Solving semi-infinite programming problems by using an interface between MATLAB and SIPAMPL

A. ISMAEL F. VAZ EDITE M.G.P. FERNANDES
Minho University, Engineering School, Campus of Gualtar
Production and Systems Department, 4710-057 Braga, Portugal
{aivaz,emgpf}@dps.uminho.pt

Abstract: A new interface between MATLAB and SIPAMPL was created, allowing the MATLAB semi-infinite programming (SIP) solver to use the SIPAMPL [11] environment to obtain the problem data to be solved. In this paper we present the new developed interface, briefly describe the `fseminf` MATLAB solver, provided in the Optimization Toolbox [9], and we show how it can be used to solve a SIP problem available in the SIPAMPL database. Additionally, we present how one may use the interface to develop new SIP algorithms for MATLAB. We report numerical results for a set of SIP problems.

Key–Words: semi-infinite programming, modeling semi-infinite programming problems, MATLAB optimization toolbox, SIPAMPL.

1 Introduction

We have recently made a compilation of semi-infinite programming (SIP) problems collected from the literature. These problems are coded in AMPL [1] and are publicly available to the research community. This problems database, as well as a set of interface routines to connect the AMPL to any SIP solver, a `select` tool and a MATLAB interface are the main components of a software package called SIPAMPL [11]. Until now, the interface between MATLAB and AMPL was done by a set of routines devoted to the use of the `fseminf` MATLAB solver. The development of new solvers in MATLAB that could obtain the coded problems in AMPL was therefore compromised. In this paper, we propose and describe a new approach that was implemented to allow MATLAB to directly use the SIPAMPL interface functions. The new created MATLAB functions add more flexibility when using SIPAMPL to provide the problem data.

We start in Section 2 by introducing the reader to semi-infinite programming and to the used notation. Section 3 is used to briefly describe the `fseminf` MATLAB solver. In Section 4 we describe how the SIPAMPL could be used by MATLAB to solve SIP problems in the database. The new approach for using SIPAMPL in a MATLAB environment is described in Section 5. Numerical results and the conclusions are presented in the last two sections.

2 Semi-infinite programming

SIP problems appear in many engineering areas such as air control pollution [5, 13], robotics [2], production planning [6, 14] and in Chebyshev approximation theory [3, 4, 10]. A nonlinear semi-infinite programming problem is described in the mathematical form as follows:

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

$T \subset R^p$ is an infinite set usually represented by a cartesian product of intervals $([\alpha_1, \beta_1] \times [\alpha_2, \beta_2] \times \cdots \times [\alpha_p, \beta_p])$. These problems are called semi-infinite programming problems due to the constraints $g_u(x,t) \leq 0$, $u = 1, \ldots, m$, that must be satisfied for all $t \in T$. We can consider T as an infinite index set and therefore (1) is a problem with finitely many variables over an infinite set of constraints.

MATLAB definition for a SIP problem is as fol-

lows:

$$\begin{aligned} &\min_{x \in R^n} f(x) \\ s.t.\ \ &g_u(x, t^u) \leq 0,\ u = 1, \ldots, m \\ &h_v(x) = 0,\ v = 1, \ldots, o \\ &h_v(x) \leq 0,\ v = o+1, \ldots, q \\ &A_e x = b_e \\ &Ax \leq b \\ &b_- \leq x \leq b_+ \\ &\forall t^u \in T\ , \end{aligned} \tag{2}$$

where $T \subset R^1$ or $T \subset R^2$.

The main difference between MATLAB definition of SIP (2) and the definition (1) is that MATLAB restricts the number of t variables to be 1 or 2 ($p = \{1, 2\}$), although different constraints may have different variables (t^u for constraint $u = 1, \ldots, m$). The SIP problems from the SIPAMPL database that can be solved by the MATLAB solver are therefore restricted to $p = 1$ or $p = 2$, since SIPAMPL assumes that all t variables are present in all the g constraints.

3 The MATLAB solver

We refer to [9] for the `fseminf` function syntax and details about the used parameters.

