is, usually, a cartesian product of intervals \( ([\alpha_1, \beta_1] \times [\alpha_2, \beta_2] \times \ldots \times [\alpha_p, \beta_p]) \). Since AMPL does not directly support SIP problems, the following assumptions for coding SIP problems in AMPL were made.

**Assumption 1** Constraint functions which depend on the infinite \((t)\) variables are coded with names starting with \(t\). Conversely, all constraint functions with names starting with \(t\) are assumed to depend on the infinite variables.

**Assumption 2** All infinite variable \((t)\) names must start with \(t\). Conversely, all variables with names starting with \(t\) are assumed to be infinite.
Assumption 3 The AMPL .row and .col files must be provided (option aux-files rc in AMPL).

As an example consider the following problem

\[
\min_{x \in \mathbb{R}^2} x_1^2 + x_2^2 \\
\text{s.t.} \quad x_1 t + x_2 t^2 \leq 0 \\
-10 \leq x_1 + x_2 \leq 10 \\
\forall t \in [0, 1]
\]

The corresponding (SIP) AMPL code is presented below

```
########################################
# Sample problem in the user manual
# aivaz@dps.uminho.pt 27/12/99
########################################

var x {1..2};
# infinite variable name must start with t
var t;

#objective function
minimize fx: x[1]^2+x[2]^2;

# infinite constraint, so name must start with t
subject to tcons:
   x[1]*t+x[2]*t^2 <= 0;
# finite constraint therefore name must not start with t
subject to constraint:

# bounds on t var
# ampl presolver will use this constraint to provide the bounds array
subject to bounds:
   0 <= t <= 1;

########################################
# End of Problem codification #
########################################
```
# do not forget to write .col and .row files
option mysolver_auxfiles rc;
# this problem has no initial guess (starting point)
option reset_initial_guesses 1;
# change solver
option solver mysolver;
# solve problem
solve;

#########################
# Solution found #
#########################
printf "Solution found\n";
display x;
display fx;

mysolver is the SIP solver and may be replaced to meet ones requirement.

4 AMPL

AMPL is an algebraic mathematical programming language that allows the programming of mathematical problems. AMPL provides a way to communicate with a wide variety of solvers. The flexible and natural language used by AMPL was the main reason for choosing AMPL for our development. The extension proposed here could be done in other modeling language environment, namely in CUTE (Constrained and Unconstrained Testing Environment [5]), but the Standard Input Format (SIF) language is not as natural as AMPL’s. AMPL is a commercial software but a student edition is available for evaluation. SIPAMPL can be obtained by the internet and the reader can evaluate this software freely. The SIPAMPL directory structure is shown in the Appendix.

Throughout the paper, we assume that the reader is familiar with AMPL and the C programming language. For a background reading, see [12, 13] and [23]. An outline on how to hook a solver to AMPL is given in the Appendix.

5 Interfacing the solver with AMPL

To build the SIP interface for AMPL the following steps should be taken:

- Get AMPL and the solvers.tar file from the internet (see Appendix for address and for a brief description on how to install). Install AMPL,
uncompress and build amplsolver.a for Linux or amplsolv.lib for MS-Windows.

- Get SIPAMPL (see section 6 for internet address). Uncompress SIPAMPL within solver directory (see Appendix).

- Type make (Linux) or nmake -f makefile.vc (MS-Windows) in SIPAMPL directory. A new file libsip.a (Linux) or sipampl.lib (MS-Windows) should be created.

The files libsip.a or sipampl.lib and sip.h (include file within SIPAMPL) are needed to use SIPAMPL interface. To use SIPAMPL the next steps should be done:

- Include sip.h in solver code.
- Read stub.nl file in the usual AMPL way (see [13]).
- Call sip_init SIPAMPL function to start using SIPAMPL.
- Call SIPAMPL functions as needed.
- call sip_free to release memory allocated by sip_init.

In sip.h a SIP data structure and some variables are defined. Table 1 shows the variables, their meaning and their correspondence in the SIP definition in (1). Note that these variables are macros to the SIP data structure. If one needs another SIP data structure (for example, if one needs two problems in memory) one should create new macros or use the data structure directly.

To describe the SIPAMPL functions the following definitions are made. Since AMPL does not support SIP problems the presolver has no special care with the variables declaration order, so the variables which names start with t and the others are mixed up. The nature of SIP requests these components to be separated (the solver may need to change some variables while others are kept fixed). Let the original variables array be defined by $x_{NLP} = (x_1, x_2, ..., x_{nvar})$. We will call $x_{NLP}$ the original $x$ array of variables, where $nvar$ is an AMPL variable which gives the total number of variables in problem. The SIP $x$ and $t$ components must be obtained from $x_{NLP}$. Let us call the $x$ an $t$ components $x_{SIP}$ and $t_{SIP}$ respectively.

A brief description of the SIPAMPL functions which support the SIP evaluation functions follows. Please note that gradients, Hessians and Jacobians are always in dense format, defined in a FORTRAN way (AMPL dense format).
- **sip_extractx** Extracts the $x_{SIP}$ component from the initial variable $x_{NLP}$.

- **sip_extractt** As in **sip_extractx** but for the $t_{SIP}$ component.

- **sip_joinxt** Joins $x_{SIP}$ and $t_{SIP}$ components from SIP back into $x_{NLP}$ array of variables.

- **sip_init** Initializes the variables for the problem, allocates the arrays for the bounds and copies the bound values to the arrays. This function looks for variable and constraint names in order to keep track of the $x_{SIP}$ and $t_{SIP}$ positions in $x_{NLP}$ (keeping track of the $x$ and $t$ constraints position is also needed in order to be able to compute the original constraint position from an $x$ or $t$ constraint position). AMPL does not provides .col and .row files by default (which are needed to get variable and constraint names) so the user should provide them

  (option mysolver_auxfiles rc; - the AMPL command in example given in section 3).

- **sip_free** Frees the memory allocated during the call to **sip_init**. Only the memory is freed, the variables in the **sip** data structure are not re-initialized.

- **sip_objval** Evaluates the objective function. The objective function depends only on the $x_{SIP}$ variables. The user must also provide the objective function number (AMPL supports multi objective functions). This function calls AMPL objval with $t_{SIP}$ equal to 0.

- **sip_objgrd** Evaluates the objective function gradient vector. As in **sip_objval** only the $x_{SIP}$ is needed. **sip_objgrd** sets the array with the derivatives with respect to $x_{SIP}$ alone (the others are zero, since the objective function does not depend on $t$ variables).

