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* and
M. Paula S.F. Gomes†

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 `select` tool with new features, compiles with Microsoft Visual C/C++ and has a new MATLAB interface.

*Departamento de Produção e Sistemas, Escola de Engenharia, Universidade do Minho Campus de Gualtar, 4710 Braga, Portugal; Email: {aivaz,emgpf}@dps.uminho.pt

†Mechanical Engineering Department, Mechatronics in Medicine Laboratory, Imperial College of Science, Technology and Medicine, London SW7 2BX, UK; Email:p.gomes@ic.ac.uk

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:

$$\begin{aligned} & \min_{x \in R^n} \ f(x) \\ s.t. \ \ & g_i(x,t) \leq 0, \ \ i = 1, ..., m \\ & h_i(x) \leq 0, \ \ i = 1, ..., o \\ & h_i(x) = 0, \ \ i = o+1, ..., q \\ & \forall t \in T \ , \end{aligned} \tag{1}$$

where $f(x)$ is the objective function, $g_i(x,t), \quad i = 1, ..., m$, are the infinite and $h_i(x), \quad i = 1, ..., q$, the finite constraint functions. $T \subset R^p$ is, usually, a cartesian product of intervals $([\alpha_1, \beta_1] \times [\alpha_2, \beta_2] \times ... \times [\alpha_p, \beta_p])$. 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

$$\begin{aligned} \min_{x \in R^2} \ & x_1^2 + x_2^2 \\ s.t. \quad & x_1 t + x_2 t^2 \leq 0 \\ -10 \leq & x_1 + x_2 \leq 10 \\ & \forall t \in [0,1] \end{aligned}$$

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:
  -10  <= x[1]+x[2] <=  10;

# 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 $x_{NLP} = (x_1, x_2, ..., x_{\mathbf{nvar}})$. 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 $\mathbf{0}$.

- `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.

- `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.

Variable	Description	SIP
int nxsip	x_{SIP} array dimension (number of variables that do not start by t)	n
int ntsip	t_{SIP} array dimension (number of variables that start by t)	p
int nxsipc	Number of constraints that do not depend on t (constraints that do not start by t)	q
int ntsipc	Number of constraints that depend on t (constraints that start by t)	m
real *XBU	Array of upper bounds on x_{SIP} variables	
real *XBL	Array of lower bounds on x_{SIP} variables	
real *TBU	Array of upper bounds on t_{SIP} variables	α_j
real *TBL	Array of lower bounds on t_{SIP} variables	β_j
real *XCBU	Array of upper bounds on x_{SIP} constraints	
real *XCBL	Array of lower bounds on x_{SIP} constraints	
real *TCBU	Array of upper bounds on t_{SIP} constraints	
real *TCBL	Array of lower bounds on t_{SIP} constraints	

Table 1: Variables defined in `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.

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

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); "nxc" 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^1 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 *hettich8*). In the second one the objective function depends on a set of points that are unknown to us (problem *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 *leon13* and *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 *hettich10* was taken from [25]. Example 1 and 2 from [41] were also coded. In fact, example 2 (*teo2*) appeared in a previous paper from Gonzaga *et al.* [15] and is uncorrectly 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 *blankenship2* and *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 correct 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[1]. 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).

Problem	nx	nt	nrx	nrt	Problem	nx	nt	nrx	nrt
andreson1	3	2	0	1	blankenship1	2	1	0	1
blankenship2	2	4	0	4	blankenship3	3	2	3	3
coopeL	2	1	0	1	coopeM	2	1	1	1
coopeN	2	1	0	1	elke1	9	1	0	10
elke2	9	1	0	10	elke3	9	1	0	10
elke4	9	1	0	4	elke5	9	1	0	10
elke6	9	1	0	10	elke7	9	1	0	10
elke8	9	1	0	7	elke9	9	1	0	7
elke10	9	1	0	7	elke1std	9	1	0	19
elke2std	9	1	0	19	elke3std	9	1	0	19
elke4std	9	1	0	7	elke5std	9	1	0	19
elke6std	9	1	0	19	elke7std	9	1	0	19
fang1	50	1	0	1	fang2	50	1	0	1
fang3	50	1	0	1	ferris1	7	1	0	2
ferris2	7	1	0	1	gockenbach1	33	1	120	16
gockenbach2	33	1	120	16	gockenbach3	33	1	120	16
gockenbach4	33	1	120	16	gockenbach5	33	1	120	16
gockenbach6	33	1	120	16	gockenbach7	33	1	120	16
gockenbach8	33	1	120	16	gockenbach9	33	1	120	16
gockenbach10	33	1	120	16	goerner1	4	1	0	2
goerner2	5	1	0	2	goerner3	7	1	0	2

Continues ...