The `fseminf` algorithm consists of a quasi-Newton SQP algorithm that uses line search with a merit function applied to a finite problem that results from the SIP problem with the g constraints discretized in certain points. The discretization of the set T is based on an interval sampling `s` that is used to provide an equally spaced grid of points where the g constraint is computed.

To obtain the finite problem the algorithm proceeds in the following way. If $p = 1$ in the g constraint of index u then `s(`u`,1)` is the sampling interval for the scalar element t^u. If $p = 2$ in the g constraint of index u then `s(`u`,1)` and `s(`u`,2)` are the sampling intervals for the two elements vector t^u.

`fseminf` does not use all the grid computed constraint values. It identifies peaks in the data and estimates the maxima of the discretized constraints by using quadratic or cubic interpolation. This approach does not find the maximizers (t^u variable(s) value(s)), but just an estimate of the maxima, therefore the derivatives can not be used, even if they are available.

This algorithm can therefore be classified as a discretization method [5] where the infinite constraints are discretized in an equally spaced grid of points. Since the number of maxima can change during the optimization the Lagrange multipliers estimates are reallocated to the new set of maxima.

4 Using MATLAB with (the old) SIPAMPL interface

The SIPAMPL provides a database with over one hundred and sixty coded SIP problems. The use of the SIPAMPL interface routines by MATLAB allows it to solve the coded problems. The `sipampl` MEX function does the interface between MATLAB and SIPAMPL. The function `sipampl` provided in the SIPAMPL software package is limited to $p = 1$ or $p = 2$ for the reasons already stated in section 2.

The `sipampl` syntax is:

```
[x0,nth,xbl,xbu] = sipampl('stub')
               f = sipampl(x)
        [f,Grad] = sipampl(x)
[c,ceq,K1,...,Knth,s] = sipampl(x,s)
                        sipampl('msg',x)
```

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

- `[x0,nth,xbl,xbu] = sipampl('stub')` reads the problem named `stub` (`stub.nl` file provided by AMPL) and returns the initial guess `x0`, the number of t constraints `nth`, 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,..., Knth,s] = sipampl(x,s)` evaluates the constraints at `x`. `s` is the step size for the grid where the g constraints are evaluated (sampling interval). `c` and `ceq` are vectors for the values of the h constraints, `c` for inequality constraints and `ceq` for equality constraints. `K1`,...,`Knth` are `nth` vectors (or matrices) with the g constraints evaluated on the grid.
- `sipampl('msg',x)` writes the AMPL solution `x` with the text message `msg`.

The interface includes two files. `sipampl.c` is the main program that provides the MEX [7] function

`sipampl`. `sipsolve.m` is a MATLAB file with a short example of how to use MATLAB to solve problems coded with SIPAMPL.

These files are also available by the internet[1] with the problems database and SIPAMPL.

The `sipampl` function requires the intermediate `.nl`, `.col` and `.row` files that are provided by AMPL although they are not directly available. The AMPL `write` command can be used to obtain the needed files.

By providing the MATLAB calls to AMPL in a single `sipampl` MEX function disables the user from directly evaluate the constraints. For example, if the user wishes to plot a graph with the constraint or just to change the initial sample interval he must edit the C source file `sipampl.c`. This limitation and the fact that `sipampl` is closely related to the use of `fseminf` led to the new approach proposed in the next section.

5 A new approach for the SIPAMPL interface with MATLAB

The new approach consists of exporting all the SIPAMPL interface routines directly to MATLAB. The new created 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 the calls to objective or constraints functions;
- `sip_conval` returns all the constraint values;
- `sip_contval` returns an g constraint value;
- `sip_contgrd` returns an g constraint gradient;
- `sip_conthes` returns an g constraint Hessian at the last value used in the calls to objective or constraints functions;
- `sip_conxeqval` returns an equality h constraint value;
- `sip_conxeqgrd` returns an equality h constraint gradient;
- `sip_conxeqhes` returns an equality h constraint Hessian at the last value used in the calls to objective or constraints functions;
- `sip_conxineqval` returns an inequality h constraint value;
- `sip_conxineqgrd` returns an inequality h constraint gradient;
- `sip_conxineqhes` returns an inequality h constraint Hessian at the last value used in the calls to objective or constraints functions;
- `sip_jacval` returns the Jacobian of the h/g constraints;
- `sip_usage` prints the `sip_`*xxx* usage. Used by other functions when reporting an invalid number of arguments.