- **sip_objhes** Evaluates the objective function Hessian matrix. Again the objective function depends only on $x_{SIP}$, so the considerations made to **sip_objval** and **sip_objgrd** are also valid.

- **sip_conval** Evaluates the constraints. Constraints depend on $x_{SIP}$ and some on $t_{SIP}$, so both must be provided. **sip_conval** fills two arrays, one with the values of constraints that do not depend on $t$ and the other with the constraints that depend on $t$. **sip_conval** generates $x_{NLP}$ from $x_{SIP}$ and $t_{SIP}$ and calls AMPL conval function. It splits conval result into two arrays.

6
• **sip_jacval** Evaluates the constraint Jacobian. Needs \( x_{SIP} \) and \( t_{SIP} \). Fills two matrices (for \( x \) and \( t \) constraints).

• **sip_conxval** Evaluates an \( x \) constraint. The constraint number must be supplied. The constraint number refers to \( x \) constraints only. Only \( x_{SIP} \) is needed.

• **sip_contval** Same as **sip_conxval** but for \( t \) constraints. \( x_{SIP} \) and \( t_{SIP} \) are needed.

• **sip_conxgrd** Evaluates an \( x \) constraint gradient. Only needs \( x_{SIP} \) and fills an array with the derivatives with respect to \( x_{SIP} \) components.

• **sip_contgrd** Evaluates a \( t \) constraint gradient. Needs \( x_{SIP} \) and \( t_{SIP} \). Fills two arrays as in **sip_conval**.

• **sip_conxhes** Evaluates an \( x \) constraint Hessian matrix. Only \( x_{SIP} \) is needed. Fills one matrix.

• **sip_conthes** Evaluates a \( t \) constraint Hessian matrix. Needs \( x_{SIP} \) and \( t_{SIP} \). Fills three matrices, derivatives w.r.t. \( xx \), w.r.t. \( xt \) and w.r.t. \( tt \).

**sip_extractx**, **sip_extractt** and **sip_joinxt** are mostly used as support functions to the SIP evaluation functions, that make the interface with the AMPL, since when calling AMPL functions for nonlinear programming an \( x_{NLP} \) must be supplied.

The **asl** data structure is also available, so if one needs other information about the problem the functions from the standard nonlinear interface can be used. It is the user responsibility to read the **stub.nl** file and to provide the corresponding **asl** data structure.

In Table 2 the function prototypes are shown. The corresponding function argument descriptions are given in Table 3.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>SIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>int nxsip</td>
<td>( x_{SIP} ) array dimension (number of variables that do not start by t)</td>
<td></td>
</tr>
<tr>
<td>int ntsip</td>
<td>( t_{SIP} ) array dimension (number of variables that start by t)</td>
<td></td>
</tr>
<tr>
<td>int nxsipc</td>
<td>Number of constraints that do not depend on ( t ) (constraints that do not start by t)</td>
<td>( p )</td>
</tr>
<tr>
<td>int ntsipc</td>
<td>Number of constraints that depend on ( t ) (constraints that start by t)</td>
<td>( q )</td>
</tr>
<tr>
<td>real *XBU</td>
<td>Array of upper bounds on ( x_{SIP} ) variables</td>
<td></td>
</tr>
<tr>
<td>real *XBL</td>
<td>Array of lower bounds on ( x_{SIP} ) variables</td>
<td></td>
</tr>
<tr>
<td>real *TBU</td>
<td>Array of upper bounds on ( t_{SIP} ) variables</td>
<td>( \alpha )</td>
</tr>
<tr>
<td>real *TBL</td>
<td>Array of lower bounds on ( t_{SIP} ) variables</td>
<td>( \beta )</td>
</tr>
<tr>
<td>real *XCBU</td>
<td>Array of upper bounds on ( x_{SIP} ) constraints</td>
<td></td>
</tr>
<tr>
<td>real *XCBL</td>
<td>Array of lower bounds on ( x_{SIP} ) constraints</td>
<td></td>
</tr>
<tr>
<td>real *TCBU</td>
<td>Array of upper bounds on ( t_{SIP} ) constraints</td>
<td></td>
</tr>
<tr>
<td>real *TCBL</td>
<td>Array of lower bounds on ( t_{SIP} ) constraints</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Variables defined in \( \text{sip.h} \).
/* Initialize SIP struct and variables */
int sip_init(ASL *a);
/* Extracts x component from original x */
void sip_extractx(real *x, real *X);
/* Extracts t component from original x */
void sip_extractt(real *x, real *T);
/* Joins x and t in original x */
void sip_joinxt(real *X, real *T, real *x);
/* Objective function value */
real sip_objval(int nobj, real *X, fint *nerror);
/* Objective function gradient vector */
void sip_objgrd(int nobj, real *X, real *G, fint *nerror);
/* Objective function hessian matrix */
void sip_objhes(int nobj, real *H, fint *nerror);
/* Vector of constraints values */
void sip_conval(real *X, real *T, real *RX, real *RT,
               fint *nerror);
/* Constraints Jacobian matrix*/
void sip_jacval(real *X, real *T, real *JX, real *JT,
               fint *nerror);
/* x Constraint value */
real sip_conxval(int ncon, real *X, fint *nerror);
/* t Constraint value */
real sip_contval(int ncon, real *X, real *T, fint *nerror);
/* x Constraint gradient vector */
void sip_conxgrd(int ncon, real *X, real *G, fint *nerror);
/* t Constraint gradient vector */
void sip_contgrd(int ncon, real *X, real *T, real *GX,
                 real *GT, fint *nerror);
/* x Constraint hessian matrix */
void sip_conxhes(int ncon, real *H, fint *nerror);
/* t Constraint hessian matrix */
void sip_conthes(int ncon, real *HX, real *HT, real *HXT, fint *nerror);
/* free allocated memory in SIP struct */
void sip_free(void);