[1] *http://www.eng.uminho.pt/~dps/aivaz/*

... continued

Problem	nx	nt	nrx	nrt	Problem	nx	nt	nrx	nrt
goerner4	7	2	0	2	goerner5	7	2	0	2
goerner6	16	2	0	2	goerner7	8	2	0	2
gugat1	9	1	0	4	gugat2	9	1	0	4
gugat3	7	1	2	2	gugat4a	9	1	0	4
gugat4b	9	1	0	4	gugat4c	9	1	0	4
gugat4d	9	1	0	4	gugat4e	9	1	0	4
gugat4f	9	1	0	4	gugat5a	7	1	0	4
gugat5b	7	1	0	4	gugat5c	7	1	0	4
gugat5d	7	1	0	4	gugat5e	7	1	0	4
gugat5f	7	1	0	4	gugat6	6	1	0	4
gugat7	4	1	0	4	hettich1	9	2	0	2
hettich2	3	1	0	2	hettich3	5	1	0	2
hettich4	2	1	0	2	hettich5	3	2	0	2
hettich6	7	2	0	2	hettich7	7	2	0	2
hettich8	5	1	0	2	hettich9	11	2	0	2
hettich10	2	1	0	2	hettich10c	2	1	0	2
kortanek1	2	1	0	1	kortanek2	2	2	0	1
kortanek3	7	1	0	1	kortanek4	8	1	0	1
leon1	4	1	0	2	leon2	6	1	0	2
leon3	6	1	0	2	leon4	7	1	0	2
leon5	8	1	0	2	leon6	5	1	0	2
leon7	5	1	0	2	leon8	7	1	0	2
leon9	7	1	0	2	leon10	3	1	0	2
leon11	3	1	0	2	leon12	2	1	0	1
leon13	2	1	0	1	leon14	2	1	0	1
leon15	2	1	0	1	leon16	3	1	0	1
leon17	3	1	0	1	leon18	2	1	0	1
leon19	5	1	0	1	li1	10	1	0	1
li2	6	1	0	1	lin1	6	2	0	1
liu1	2	1	0	1	liu2	2	1	0	1
liu3	16	1	0	2	matlab1	3	1	0	2
matlab2	3	2	0	1	polak1	4	2	0	2
polak2	4	2	0	2	potchinkov1	298	2	0	4
potchinkov2	65	3	0	6	potchinkov3	66	2	0	4
potchinkov4a	67	1	19	2	potchinkov4b	65	1	19	1
potchinkovPL	122	2	0	4	potchinkovPLR	122	2	0	4
powell1	2	1	0	1	priceK	2	1	0	1
priceS3	4	3	0	1	priceS4	4	4	0	1
priceS5	4	5	0	1	priceS6	4	6	0	1
priceT	4	3	0	1	priceU	4	6	0	1
reemtsen1	11	3	0	2	reemtsen2	10	2	0	2

Continues ...

... continued

Problem	nx	nt	nrx	nrt	Problem	nx	nt	nrx	nrt
reemtsen3	10	2	0	2	reemtsen4	37	2	0	2
reemtsen5	11	3	0	2	still1	2	1	0	1
tanaka1	2	1	1	1	teo1	3	1	0	1
teo2	3	1	0	1	userman	2	1	1	1
watson1	2	1	0	1	watson2	2	1	0	1
watson3	3	1	0	1	watson4a	3	1	0	1
watson4b	6	1	0	1	watson4c	8	1	0	1
watson5	3	1	0	1	watson6	2	1	0	1
watson7	3	2	0	1	watson8	6	2	0	1
watson9	6	2	0	1	watson10	3	2	0	1
watson11	3	2	0	1	watson12	3	2	0	1
watson13	3	2	0	1	watson14	2	1	0	1
zhou1	2	1	0	1					

