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

NSIPS version 2.1
Nonlinear Semi-Infinite Programming Solver

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

Technical Report

ALG/EF/5-2002

November 2002

NSIPS version 2.1
Nonlinear Semi-Infinite Programming Solver

A. Ismael F. Vaz
Edite M.G.P. Fernandes
Departamento de Produção e Sistemas,
Escola de Engenharia, Universidade do Minho
Campus de Gualtar, 4710 Braga, Portugal
Email: {aivaz,emgpf}@dps.uminho.pt

M. Paula S.F. Gomes
Mechanical Engineering Department,
Mechatronics in Medicine Laboratory,
Imperial College of Science, Technology and Medicine,
London SW7 2BX, United Kingdom
Email:p.gomes@ic.ac.uk

November 2002

Contents

Chapter 1

Change Log

1.1 Changes from version 1.0:

- Corrected a bug in the Hash table of the discretization method;
- One more multiplier penalty function;
- Infeasible interior point quasi-Newton BFGS algorithm;
- Solver complains if problem should not be solved by the selected method.

1.2 Changes from version 2.0:

- NSIPS, `select` tool, `sipampl` MATLAB function: code changed to compile with the Microsoft Visual C/C++ compiler;

Chapter 2

Introduction

In a previous work we have extended the AMPL [10] environment to allow the codification of semi-infinite programming problems (SIP) and coined it as SIPAMPL [36]. The interface to the SIPAMPL database is now improved with the NSIPS software providing the database and the interface with a solver.

The NSIPS includes (at this moment) four solvers: a discretization solver [37, 45, 42]; a penalty technique solver [39, 41], a sequential quadratic programming (SQP) solver [44] and an infeasible quasi-Newton interior point solver [46].

In Chapter 3 we describe how the NSIPS was made, how to install and how it interconnects with the AMPL binary. Chapter 4 is devoted to the discretization method which is based on two discretization algorithms proposed by Hettich [15, 16] and Reemtsen [29] and adapted for nonlinear semi-infinite programming [37] and to a discretization method based on Hettich version with pseudo-random points. Chapter 5 describes the penalty method which implements several penalty functions and requires numerical computation of integrals [39, 40, 41]. Chapter 6 describes a sequential quadratic programming (SQP) method where the quadratic SIP problem (QSIP) is solved by a dual parametrization of the dual functions [43, 44]. Chapter 7 describes the interior point method. Chapter 8 shows, in a user point of view, the SIPAMPL interface and database and Chapter 9 presents the *select* tool which is distributed with the NSIPS software.

The remaining of this chapter is devoted to a brief introduction to semi-infinite programming.

2.1 Disclaimer

This software was developed in the context of the Ph.D. work of the first author advised by the second and third authors. This work is far from being complete. We disclaim any responsibility for the use of this software and solutions presented by the solver should be carefully used. This software is provides in the *as is* basis and may not suite your purpose although we welcome any suggestions and improvements.

2.2 Semi-Infinite Programming

A SIP problem is a mathematical program of the form:

$$\begin{aligned} &\min_{x \in R^n} f(x) \\ s.t.\ \ &g_i(x,t) \leq 0,\ i = 1,\ldots,m \\ &h_i(x) \leq 0,\ i = 1,\ldots,o \\ &h_i(x) = 0,\ i = o+1,\ldots,q \\ &\forall t \in T\ , \end{aligned} \tag{2.1}$$

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

These problems are called semi-infinite programming problems due to the constraints $g_i(x,t) \leq 0,\ i = 1,\ldots,m$. T is an infinite index set and therefore (2.1) is a problem with finitely many variables over an infinite set of constraints.

A natural way to solve the SIP problem (2.1) is to replace the infinite set T by a finite one. There are several ways of doing this. One is by discretization methods, the other is by exchange methods, and another by reduction methods (see [17], for a more detailed explanation).

In discretization methods the infinite set T is replaced by a sequence of subsets $T_0 \subset T_1 \subset \cdots \subset T_{\mathcal{N}} \subset T$ (usually the subsets $T_k,\ k = 0,\ldots,\mathcal{N}$ are grids of points). In each iteration some points in the subset T_k are chosen and used in the constraints to form a finite sub-problem. The solution to the SIP problem is approximated by the solution on the final subset $T_{\mathcal{N}}$ and may not be a stationary point for SIP (see, for example [15, 16, 20, 29, 37]).