Table 2: Function prototypes.
<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
<th>Dimension in SIPAMPL (in (1))</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>$x_{SIP}$ array from SIP</td>
<td>nxsip(n)</td>
</tr>
<tr>
<td>T</td>
<td>$t_{SIP}$ array from SIP</td>
<td>ntsip(p)</td>
</tr>
<tr>
<td>x</td>
<td>$x_{NLP}$ array from the original nonlinear problem</td>
<td></td>
</tr>
<tr>
<td>nobj</td>
<td>Objective number (AMPL allows multi objective functions)</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>Objective or constraint gradient vector</td>
<td>nxsip(n)</td>
</tr>
<tr>
<td>H</td>
<td>Objective or constraint hessian matrix</td>
<td>nxsip x nxsip(n × n)</td>
</tr>
<tr>
<td>RX</td>
<td>x constraint values array</td>
<td>nxsipc(q)</td>
</tr>
<tr>
<td>RT</td>
<td>t constraint values array</td>
<td>ntsipc(m)</td>
</tr>
<tr>
<td>JX</td>
<td>x constraint jacobian matrix</td>
<td>nxsipc x nxsip(q × n)</td>
</tr>
<tr>
<td>JT</td>
<td>t constraint jacobian matrix</td>
<td>ntsipc x ntsip(m × p)</td>
</tr>
<tr>
<td>GX</td>
<td>x constraint gradient vector</td>
<td>nxsipc(q)</td>
</tr>
<tr>
<td>GT</td>
<td>t constraint gradient vector</td>
<td>ntsipc(m)</td>
</tr>
<tr>
<td>HX</td>
<td>x constraint hessian matrix</td>
<td>nxsip x nxsip(n × n)</td>
</tr>
<tr>
<td>HT</td>
<td>t constraint hessian matrix</td>
<td>ntsip x ntsip(p × p)</td>
</tr>
<tr>
<td>HXT</td>
<td>x,t constraint hessian matrix</td>
<td>nxsip x ntsip(n × p)</td>
</tr>
<tr>
<td>ncon</td>
<td>Constraint number in x or t constraints</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(for example if ncon=1 in a call to sip_contval the first t constraint is evaluated, and possible not the first declared constraint)</td>
<td></td>
</tr>
<tr>
<td>nerror</td>
<td>Error variable in AMPL (may be null, if no error control is wanted)</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Function arguments.
6 The test problem database

The problems in the database were obtained from several papers and books on SIP and were coded in SIPAMPL. They comprise linear and non-linear, academic and real life problems. Several problems have parameters that can be changed by the user to generate new problems. Most are of small dimension (less than 50 finite and 10 infinite variables). Table 4 lists the problems in the database. In the table “Problem” is the problem filename (with .mod extension), “nx” is the number of \( x \) variables (\( n \)); “nt” is the number of \( t \) variables (\( p \)); “ncx” is the number of finite constraints (\( q \)) and “ntc” is the number of infinite constraints (\( m \)).

Ten problems are those described by Haaren-Retagne in [18] and ten more optimal signal set design [14] coded as reported in [45]. Sixteen of the problems correspond to the Watson set (see [47] and [7]). Seven other problems, quadratic and nonlinear, are described by Price ([35]). One problem described by Price has an error. In problem S, Price considered \( T \) to be the set \([0,1]^p\), but the solutions presented are in the set \([0,2]^p\). The later set was coded. Three \( C^3 \) problems from Price and Coope ([36]) were coded. Twelve Chebyshev approximation problems have been taken from Hettich [19, 20] and Reemtsen [37]. Two more Chebyshev problems from [21] were coded. In the first one Hettich classifies this problem as a quadratic one, but no quadratic terms in the objective function are seen (problem \textit{hettich8}). In the second one the objective function depends on a set of points that are unknown to us (problem \textit{hettich9}). Three quadratic problems were taken from Liu, Teo and Ito [30] and other linear problem from Lin, Fang and Wu ([29], section 5.2). In this paper the solution to the problem is the one reported in [7], but the problem is not the same (a coefficient in the objective function, the constraint and the variables bounds differ from the original one). The problem was coded as described in [29]. Three linear problems from Fang and Wu [9] and two from Ferris and Philpott [10] were coded. Seventeen linear problems were taken from Leon, Sanmatias and Vercher [27]. Problems \textit{leon13} and \textit{leon18} correspond to problem 1 and 2 in [32] respectively. Two problems from production planning were coded as described by Li and Wang [28] and Wang and Fang [46]. One problem was coded from Tanaka [40] and two more from Polak [33]. Problem \textit{hettich10} was taken from [25]. Example 1 and 2 from [41] were also coded. In fact, example 2 (\textit{teo2}) appeared in a previous paper from Gonzaga et al. [15] and is incorrectly described in [41]. Example 2 appears correctly described in [22, 42]. Three problems from Blankenship and Falk [4] and seven filter design problems proposed by Potchinkov in [34] were also included. In problems \textit{blankenship2} and \textit{blankenship3}, obtained from [4], the set \( T \) is of type \([0, +\infty[\). The problem used by Powell, to show
that the generalized Karmarkar [2] algorithm can converge to a non optimal solution was also codified, as described in [43]. One problem used by Still [39] to show the importance of including the boundary points in a discretization of the infinite set $T$ and a problem proposed by Anderson and Lewis in [3] were also coded. In [16] the author proposes 24 SIP problems. Since 17 were already in the database, the remaining 7 were coded. Seventeen problems from [17] and one from [48] were coded. In [48] the authors present a corrected version of problem hettich10 that was coded as hettich10c. Four problems from [24] and one from [26] were coded. The two MATLAB examples from the user manual were also coded.

A total of one hundred and thirty nine problems were coded. Starting points and solutions, when available from the authors, were also included in the database. The database, as well as SIPAMPL can be obtained via the internet. We trust this development will be of aid to researchers benchmarking SIP algorithms and readers can propose problems to this database by sending them to the first author (see email address in first page).