Table 4: Problem in SIPAMPL database

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] = sipampl('userman')
               f = sipampl(x)
          [f,Grad] = sipampl(x)
[c,ceq,K1,...,Kntc,s] = sipampl(x,s)
                      sipampl('msg',x)
```

The behaviour of `sipampl` MATLAB function depends on the number of input and output arguments, so:

- `[x0,ntc,xbl,xbu] = sipampl('userman')` reads the problem named `stub` (`stub.nl` 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 = sipampl(x)` returns the objective function value at `x`.

- `[f,Grad] = sipampl(x)` returns the objective value and the gradient vector at `x`.

- `[c,ceq,K1,..., Kntc,s] = 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 `ceq` are the vectors with the values of the finite constraints, `c` for inequality constraints and `ceq` for equality constraints. The `K1`,...,`Kntc` are `ntc` vectors (or matrices) with the infinite constraints evaluated at the grid.

- `sipampl('msg',x)` writes the ampl solution `x` with the `msg` text message.

A simple example using these functions is:

```
>> [x0,ntc,xbl,xbu] = sipampl('userman');
>> options = optimset('GradObj', 'on');
>> x = fseminf('sipampl',x0,ntc,'sipampl',[],[],[],[],...
               xbl,xbu,options);
>> sipampl('Solution found by MATLAB',x);
```

The interface developed here includes two files. `sipampl.c` is the main program which provides the MEX ([31]) function. `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 `make` (Linux) or `nmake -f makefile.vc` (MS-Windows). `(n)make` calls the `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_`*xxx* 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 `sipampl2`. 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`.

- `[inf]=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 `inf` is the value of the infinite constraint `n` evaluated at `x`,`t`.

- `[gx, gt]=sip_contgrd(n,x,t);`

  where `n` is a constraint number, `x` is an array of finite variables, `t` is an array of infinite variables, `gx` is the gradient, w.r.t. x, of the infinite constraint `n` computed at `x`,`t` and `gt` is the gradient, w.r.t. t, of infinite constraint `n` computed at `x`,`t`.

- `[hx, ht, hxt]=sip_conthes(n);`

  where `n` is a constraint number, `hx` is the Hessian, w.r.t. xx, of the infinite constraint `n` computed at `x`,`t`, `ht` is the Hessian, w.r.t. tt, of the infinite constraint `n` computed at `x`,`t` and `hxt` is the Hessian, w.r.t. xt, of the infinite constraint `n` computed at `x`,`t`.

- `[ceq]=sip_conxeqval(n,x);`

  where `n` is a constraint number, `x` is an array of finite variables and `ceq` is the value of equality finite constraint `n` evaluated at `x`.

- `[geq]=sip_conxeqgrd(n,x);`

  where `n` is a constraint number, `x` is an array of finite variables and `geq` is the gradient of the equality finite constraint `n` computed at `x`.

- `[heq]=sip_conxeqhes(n);`

  where `n` is a constraint number, `heq` is the Hessian of equality finite constraint `n` computed at `x`;

- `[c]=sip_conxineqval(n,x);`

  where `n` is a constraint number, `x` is an array of finite variables and `c` is the value of inequality finite constraint `n` evaluated at `x`.

- `[g]=sip_conxineqgrd(n,x);`

  where `n` is a constraint number, `x` is an array of finite variables and `g` is the gradient of the inequality finite constraint `n` computed at `x`;

- `[h]=sip_conxineqhes(n);`

  where `n` is a constraint number, `h` is the Hessian of the inequality finite constraint `n` computed at `x`.

- `[jacc, jacceq, jactinf, jacxinf]=sip_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×number of finite variables), `jacceq` is the Jacobian of the equality finite constraints (number of equality constraints×number of finite variables), `jactinf` is the Jacobian w.r.t. t of the infinite constraints (number of infinite constraints×number of infinite variables) and `jacxinf` is the Jacobian w.r.t. x of infinite constraints (number of infinite constraints×number of finite 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

$$\begin{aligned} & \min_{x \in R^3} \; f(x) \equiv (x_1 - 0.5)^2 + (x_2 - 0.5)^2 + (x_3 - 0.5)^2 \\ s.t. \quad & \sin(t_1 x_1)\cos(t_1 x_2) - \frac{1}{1000}(t_1 - 50)^2 - \sin(t_1 x_3) - x_3 - 1 \leq 0, \\ & \sin(t_1 x_2)\cos(t_1 x_1) - \frac{1}{1000}(t_1 - 50)^2 - \sin(t_1 x_3) - x_3 - 1 \leq 0, \\ & \forall t_1 \in [1, 100] \; . \end{aligned} \tag{2}$$

as example. We start by writing a M-File for the objective function. Let `mysipfun.m` be such a file with the following content.

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

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

f=sip_objval(x);

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

We write an M-File for the constraints. Let `mysipcon.m` be such a file with the following content.

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

```
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.