The number of arguments in each of these functions 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, after doing some checkup on the arguments, call the MEX file `sipampl2`. The interface between AMPL and MATLAB is accomplished by this MEX file, which is provided in source (C programming language) and was tested in Linux and Windows platforms with MATLAB version 6.1, release 12.1. The *mex* compiler is provided with the MATLAB software and it allows code to be written in the C programming language to be used by the MATLAB (see [7] for details).

Figure 1 shows a diagram with the several interfaces between AMPL and MATLAB.

We will use the one-dimensional problem `matlab1` available in the SIPAMPL database as an example. This and `matlab2` are the two problems described in the MATLAB user manual. We start by writing an 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');
```

[1] http://www.norg.uminho.pt/aivaz/

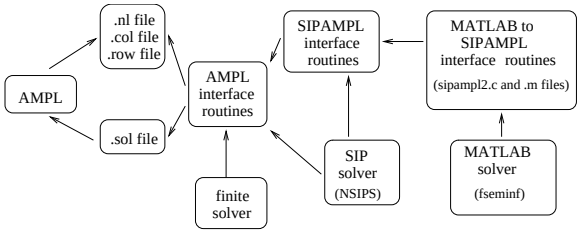

Figure 1: Diagram of interfaces between AMPL and MATLAB

```
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 files and using AMPL to produce `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')
...
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.

6 Numerical results

The user defined MATLAB function to evaluate the g constraints is significantly different from one- or two-dimensional problems. To run all SIPAMPL database 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_con.m` should be edit to set the initial sampling interval for each problem. For problem `matlab1` we have used `s=[0.2 0; 0.2 0]` and for `matlab2` `s=[2 2]`.

`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 argument, but adds a line to the file `results` in a LaTeX syntax and prints the solution found. In this case the file contains the following two lines:

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

where the first column corresponds to the problem name, the second gives the number of iterations, the third gives the number of objective function evaluations and the last is the objective function value at the found solution.

SIPAMPL has a `select` tool that allows the selection of the problems from the database with given characteristics. Table 1 shows the SIPAMPL problems from the database with $p = 1$ or $p = 2$ and with both limits of the infinite set T finite. Problems that do not have an initial guess were not considered, since providing a random initial guess for some problems can really affect the solver performance. In problems *gockenbach1*, ..., *gockenbach10*, the initial guesses are randomly generated by AMPL (these problems were coded in a way that an initial guess is randomly generated whenever AMPL is called). These initial guesses remained fixed in all the solver runs after obtaining the `.nl`, `.col` and `.row` files.

To produce the `.nl`, `.col` and `.row` files while selecting the problems from the database, the `select` tool must be used in expert mode (see [12]).

The initial sampling interval was set to `s(i,j)` = $\frac{\beta_j - \alpha_j}{100}$, $i = 1, \ldots, m$, $j = 1, \ldots, p$. The `fseminf` function uses quadratic or cubic interpolation to compute the maxima of the infinite constraints. To obtain a quadratic/cubic interpolation of the g constraint function at least three/four points in the grid are necessary. The sampling interval (grid space `s`) is updated by the `fseminf` function during the iterative process. Since `fseminf` does not check for integrity in the sampling interval we check and change the sampling interval to provide a grid with at least four points per constraint.

In the table: "Problem" is the problem name under SIPAMPL, n is the number of x variables, p is the number of t variables; q is the number of h constraints, m is the number of g constraints, "nit" is the number of iterations, "nf" is the number of function evaluations and "fx" is the objective value at the found solution.

Some problems were not solved because of the following reasons:

[1] - Problem was stopped with a division by zero in the cubic interpolation (used to estimate the constraints peaks).