<table>
<thead>
<tr>
<th>Problem</th>
<th>nx</th>
<th>nt</th>
<th>nrx</th>
<th>nrt</th>
<th>Problem</th>
<th>nx</th>
<th>nt</th>
<th>nrx</th>
<th>nrt</th>
</tr>
</thead>
<tbody>
<tr>
<td>anderson1</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>blankenshi1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>blankenshi2</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>blankenshi3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>coopeL</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>coopeM</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>coopeN</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>elke1</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>elke2</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>10</td>
<td>elke3</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>elke4</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>elke5</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>elke6</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>10</td>
<td>elke7</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>elke8</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>elke9</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>elke10</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>elke1std</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>19</td>
</tr>
<tr>
<td>elke2std</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>19</td>
<td>elke3std</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>19</td>
</tr>
<tr>
<td>elke4std</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>elke5std</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>19</td>
</tr>
<tr>
<td>elke6std</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>19</td>
<td>elke7std</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>19</td>
</tr>
<tr>
<td>fang1</td>
<td>50</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>fang2</td>
<td>50</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>fang3</td>
<td>50</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>ferrals1</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>ferrals2</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>gockenbach1</td>
<td>33</td>
<td>1</td>
<td>120</td>
<td>16</td>
</tr>
<tr>
<td>gockenbach2</td>
<td>33</td>
<td>1</td>
<td>120</td>
<td>16</td>
<td>gockenbach3</td>
<td>33</td>
<td>1</td>
<td>120</td>
<td>16</td>
</tr>
<tr>
<td>gockenbach4</td>
<td>33</td>
<td>1</td>
<td>120</td>
<td>16</td>
<td>gockenbach5</td>
<td>33</td>
<td>1</td>
<td>120</td>
<td>16</td>
</tr>
<tr>
<td>gockenbach6</td>
<td>33</td>
<td>1</td>
<td>120</td>
<td>16</td>
<td>gockenbach7</td>
<td>33</td>
<td>1</td>
<td>120</td>
<td>16</td>
</tr>
<tr>
<td>gockenbach8</td>
<td>33</td>
<td>1</td>
<td>120</td>
<td>16</td>
<td>gockenbach9</td>
<td>33</td>
<td>1</td>
<td>120</td>
<td>16</td>
</tr>
<tr>
<td>gockenbach10</td>
<td>33</td>
<td>1</td>
<td>120</td>
<td>16</td>
<td>goerner1</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>goerner2</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>goerner3</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Continues ...

\(^1\)http://www.ing.uminho.pt/~dps/aivaz/
<table>
<thead>
<tr>
<th>Problem</th>
<th>nx</th>
<th>nt</th>
<th>nr</th>
<th>nrt</th>
<th>Problem</th>
<th>nx</th>
<th>nt</th>
<th>nr</th>
<th>nrt</th>
</tr>
</thead>
<tbody>
<tr>
<td>goerner4</td>
<td>7</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>goerner5</td>
<td>7</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>goerner6</td>
<td>16</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>goerner7</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>gugat1</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>gugat2</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>gugat3</td>
<td>7</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>gugat4a</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>gugat4b</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>gugat4c</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>gugat4d</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>gugat4e</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>gugat4f</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>gugat5a</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>gugat5b</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>gugat5c</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>gugat5d</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>gugat5e</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>gugat5f</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>gugat6</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>gugat7</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>hettich1</td>
<td>9</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>hettich2</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>hettich3</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>hettich4</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>hettich5</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>hettich6</td>
<td>7</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>hettich7</td>
<td>7</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>hettich8</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>hettich9</td>
<td>11</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>hettich10</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>hettich10c</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>kortonek1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>kortonek2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>kortonek3</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>kortonek4</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>leon1</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>leon2</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>leon3</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>leon4</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>leon5</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>leon6</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>leon7</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>leon8</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>leon9</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>leon10</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>leon11</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>leon12</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>leon13</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>leon14</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>leon15</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>leon16</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>leon17</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>leon18</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>leon19</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>li1</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>li2</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>lin1</td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>liu1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>liu2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>liu3</td>
<td>16</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>matlab1</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>matlab2</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>polak1</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>polak2</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>potchinkov1</td>
<td>298</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>potchinkov2</td>
<td>65</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>potchinkov3</td>
<td>66</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>potchinkov4a</td>
<td>67</td>
<td>1</td>
<td>19</td>
<td>2</td>
<td>potchinkov4b</td>
<td>65</td>
<td>1</td>
<td>19</td>
<td>1</td>
</tr>
<tr>
<td>potchinkovPL</td>
<td>122</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>potchinkovPLR</td>
<td>122</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>powell1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>priceK</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>priceS3</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>priceS4</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>priceS5</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>priceS6</td>
<td>4</td>
<td>6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>priceT</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>priceU</td>
<td>4</td>
<td>6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>reemtsen1</td>
<td>11</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>reemtsen2</td>
<td>10</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Continues …
7 Interfacing SIPAMPL with an existing solver

In this section a description of an interface between MATLAB [1] and SIPAMPL is given. This interface allows problems coded in SIPAMPL to be solved with MATLAB. MATLAB in its Optimization Toolbox ([6]) provides an Optimization Function (fseminf) for semi-infinite programming. The range of problems that can be solved by MATLAB is limited since the constraints are limited to two infinite variables in each infinite constraint. The interface to MATLAB optimization toolbox is limited to two infinite variables, since problem formulation presented herein supposes that all the infinite variables appear in all the infinite constraints.

MATLAB sipampl syntax is:

\[ [x0,ntc,xbl,xbu] = \text{sipampl('userman')} \]
\[ f = \text{sipampl}(x) \]
\[ [f,Grad] = \text{sipampl}(x) \]
\[ [c,ceq,K1,...,Kntc,s] = \text{sipampl}(x,s) \]
\[ \text{sipampl('msg',x)} \]

The behaviour of sipampl MATLAB function depends on the number of input and output arguments, so:
• \([x0,ntc,xbl,xbu] = \text{sipampl('userman')}\) reads the problem named
  stub (stub.n1 file) and returns \(x0\) the initial guess, \(ntc\) the number of
  infinite constraints, and the lower and upper bounds on the \(x\) variables
  \(xbl\) and \(xbu\), respectively.

• \(f = \text{sipampl}(x)\) returns the objective function value at \(x\).

• \([f,\text{Grad}] = \text{sipampl}(x)\) returns the objective value and the gradient
  vector at \(x\).

• \([c,\text{ceq},K1,\ldots,Kn\text{tc},s] = \text{sipampl}(x,s)\) evaluates the constraints
  at \(x\). \(s\) is the step size for the grid where the infinite constraints are
  evaluated (see [6] for more details). \(c\) and \(\text{ceq}\) are the vectors with the
  values of the finite constraints, \(c\) for inequality constraints and \(\text{ceq}\) for
  equality constraints. The \(K1,\ldots,Kn\text{tc}\) are \(ntc\) vectors (or matrices) with
  the infinite constraints evaluated at the grid.

• \(\text{sipampl('msg',x)}\) writes the ampl solution \(x\) with the \text{msg} text mes-
  sage.