`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. `sip_fun.m` and `sip_con.m` are the M-files that compute the objective function and constraints values, respectively. `sip_solve.m` does all the interface to the SIPAMPL routines and calls `fseminf` to solve the problem given as the only requested argument. `sip_solve` does not return any arguments, but adds a line to the `results` file in a LaTeX syntax and prints the solution found. The use of `sip_solve` to solve `matlab1` and `matlab2` problems is shown bellow.

```
>> sip_solve('matlab1')
Optimization terminated successfully:
 Search direction less than 2*options.TolX and
  maximum constraint violation is less than options.TolCon
Active Constraints:
     7
    10

Solution:
    0.6673
    0.3013
    0.4023

Objective value:
    0.0770
```

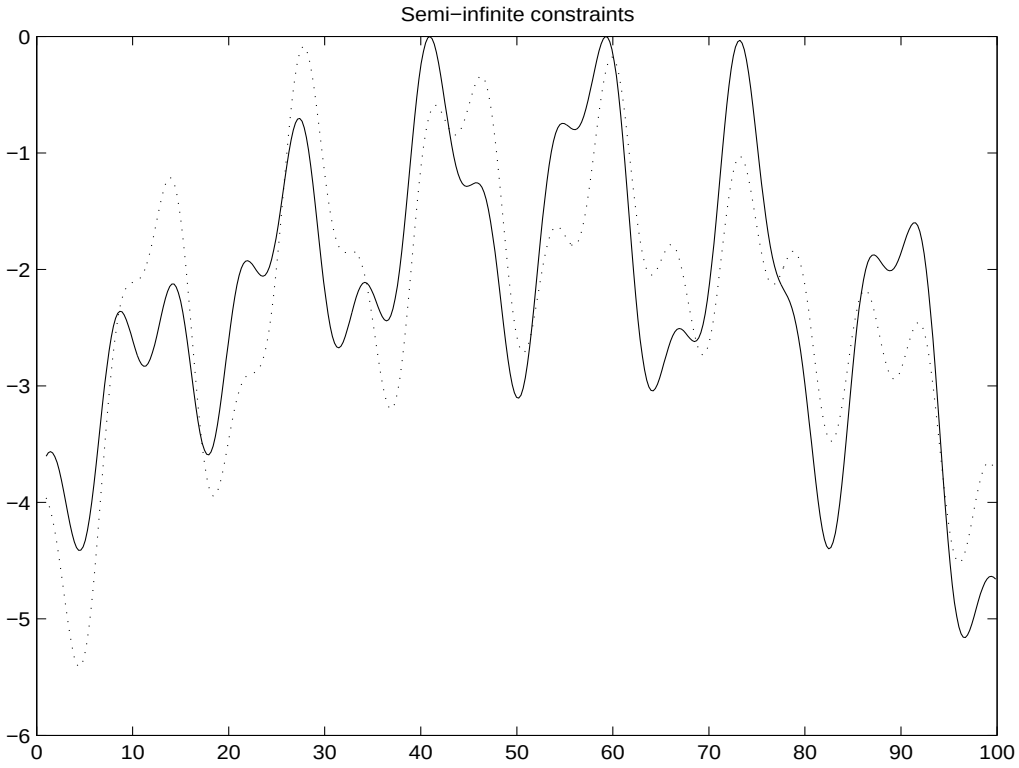

Figure 1: Plot of the infinite constraints in one-dimensional MATLAB problem, in the solution found

```
>> 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 problem 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; "default" 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. Subsection 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 `nmake -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

Option	type	allowed values	default	SIP
Objective type	list	Linear Quadratic Polynomial Generic All type	All type	
Constraints type	list	Linear Quadratic Polynomial Generic All type	All type	
Number finite variables	range	Nonnegative integers	[0,+∞]	n
Number infinite variables	range	Nonnegative integers	[0,+∞]	p
Number finite constraints	range	Nonnegative integers	[0,+∞]	q
Number infinite constraints	range	Nonnegative integers	[0,+∞]	m
Limits finite variables	list	Both limits finite Lower limits finite Upper limits finite None limits finite Everything	Everything	
Limits infinite variables	list	Both limits finite Lower limits finite Upper limits finite None limits finite Everything	Everything	[α, β]
Initial guess	list	With initial guess Without initial guess With or without initial guess	With or without initial guess	