[2] - The number of g constraints makes these problems time consuming when solved by MATLAB. We have stopped *gockenbach9* after 12 hours of computation and then decided to reduce the initial sampling size to $s(i,1) = \frac{\beta_1 - \alpha_1}{50}$, $i = 1, \ldots, m$. With this sampling a division by zero, in the cubic interpolation, occurred in problems *gochenbach10*, *gochenbach8*, *gockenbach7* and *gockenbach6*. *gockenbach5* was solved and *gockenbach9*, *gockenbach4*, *gockenback3*, *gockenback2* and *gockenback1* were stopped after 10 hours of computation time.

Problem	n	p	q	m	nit	nf	fx
andreson1	3	2	0	1	4	21	-3.33E-1
blankenship1	2	1	0	1	16	84	0.00E+0
coopeL	2	1	0	1	7	34	3.43E-1
coopeM	2	1	1	1	4	20	1.00E+0
coopeN	2	1	0	1	5	21	0.00E+0
elke10	9	1	0	7			[1]
elke1std	9	1	0	19			[1]
elke2std	9	1	0	19			[1]
elke3std	9	1	0	19			[1]
elke4std	9	1	0	7			[1]
elke5std	9	1	0	19			[1]
elke6std	9	1	0	19			[1]
elke7std	9	1	0	19			[1]
elke8	9	1	0	7			[1]
elke9	9	1	0	7			[1]
fang1	50	1	0	1	17	885	4.79E-1
fang2	50	1	0	1	41	2220	6.93E-1
fang3	50	1	0	1	78	4229	1.72E+0
ferris1	7	1	0	2	39	365	2.19E-3
ferris2	7	1	0	1	26	237	-1.78E+0
gockenbach1	33	1	120	16			[2]
gockenbach10	33	1	120	16			[2]
gockenbach2	33	1	120	16			[2]
gockenbach3	33	1	120	16			[2]
gockenbach4	33	1	120	16			[2]
gockenbach5	33	1	120	16	13	470	-7.88E-3
gockenbach6	33	1	120	16			[2]
gockenbach7	33	1	120	16			[2]
gockenbach8	33	1	120	16			[2]
gockenbach9	33	1	120	16			[2]
goerner1	4	1	0	2	15	98	4.13E-3
goerner2	5	1	0	2	15	123	4.64E-3
goerner3	7	1	0	2	18	167	6.92E-4
goerner4	7	2	0	2	9	85	5.24E-2
goerner5	7	2	0	2	17	183	2.71E-2
goerner6	16	2	0	2	52	1030	2.31E-3
goerner7	8	2	0	2	34	479	9.50E-2
hettich10c	2	1	0	2			[1]
hettich5	3	2	0	2	9	78	5.40E-1
leon1	4	1	0	2	33	277	5.22E-3
leon10	3	1	0	2	7	42	5.37E-1
leon11	3	1	0	2	55	302	1.85E+0
leon12	2	1	0	1	12	66	-1.00E+0
leon13	2	1	0	1			[1]
leon14	2	1	0	1			[1]
leon15	2	1	0	1	6	25	-6.67E-1
leon16	3	1	0	1	18	151	1.86E+0
leon17	3	1	0	1	2	11	-2.00E+0
leon18	2	1	0	1			[1]
leon19	5	1	0	1	24	187	7.86E-1
leon2	6	1	0	2	24	202	4.20E-5
leon3	6	1	0	2	14	143	4.84E-3
leon4	7	1	0	2	15	149	2.60E-3
leon5	8	1	0	2	32	380	1.43E-2
leon6	5	1	0	2	41	347	1.38E-4
leon7	5	1	0	2	32	245	1.97E-3
leon8	7	1	0	2	10	107	5.44E-2
leon9	7	1	0	2	23	268	2.04E-1
li1	10	1	0	1	62	1012	2.81E+5
li2	6	1	0	1	22	199	3.96E+4
lin1	6	2	0	1	22	189	-1.82E+0

Table 1: Numerical results with selected problems

7 Conclusions

We provide new MATLAB functions that use the SIPAMPL interface routines to obtain the SIP problem data to be solved. These new functions allow the use of the MATLAB `fseminf` function to solve SIP problems and support the development of new solvers in MATLAB that can be connected with the SIPAMPL problems database. We illustrate this new approach with numerical results of the `fseminf` SIP MATLAB solver.