A simple example using these functions is:

\[
\begin{align*}
\text{>> [x0,ntc,xbl,xbu] = sipampl('userman');} \\
\text{>> options = optimset('GradObj', 'on');} \\
\text{>> x = fseminf('sipampl',x0,ntc,'sipampl',[],[],[],[],...} \\
\text{\quad xbl,xbu,options);} \\
\text{>> sipampl('Solution found by MATLAB',x);} \\
\end{align*}
\]

The interface developed here includes two files. \text{sipampl.c} is the main
program which provides the MEX ([31]) function. \text{sipsolve.m} is a MATLAB
file with a short example of how to use MATLAB to solve problems codified
with SIPAMPL.

These files are also available by the internet with the problem database
and SIPAMPL (see section 6). A step by step installation follows:

• The MATLAB interface is provided with SIPAMPL (see section 6).

• Change to MATLAB directory (inside SIPAMPL directory).

• Type \text{make} (Linux) or \text{mmake} \(-f \text{makefile.vc}\) (MS-Windows). \(n)\text{make}
calls the \text{mex} compiler from MATLAB.

• A MATLAB executable is built.
• Place the MATLAB executable in a directory such that MATLAB is able to find it (MATLAB path).

• Call AMPL with a problem and build stub.nl, stub.row and stub.col files.

• Call MATLAB Optimization toolbox as described previously.

8 A new approach to SIPAMPL interface with MATLAB

The new approach consists of exporting all the SIPAMPL interface routines directly to MATLAB. The new MATLAB functions are:

• sip_init initializes the use of the SIPAMPL interface routines;

• sip_end cleans memory and writes a solution file for AMPL;

• sip_objval returns the objective value;

• sip_objgrd returns the objective gradient;

• sip_objhes returns the objective Hessian at the last value used in calls to objective or constraints functions;

• sip_conval returns the constraints value;

• sip_contval returns an infinite constraint value;

• sip_contgrd returns an infinite constraint gradient;

• sip_conthes returns an infinite constraint Hessian at the last value used in calls to objective or constraints functions;

• sip_conxeqval returns an equality finite constraint value;

• sip_conxeqgrd returns an equality finite constraint gradient;

• sip_conxeqhes returns an equality finite constraint Hessian at the last value used in calls to objective or constraints functions;

• sip_conxineqval returns an inequality finite constraint value;

• sip_conxineqgrd returns an inequality finite constraint gradient;
• **sip_conxineqhes** returns an inequality finite constraint Hessian at the last value used in calls to objective or constraints functions;

• **sip_jacval** returns the Jacobian;

• **sip_usage** prints the **sip_*** usage. Used by other functions when reporting an invalid number of arguments.

The number of arguments in each function can be consulted by issuing the **sip_usage** function. To get further help about each function the MATLAB help command can be used.

All functions use the MEX file **sipamp12**. This MEX file is provided in source (C programming language) and was tested in Linux and Windows platforms with MATLAB version 6.1, release 12.1.

The syntax of the new MATLAB functions is

• **[x, xbl, xbu, tbl, tbu]=sip_init(stub);**
  where **stub** is a string with the filename of the .nl file to be solved, **x** is the initial guess, **xbl** is the lower bound array of the finite variables, **xbu** is the upper bound array of the finite variables, **tbl** is the lower bound array of the infinite variables and **tbu** is the upper bound array of the infinite variables.

• **sip_end(str,x);**
  where **str** is a string with the message written to the AMPL .sol file and **x** is the solution found.

• **[f]=sip_objval(x);**
  where **x** is an array of finite variables and **f** is the objective value at **x**.

• **[g]=sip_objgrd(x);**
  where **x** is an array of finite variables and **g** is the objective gradient at **x**.

• **[h]=sip_objhes();**
  where **h** is the objective Hessian at **x**.

• **[c, ceq, inf]=sip_conval(x,t);**
  where **x** is an array of finite variables, **t** is an array of infinite variables, **c** is the array of inequality constraints evaluated at **x**, **ceq** is the array of equality constraints evaluated at **x** and **inf** is the array of infinite constraints evaluated at **x,t**.
• \([\text{inf}] = \text{sip\_contval}(n, x, t)\);  
  where \(n\) is a constraint number, \(x\) is an array of finite variables, \(t\) is an 
  array of infinite variables and \(\text{inf}\) is the value of the infinite constraint 
  \(n\) evaluated at \(x, t\).

• \([\text{gx}, \text{gt}] = \text{sip\_contgrad}(n, x, t)\);  
  where \(n\) is a constraint number, \(x\) is an array of finite variables, \(t\) is an 
  array of infinite variables, \(\text{gx}\) is the gradient, w.r.t. \(x\), of the infinite 
  constraint \(n\) computed at \(x, t\) and \(\text{gt}\) is the gradient, w.r.t. \(t\), of infinite 
  constraint \(n\) computed at \(x, t\).

• \([\text{hx}, \text{ht}, \text{hxt}] = \text{sip\_conthees}(n)\);  
  where \(n\) is a constraint number, \(\text{hx}\) is the Hessian, w.r.t. \(xx\), of the 
  infinite constraint \(n\) computed at \(x, t\), \(\text{ht}\) is the Hessian, w.r.t. \(tt\), of 
  the infinite constraint \(n\) computed at \(x, t\) and \(\text{hxt}\) is the Hessian, w.r.t. 
  \(xt\), of the infinite constraint \(n\) computed at \(x, t\).

• \([\text{ceq}] = \text{sip\_conseqval}(n, x)\);  
  where \(n\) is a constraint number, \(x\) is an array of finite variables and \(\text{ceq}\) 
  is the value of equality finite constraint \(n\) evaluated at \(x\).

• \([\text{geq}] = \text{sip\_conseqgrad}(n, x)\);  
  where \(n\) is a constraint number, \(x\) is an array of finite variables and \(\text{geq}\) 
  is the gradient of the equality finite constraint \(n\) computed at \(x\).

• \([\text{heq}] = \text{sip\_conseqhes}(n)\);  
  where \(n\) is a constraint number, \(\text{heq}\) is the Hessian of equality finite 
  constraint \(n\) computed at \(x\);

• \([\text{c}] = \text{sip\_conseqval}(n, x)\);  
  where \(n\) is a constraint number, \(x\) is an array of finite variables and \(\text{c}\) 
  is the value of inequality finite constraint \(n\) evaluated at \(x\).