Table 5: Questionable problems characteristics

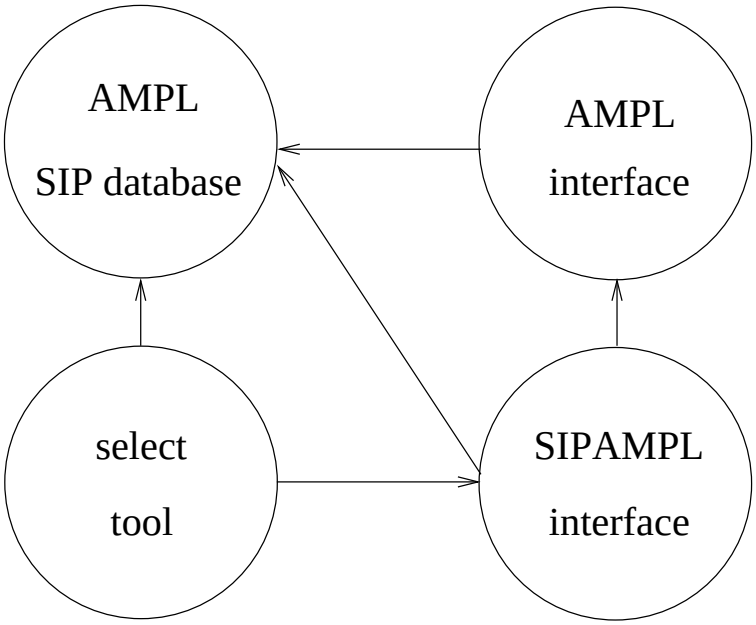

Figure 2: Interconnections between *select* and SIPAMPL

```
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 `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 `nsips` or renamed `nsips`.
- the user must edit all the files to change `nsips` for his own solver name.

To abbreviate the second action we give some *bash* shell commands to do this in an automatic form

for i in `whatever files you want to change solver`
do
sed s/nsips/`newsolver`*/ < $i > $i.new*
done

These commands will read the files `whatever files you want to change solver` and will write the new files `whatever.new files.new you.new want.new to.new change.new solver.new`.

Note that the *sed* command will replace every word *nsips* by `newsolver`. If the word `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

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

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 `amplsolver.a` (Linux) or `amplsolv.lib` (MS-Windows) library. This library provides AMPL functions. The file `asl.h` has important data structure definitions, being the `asl` 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 (`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 (`stub`.nl). The solver reads the `stub`.nl file into memory. Most of the problem is kept in memory in an `asl` 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 `stub`.sol file with the solution found. AMPL reads `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 `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/

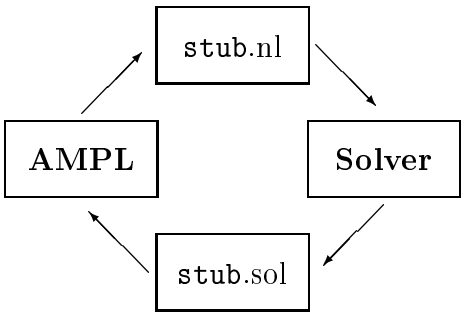

Figure 3: Interaction between AMPL and the Solver.

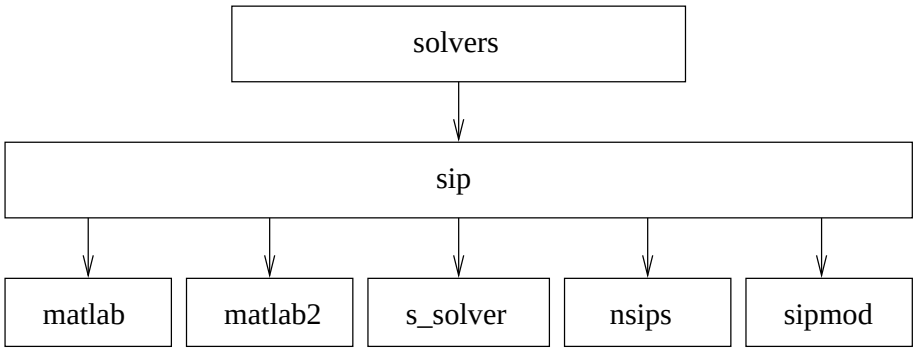

Figure 4: Organization of SIPAMPL directories

- `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 `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

[1] *MATLAB*, The MatWorks Inc., 1999, Version 5.4, Release 11.