Problem	n	p	q	m	nit	nf	fx
matlab1	3	1	0	2	16	83	3.57E-3
matlab2	3	2	0	1	4	22	2.00E-4
powell1	2	1	0	1	12	66	-1.00E+0
priceK	2	1	0	1	7	30	-3.00E+0
random	4	2	4	4	1	7	-1.00E-6
tanaka1	2	1	1	1	17	114	-1.00E+0
teo1	3	1	0	1	16	81	1.68E-1
teo2	3	1	0	1	18	91	1.85E-1
watson1	2	1	0	1	12	53	-2.50E-1
watson10	3	2	0	1	3	16	2.75E-1
watson11	3	2	0	1	13	71	-4.39E+0
watson12	3	2	0	1	5	26	1.95E+0
watson13	3	2	0	1	13	74	1.95E+0
watson14	2	1	0	1	30	204	2.13E+0
watson2	2	1	0	1	7	29	2.43E+0
watson3	3	1	0	1	41	305	5.13E+0
watson4a	3	1	0	1	19	152	6.47E-1
watson4b	6	1	0	1	17	175	6.17E-1
watson4c	8	1	0	1	30	398	6.16E-1
watson5	3	1	0	1	7	55	4.30E+0
watson6	2	1	0	1	18	102	9.72E+1
watson7	3	2	0	1	7	36	1.00E+0
watson8	6	2	0	1	36	392	2.44E+0
watson9	6	2	0	1	38	412	-1.04E+1
zhou1	2	1	0	1			[1]

Table 1: Numerical results with selected problems (continued).

Acknowledgements: The research was supported by the Algoritmi Research Center, and by the Portuguese FCT under grant POCI/MAT/58957/2004.

References:

[1] R. Fourer, D.M. Gay, and B.W. Kernighan. A modeling language for mathematical programming. *Management Science*, 36(5):519–554, 1990.

[2] E. Haaren-Retagne. *A Semi-Infinite Programming Algorithm for Robot Trajectory Planning*. PhD thesis, University of Trier, 1992.

[3] R. Hettich. An implementation of a discretization method for semi-infinite programming. *Mathematical Programming*, 34(3):354–361, 1986.

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

[5] R. Hettich and K.O. Kortanek. Semi-infinite programming: Theory, methods, and applications. *SIAM Review*, 35(3):380–429, 1993.

[6] Y. Li and D. Wang. A semi-infinite programming model for Earliness/Tardiness production planning with simulated annealing. *Mathematical and Computer Modelling*, 26(7):35–42, 1997.

[7] MathWorks. *Application Program Interface Guide*. The MathWorks Inc., 1996.

[8] MathWorks. *MATLAB*. The MathWorks Inc., 2001. Version 6.1, Release 12.1.

[9] MathWorks. *MATLAB Optimization Toolbox*. The MathWorks Inc., 2001. In [8].

[10] R. Reemtsen. Discretization methods for the solution of semi-infinite programming problems. *Journal of Optimization Theory and Applications*, 71(1):85–103, 1991.

[11] A.I.F. Vaz, E.M.G.P. Fernandes, and M.P.S.F. Gomes. SIPAMPL: Semi-infinite programming with AMPL. *ACM Transactions on Mathematical Software*, 30(1):47–61, March 2004.

[12] A.I.F. Vaz, E.M.G.P. Fernandes, and M.P.S.F. Gomes. SIPAMPL v2.1: Semi-Infinite Programming with AMPL. Technical Report ALG/EF/2-2003, Universidade do Minho, Braga, Portugal, Dezembro 2004. http://www.norg.uminho.pt/aivaz/.

[13] A.I.F. Vaz and E.C. Ferreira. Semi-infinite air pollution control problems. In *XXVIII Congreso Nacional de la SEIO*, page CDROM, Cádiz, Espanha, 2004. ISBN 84-689-0438-4.

[14] 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(8):95–106, 1996.