• \([\text{g}] = \text{sip\_conseqgrad}(n, x)\);  
  where \(n\) is a constraint number, \(x\) is an array of finite variables and \(\text{g}\) 
  is the gradient of the inequality finite constraint \(n\) computed at \(x\);

• \([\text{h}] = \text{sip\_conseqhes}(n)\);  
  where \(n\) is a constraint number, \(\text{h}\) is the Hessian of the inequality finite 
  constraint \(n\) computed at \(x\).
• \([jacc, jacceq, jactinf, jacxinf]=\text{sip}_\text{jacval}(x,t)\);

\(x\) is an array of finite variables, \(t\) is an array of infinite variables, \(jacc\) is the Jacobian of the inequality finite constraints (number of inequality constraints \(\times\) number of finite variables), \(jacceq\) is the Jacobian of the equality finite constraints (number of equality constraints \(\times\) number of finite variables), \(jactinf\) is the Jacobian w.r.t. \(t\) of the infinite constraints (number of infinite constraints \(\times\) number of infinite variables) and \(jacxinf\) is the Jacobian w.r.t. \(x\) of infinite constraints (number of infinite constraints \(\times\) number of infinite variables);

The use of the SIPAMPL MATLAB interface to solve a problem is done in a similar way as to solve a MATLAB coded problem. We will use the one-dimensional MATLAB problem

\[
\begin{align*}
\min_{x \in \mathbb{R}^3} & \ f(x) = (x_1 - 0.5)^2 + (x_2 - 0.5)^2 + (x_3 - 0.5)^2 \\
\text{s.t.} & \ \sin(t_1x_1)\cos(t_1x_2) - \frac{1}{1000}(t_1 - 50)^2 - \sin(t_1x_3) - x_3 - 1 \leq 0, \\
& \ \sin(t_1x_2)\cos(t_1x_1) - \frac{1}{1000}(t_1 - 50)^2 - \sin(t_1x_3) - x_3 - 1 \leq 0, \\
& \ \forall t_1 \in [1,100].
\end{align*}
\]

as example. We start by writing a M-File for the objective function. Let \(\text{mysipfun.m}\) be such a file with the following content.

```matlab
function [f,g]=mysipfun(x,s)

if nargin < 1 | nargout<1
    error('Invalid number of arguments');
end

f=sip_objval(x);

if nargin > 1
    g=sip_objgrd(x);
end
```

We write an M-File for the constraints. Let \(\text{mysipcon.m}\) be such a file with the following content.

```matlab
function [c,ceq,K1,K2,s]=mysipcon(x,s)
```

19
if nargin < 2 | nargout<5
    error('Invalid number of arguments');
end

if isnan(s(1,1)),
    s=[0.2 0; 0.2 0];
end

w1=1:s(1,1):100; w2=1:s(2,1):100;

lw1=length(w1); lw2=length(w2); K1=zeros(lw1,1); K2=zeros(lw2,1);

for i=1:lw1
    K1(i)=sip_contval(0,x,w1(i));
end

for i=1:lw2
    K2(i)=sip_contval(1,x,w2(i));
end

c=[]; ceq=[];

plot(w1,K1,'-',w2,K2,':'), title('Semi-infinite constraints')
drawnow

After writing both file and using AMPL to produce the matlab1.nl, matlab1.col and matlab1.row we can use MATLAB in the following way to solve the problem

>> [x0, xbl, xbu, tbl, tbu]=sip_init('matlab1');
>> [x, fval]=fseminf('mysipfun',x0,2,'mysipcon')
Optimization terminated successfully:
    Search direction less than 2*options.TolX and
    maximum constraint violation is less than options.TolCon
Active Constraints:
    7
    10
\[ x = \]

\[
0.6673 \\
0.3013 \\
0.4023 \\
\]

\[ fval = \]

\[
0.0770 \\
\]

which produces the same solution as in the MATLAB optimization toolbox manual.

\texttt{mysipcon} will produce a graphic in each call. Figure 1 presents the infinite constraints in the solution found.

To run all SIPAMPL problems from a dimensional independently way we have written three M-files. \texttt{sip\_fun.m} and \texttt{sip\_con.m} are the M-files that compute the objective function and constraints values, respectively. \texttt{sip\_solve.m} does all the interface to the SIPAMPL routines and calls \texttt{fseminf} to solve the problem given as the only requested argument. \texttt{sip\_solve} does not return any arguments, but adds a line to the \texttt{results} file in a \LaTeX\ syntax and prints the solution found. The use of \texttt{sip\_solve} to solve \texttt{matlab1} and \texttt{matlab2} problems is shown bellow.

\[
\texttt{>> sip\_solve('matlab1')} \\
\text{Optimization terminated successfully:} \\
\text{Search direction less than 2*options.TolX and} \\
\text{maximum constraint violation is less than options.TolCon} \\
\text{Active Constraints:} \\
\quad 7 \\
\quad 10 \\
\text{Solution:} \\
\quad 0.6673 \\
\quad 0.3013 \\
\quad 0.4023 \\
\text{Objective value:} \\
\quad 0.0770 \\
\]
Figure 1: Plot of the infinite constraints in one-dimensional MATLAB problem, in the solution found
>> sip_solve('matlab2')
Optimization terminated successfully:
  Magnitude of directional derivative in search direction
   less than 2*options.TolFun and maximum constraint violation
   is less than options.TolCon
Active Constraints:
   18

Solution:
   0.2926
   0.1874
   0.2202

Objective value:
   0.0091

>>

   sip_con.m should be edit to set the initial sampling interval. For problem
matlab1 we have used s=[0.2 0 0.2 0] and for matlab2 s=[2 2]. The file
results contains the following two lines.

matlab1 & 8 & 41 & 0.077014\\
matlab2 & 9 & 47 & 0.009132\\

   We have changed the select tool from SIPAMPL in order to write a
M-file that solves all the selected problems.

   To produce the .nl, .col and .row files the select tool must be used in
expert mode (see Section 9.3).