[2] I. Adler, M.G.C. Resende, G. Veiga, and N. Karmarkar, *An implementation of karmarkar's algorithm for linear programming*, Mathematical Programming **44** (1989), 297–335.

[3] E.J. Anderson and A.S. Lewis, *An extension of the simplex algorithm for semi-infinite linear programming*, Mathematical Programming **44** (1989), 247–269.

[4] J.W. Blankenship and J.E. Falk, *Infinitely constrained optimization problems*, Journal of Optimization Theory and Applications **19** (1976), no. 2, 261–281.

[5] I. Bongartz, A.R. Conn, N. Gould, and Ph.L. Toint, *CUTE: Constrained and Unconstrained Testing Environment*, ACM Transactions on Mathematical Software **21** (1995), no. 1, 123–160, ftp://130.246.9.91/pub/cute.

[6] T. Coleman, M.A. Branch, and A. Grace, *Optimization Toolbox for Use with MATLAB*, The MathWorks Inc., 1999.

[7] I.D. Coope and G.A. Watson, *A projected lagrangian algorithm for semi-infinite programming*, Mathematical programming **32** (1985), no. 3, 337–356.

[8] R. Hettich (Ed.), *Semi-Infinite Programming*, Springer-Verlag, 1979.

[9] S.-C. Fang and S.-Y. Wu, *An inexact approach to solving linear semi-infinite programming problems*, Optimization **28** (1994), 291–299.

[10] M.C. Ferris and A.B. Philpott, *An interior point algorithm for semi-infinite linear programming*, Mathematical Programming **43** (1989), 257–276.

[11] A.V. Fiacco and K.O. Kortanek (Eds.), *Semi-Infinite Programming and Applications*, Springer-Verlag, 1983.

[12] R. Fourer, D.M. Gay, and B.W. Kernighan, *A modeling language for mathematical programming*, Management Science **36** (1990), no. 5, 519–554.

[13] D.M. Gay, *Hooking your solver to AMPL*, Numerical Analysis Manuscript 93-10, AT&T Bell Laboratories, ftp://netlib.bell-labs.com/netlib/att/cs/doc/93/4-10.ps.Z (1993).

[14] M.S. Gockenbach and A.J. Kearsley, *Optimal signal sets for non-gaussian detectors*, SIAM Journal on Optimization **9** (1999), no. 2, 316–326.

[15] C. Gonzaga, E. Polak, and R. Trahan, *An improved algorithm for optimization problems with functional inequality constraints*, IEEE Transactions on Automatic Control **AC-25** (1980), no. 1, 49–54.

[16] S. Görner, *Ein hybridverfahren zur lösung nichtlinearer semi-infiniter optimierungsprobleme*, Ph.D. thesis, Fachbereich Mathematik der Technischen Universität Berlin, 1997.

[17] M. Gugat, *Fractional semi-infinite programming*, Ph.D. thesis, Trier University, Germany, 1994.

[18] E. Haaren-Retagne, *A semi-infinite programming algorithm for robot trajectory planning*, Ph.D. thesis, University of Trier, 1992.

[19] R. Hettich, *A comparison of some numerical methods for semi-infinite programming*, in [8] (1979), 112–125.

[20] ———, *An implementation of a discretization method for semi-infinite programming*, Mathematical Programming **34** (1986), no. 3, 354–361.

[21] R. Hettich and G. Gramlich, *A note on an implementation of a method for quadratic semi-infinite programming*, Mathematical Programming **46** (1990), 249–254.

[22] L.S. Jennings and K.L. Teo, *A computational algorithm for functional inequality constrained optimization problems*, Automatica **26** (1990), no. 2, 371–375.

[23] B.W. Kernighan and D.M. Richie, *The C Programming Language*, Prentice Hall, 1988.