9 The select tool

In linear/nonlinear semi-infinite programming, as in finite programming, the
algorithms developed are sometimes limited or appropriate to specific prob-
lem structure. For example, to solve a quadratic problems one should use (to
get the best performance) an algorithm suited for quadratic programming.
As SIPAMPL is a generic database, we may want to select some problems
with specific characteristics from all the database problems. The select tool
allows this selection based on the characteristics printed in Table 5. In table:
“Option” is the name of the option which can be changed; “type” is the type
of data the *select* tool is waiting for. In *list* the *select* tool will present the
allowed values in a list and waits for a selection. In *range* the *select* tool
will ask for two values, one for the lower bound and the other for the upper
bound; “allowed values” are the allowed values for the selected option; “de-
fault” are the default values for the option and “SIP” is the terminology in
the problem definition (1). The next subsection will present the installation
instructions. Subsection 9.2 is devoted to the implementation details. Sub-
section 9.3 describes the expert mode. Subsection 9.4 gives a session example
of the *select* tool.

9.1 Installing the *select* tool

At this moment the *select* tool is available for Linux and Microsoft Windows
operating systems and makes some operating systems calls (to read database
directory, invoke AMPL binary, etc) and so portability is compromised (but
porting to other operating system should be easy).

The *select* tool is provided in only two files: select.h and select.c. A
makefile is also provided and to produce the *select* binary one just have
to type make in the *select* directory. The makefile.vc is intended for the
Microsoft Visual C/C++ compiler and the make -f makefile.vc should
be used to produce the MS-Windows version. The *select* tool will behave
differently, depending on the operating system, when writing a script to run
all the selected problems.

9.2 Implementation details

The *select* tool trusts the information about the objective and constraints
type coded in the problem by the user. The commented lines

    # Objective: Quadratic
    # Constraints: Linear

in the example shown (Section 3) are very important since they provide
the only information available for the *select* tool about the objective and
constraints type.

Recall that $x_1 e^t \leq 0$ is a linear constraint in SIP, but AMPL will consider
it as nonlinear because of the $e^t$ factor.

*select* will extract the remaining information from the .mod file in the
database directory. To call the SIPAMPL interface the *select* tool needs the
.nl, .col and .row files. To produce them *select* copies the .mod file to a
temporary file and replaces the solve; AMPL command with
<table>
<thead>
<tr>
<th>Option</th>
<th>type</th>
<th>allowed values</th>
<th>default</th>
<th>SIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective type</td>
<td>list</td>
<td>Linear</td>
<td>All type</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Quadratic</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Polynomial</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Generic</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>All type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constraints type</td>
<td>list</td>
<td>Linear</td>
<td>All type</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Quadratic</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Polynomial</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Generic</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>All type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number finite variables</td>
<td>range</td>
<td>Nonnegative integers</td>
<td>$[0, +\infty]$</td>
<td>$n$</td>
</tr>
<tr>
<td>Number infinite variables</td>
<td>range</td>
<td>Nonnegative integers</td>
<td>$[0, +\infty]$</td>
<td>$p$</td>
</tr>
<tr>
<td>Number finite constraints</td>
<td>range</td>
<td>Nonnegative integers</td>
<td>$[0, +\infty]$</td>
<td>$q$</td>
</tr>
<tr>
<td>Number infinite constraints</td>
<td>range</td>
<td>Nonnegative integers</td>
<td>$[0, +\infty]$</td>
<td>$m$</td>
</tr>
<tr>
<td>Limits finite variables</td>
<td>list</td>
<td>Both limits finite</td>
<td>Everything</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lower limits finite</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Upper limits finite</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>None limits finite</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Everything</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Limits infinite variables</td>
<td>list</td>
<td>Both limits finite</td>
<td>Everything</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lower limits finite</td>
<td>$[\alpha, \beta]$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Upper limits finite</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>None limits finite</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Everything</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initial guess</td>
<td>list</td>
<td>With initial guess</td>
<td>With or without initial guess</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Without initial guess</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>With or without initial guess</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Questionable problems characteristics
option auxfiles rc; write gtmpfile;

The amount of time spent for processing a file in the database is due to the amount of time spent in writing this temporary file and the processing time used by the AMPL binary to obtain the .nl, .col and .row files. This drawback is compensated with the unneeded work of the user in the codification process and it is not subject to mistakes (in counting variables, etc).

The interconnections for the select tool can be seen in the diagram of Figure 2.

9.3 Expert mode in the select tool

When the select tool, looks for the requested characteristics, provided by the user for a given problem, it generates a temporary file and calls the ampl binary to provide the .nl, .col and .row temporary files. In the default usage the select tool automatically removes these temporary files. In expert mode (use select -x) the select tool asks the user to provide an additional directory (the deposit directory) to place the temporary files and the select tool then only removes the unmatched files.
9.4 A session example

To show how a user can select a problem from the SIPAMPL database we present an interactive session with the select tool.

C:\AMPL\SOLVERS\sip\tools> is the machine prompt. The user starts by setting the environment AMPLFUNC variable to load the bspline.dll dynamic library (needed to the robotics problems in the SIPAMPL database) and then invoking the select tool in expert mode with the .\select -x command. The default database directory and AMPL binary are accepted as being correct. A quadratic objective function with only one infinite variable is to be selected. No finite simple bounds on the finite variables is requested. After the select tool having found the problems in the database, that match the user options, the file select.res is written with the names of the problems. Under Linux a file named select.run, which is a bash shell script to run AMPL with all the found problems, can be written. select.run will have user, group and other execution permissions (mode -rwxr-xr-x). Under MS-Windows a batch script select.bat replaces the select.run option.
C:\AMPL\SOLVERS\sip\tools>set AMPLFUNC=..\nsips\bspline.dll

C:\AMPL\SOLVERS\sip\tools>.\select -x
Select v2.0 tool for SIPAMPL

Expert mode enabled

Default database directory: ..\sipmod
New database directory [CR=Accept default]:
Using database directory: ..\sipmod

Default deposit directory: ..\sipnl
New deposit directory [CR=Accept default]:
Using deposit directory: ..\sipnl

Full path for AMPL binary: ..\AMPL\exe
New full path for AMPL binary [CR=Accept default]:
Using ..\AMPL\exe when executing AMPL

Specified Options:
1) Objective Type : All type
2) Constraints Type : All type
3) 0 <= Number of finite variables <= +Infinity
4) 0 <= Number of infinite variables <= +Infinity
5) 0 <= Number of finite constraints <= +Infinity
6) 0 <= Number of infinite constraints <= +Infinity
7) Finite variables: Everything
8) Infinite variables: Everything
9) Initial guess: With or without initial guess

Enter option number to change option [CR=End]: 1
Processing option 1
   New objective type
   1) Linear
   2) Quadratic
   3) Polynomial
   4) Generic
   5) All type
Option: 2

Specified Options:
1) Objective Type : Quadratic
2) Constraints Type : All type
3) 0 <= Number of finite variables <= +Infinity
4) 0 <= Number of infinite variables <= +Infinity
5) 0 <= Number of finite constraints <= +Infinity
6) 0 <= Number of infinite constraints <= +Infinity
7) Finite variables: Everything
8) Infinite variables: Everything
9) Initial guess: With or without initial guess

Enter option number to change option [CR=End]: 4
Processing option 4
New number of infinite variables
Lower bound [CR=keep value]:
Upper bound [CR=keep value, INF=+Infinity]: 1

Specified Options:
1) Objective Type : Quadratic
2) Constraints Type : All type
3) 0 <= Number of finite variables <= +Infinity
4) 0 <= Number of infinite variables <= 1
5) 0 <= Number of finite constraints <= +Infinity
6) 0 <= Number of infinite constraints <= +Infinity
7) Finite variables: Everything
8) Infinite variables: Everything
9) Initial guess: With or without initial guess

Enter option number to change option [CR=End]: 7
Processing option 7
  Finite variables simple bounds
  1) Both limits finite
  2) Lower limit finite
  3) Upper limit finite
  4) None limit finite
  5) Everything
Option: 4

Specified Options:
1) Objective Type : Quadratic
2) Constraints Type : All type
3) 0 <= Number of finite variables <= +Infinity
4) 0 <= Number of infinite variables <= 1
5) 0 <= Number of finite constraints <= +Infinity
6) 0 <= Number of infinite constraints <= +Infinity

7) Finite variables: None limits finite
8) Infinite variables: Everything
9) Initial guess: With or without initial guess

Enter option number to change option [CR=End]:

21 file(s) found with specified options

Do you want me to:
1) Save results to file select.res
2) Save results to a batch file select.bat
3) Save results to a M-file sip_run.m
4) Print results to stdout
5) Just quit
Option: 1

21 file(s) found with specified options

Do you want me to:
1) Save results to file select.res
2) Save results to a batch file select.bat
3) Save results to a M-file sip_run.m
4) Print results to stdout
5) Just quit
Option: 5

C:\AMPL\SOLVERS\sip\tools>
10 Changing the solver name in the database problems

In each problem file the solver is selected by issuing the AMPL command
\texttt{option solver nsips;}. If a user wishes to write his own solver, then one
of the two following actions must be taken:

- the user solver must be called \texttt{nsips} or renamed \texttt{nsips}.
- the user must edit all the files to change \texttt{nsips} for his own solver name.

To abbreviate the second action we give some \texttt{bash} shell commands to do
this in an automatic form

\begin{verbatim}
for i in whatever files you want to change solver
do
  sed s/nsips/newsolver/ < $i > $i.new
done
\end{verbatim}

These commands will read the files \texttt{whatever files you want to change solver}
and will write the new files \texttt{whatever.new files.new you.new want.new}
to \texttt{new change.new solver.new}.

Note that the \texttt{sed} command will replace every word \texttt{nsips} by \texttt{newsolver}.
If the word \texttt{newsolver} exists in the file you may experience some trouble in
changing again the solver name and so the name of any solver should be a
reserved word.

11 Discussion

This is not a suitable tool to assess the execution time of algorithms.

The matrices returned by the SIPAMPL functions are always in dense
format. There are no known problems in SIP which can be considered of
large dimension that would result in large sparse matrices.

A linear constraint in SIP is of the form

$$x^T a(t) - b(t) \leq 0, \forall t \in T,$$

where $a(t)$ and $b(t)$ are functions of $t$ alone. Since $a(t)$ and $b(t)$ may be
nonlinear in $t$ the AMPL will not recognize such a constraint as a linear
constraint for SIP.
12 Conclusions and future work

SIPAMPL extends the AMPL features by allowing the coding of semi-infinite programming problems. An interface is also provided so that users can easily connect their own code to the problems database. SIPAMPL takes advantage of the AMPL features, including automatic differentiation.

Meanwhile we have developed the NSIPS [44] solver that uses the SIPAMPL to obtain the SIP test problems.

A AMPL step by step installation

The following steps should be made to connect a solver to AMPL.

- Obtain a copy of the solvers.tar\(^2\) file.

- Uncompress and build the \texttt{amplsolver.a} (Linux) or \texttt{amplsolv.lib} (MS-Windows) library. This library provides AMPL functions. The file \texttt{as1.h} has important data structure definitions, being the \texttt{as1} data structure the mostly used (it contains the problem data).

- Build a solver and link it with the AMPL library.

AMPL communicates with the solver by a file with .nl extension (\texttt{stub.nl}). In Figure 3 a diagram of the interaction between AMPL and a solver is shown. AMPL reads a problem description (from a file or from standard input) and writes a file with some information about the problem (\texttt{stub.nl}). The solver reads the \texttt{stub.nl} file into memory. Most of the problem is kept in memory in an \texttt{as1} data structure. This structure provides most of the problem data (initial guess, dual initial guess, number of objective functions, expression for the objectives, number of constraints, constraints expression, etc). After finding a solution the solver writes a \texttt{stub.sol} file with the solution found. AMPL reads \texttt{stub.sol} and can then display the contents of the variables in the problem, on user request.

Some problem data is not available in the \texttt{stub.nl} file (for example the variable and constraint names are not provided). If extra information is needed some additional files should be provided.

B SIPAMPL directory structure

The SIPAMPL directories are organized as described in Figure 4. The directories have the following meaning:

\(^2\)http://netlib.bell-labs.com/netlib/ampl/
Figure 3: Interaction between AMPL and the Solver.

Figure 4: Organization of SIPAMPL directories

- **solvers** directory refers to the directory where the file solvers.tar is uncompressed.

- The **sip** directory contains the SIPAMPL interface (where the libsip.a or sipampl.lib is placed).

- The directory **matlab** contains the MATLAB interface to the SIPAMPL subroutines and an M-file that is a sample on how to use the mex file routines. **matlab2** contains the new interface and the corresponding M-files.

- The **nsips** directory contains the B-Splines dynamic library to use with some problems and the NSIPS solver [44].

- The **s_solver** has an example on how to use the SIPAMPL interface (it calls the SIPAMPL routines and prints the returned values). The user can place here his own solver.
• The \texttt{sipmod} directory is where the problems database is located.

The user can change the directory structure as long as the makefiles and files inclusion are changed accordingly.
References