[24] K.O. Kortanek and H. No, *A central cutting plane algorithm for convex semi-infinite programming problems*, SIAM Journal on Optimization **3** (1993), no. 4, 901–918.

[25] C.T. Lawrence and A.L. Tits, *Feasible sequential quadratic programming for finely discretized problems from SIP*, in [38] (1998), 159–193.

[26] T. León, S. Sanmatías, and E. Vercher, *A multi-local optimization algorithm*, Top **6** (1998), no. 1, 1–18.

[27] _____, *On the numerical treatment of linearly constrained semi-infinite optimization problems*, European Journal of Operational Research **121** (2000), 78–91.

[28] Y. Li and D. Wang, *A semi-infinite programming model for Earliness/Tardiness production planning with simulated annealing*, Mathematical and Computer Modelling **26** (1997), no. 7, 35–42.

[29] C.-J. Lin, S.-C. Fang, and S.-Y. Wu, *An unconstrained convex programming approach to linear semi-infinite programming*, SIAM Journal on Optimization **8** (1998), no. 2, 443–456.

[30] Y. Liu, K.L. Teo, and S. Ito, *A dual parametrization approach to linear-quadratic semi-infinite programming problems*, Optimization Methods and Software **10** (1999), 471–495.

[31] MatWorks, *Application Program Interface Guide*, The MatWorks Inc., 1996.

[32] E.R. Panier and A.L. Tits, *A globally convergent algorithm with adaptively refined discretization for semi-infinite optimization problems arising in engineering design*, IEEE Transactions on Automatic Control **34** (1989), no. 8, 903–908.

[33] E. Polak, L. Qi, and D. Sun, *First-order algorithms for generalized semi-infinite min-max problems*, Computational Optimization and Applications **13** (1999), 137–161.

[34] A.W. Potchinkov, *Design of optimal linear phase FIR filters by a semi-infinite programming technique*, Signal Processing (1997), 165–180.

[35] C.J. Price, *Non-linear semi-infinite programming*, Ph.D. thesis, University of Canterbury, New Zealand, August 1992.

[36] C.J. Price and I.D. Coope, *Numerical experiments in semi-infinite programming*, Computational Optimization and Applications **6** (1996), 169–189.

[37] R. Reemtsen, *Discretization methods for the solution of semi-infinite programming problems*, Journal of Optimization Theory and Applications **71** (1991), no. 1, 85–103.

[38] R. Reemtsen and J.-J. Rückmann (Eds.), *Semi-Infinite Programming*, Kluwer Academic Publishers, 1998.

[39] G. Still, *Discretization in semi-infinite programming: The rate of convergence*, Mathematical Programming **91** (2001), 53–69.

[40] Y. Tanaka, *A trust region method for semi-infinite programming problems*, International Journal of Systems Science **30** (1999), no. 2, 199–204.

[41] K.L. Teo and C.J. Goh, *A simple computational procedure for optimization problems with functional inequality constraints*, IEEE Transactions on Automatic Control **AC-32** (1987), no. 10, 940–941.

[42] K.L. Teo, V. Rehbock, and L.S. Jennings, *A new computational algorithm for functional inequality constrained optimization problems*, Automatica **29** (1993), no. 3, 789–792.

[43] M.J. Todd, *Interior-point algorithms for semi-infinite programming*, Mathematical Programming **65** (1994), 217–245.

[44] A.I.F. Vaz, E.M.G.P. Fernandes, and M.P.S.F. Gomes, *NSIPS v2.1: Nonlinear Semi-Infinite Programming Solver*, Technical Report ALG/EF/5-2002 (2002), http://www.eng.uminho.pt/∼dps/aivaz/.

[45] ———, *Optimal signal sets via semi-infinite programming*, Investigação Operacional **22** (2002), no. 1, 87–101.

[46] D. Wang and S.-C. Fang, *A semi-infinite programming model for Earliness/Tardiness production planning with a genetic algorithm*, Computers and Mathematics with Applications **31** (1996), no. 8, 95–106.

[47] G.A. Watson, *Numerical experiments with globally convergent methods for semi-infinite programming problems*, in [11] (1983), 193–205.

[48] J.L. Zhou and A.L. Tits, *An SQP algorithm for finely discretized continuous minimax problems and other minimax problems with many objective functions*, SIAM Journal on Optimization **6** (1996), no. 2, 461–487.