In exchange methods [13, 30, 49, 53] approximated solutions to the following problems are computed, for a given $\overline{x} \in R^n$

$$\max_{t \in T} g_i(\overline{x},t),\ \ i = 1,\ldots,m. \tag{2.2}$$

The finite sub-problem is then solved with the approximated solutions.

In reduction methods [14, 21, 27, 28, 50] all the global and some local maxima for the problem (2.2) are obtained. The finite sub-problem is then solved with the solutions found to problem (2.2).

The first method implemented (Discretization method, Chapter 4) belongs to the discretization class. This solver will solve any problem in the SIPAMPL database that suits the definition (2.1). The other three methods implemented (Penalty solver, Chapter 5, SQP solver, Chapter 6 and Interior Point solver, Chapter 7) are restricted to problems in the SIPAMPL database that have no finite constraints and only one infinite variable. We include these methods in the discretization class, since the problem (2.2) is not addressed, although the methods do not use a grid of points.

Chapter 3

Nonlinear Semi-Infinite Programming Solver

The Nonlinear Semi-Infinite Programming Solver (NSIPS) is a solver for semi-infinite programming problems. NSIPS v1.0 [38] implements three different methods: discretization, penalty functions and sequential quadratic programming. The discretization algorithm has three versions and the penalty algorithm has two versions, resulting in a total of six algorithms. Version 2.0 adds the interior point method.

The discretization and SQP methods use the NPSOL [11] to solve the resulting finite sub-problem. NPSOL is a commercial software and therefore we can not provide it with the NSIPS. The distributed NSIPS binary version does not include the NPSOL subroutines and so the discretization and SQP methods can not be used. You can obtain the NPSOL through the following addresses:

```
Stanford Business Software
2680 Bayshore Parkway, Suite 304
Mountain View, CA 94043
Phone:  +1-415-962-8719
Fax:    +1-415-962-1869
```

or by the Internet address

```
http://www.sbsi-sol-optimize.com/
```

If NPSOL is available compile the `libopt.a`, using the instructions provided with it, and then use it with the NSIPS.

In the next section, the instructions on how to obtain, use and install the NSIPS software are described. Section 3.3 shows how to pass options to the NSIPS and Section 3.4 presents the option for selecting the method used to solve a problem. The options referring to the methods are discussed in the solver specific chapter. The output produced by the solver is discussed in Section 3.5 and in sections for each specific solver.

When no initial guess is known for the problem, an extra initial NLP sub-problem is solved with five discrete points per dimension and with a random initial guess (random between 0 and 1 in all dimensions). The solution from this NLP sub-problem is the initial guess to the SIP problem. For the NSIPS version without the NPSOL software it will not be possible to start any method with problems that do not have an initial guess.

3.1 Software limitations

Methods are limited to solve only problems of minimization with constraints of type $\leq$. The following table presents the other limitations.

Method	Discretization	Penalty	SQP	Interior Point
To what SIP problems does it apply?	• finite intervals of t	• only infinite constraints • finite interval of t • one infinite variable	• only infinite constraints • finite interval of t • one infinite variable	• only infinite constraints • finite interval of t • one infinite variable
External needs	NPSOL	NPSOL (just for problems without initial guess)	NPSOL	NPSOL (just for problems without initial guess)

3.2 Using and installing NSIPS

The NSIPS software can be obtained in a binary form (Linux or Microsoft Windows operating system) and in a source form.

In the binary form it should be copied to a directory where execution is permitted and preferably where the shell will look for executables (directory in the PATH environment variable).

If the problem is in a *stub* form, to solve the problem one must write

```
% nsips stub
```

in the shell prompt (% is the Linux shell prompt).

If the problem is in a module file (SIPAMPL database [47]) one should write

```
% ampl problem.mod
```

where *problem* is the problem name under the SIPAMPL database.

The same comments apply to the Windows version.

In the source form the file `nsips.tgz` should be uncompressed (`tar xvzf nsips.tgz`) in the same directory as the SIPAMPL database. Figure 3.1 presents a typical directory structure for the SIPAMPL database and NSIPS.

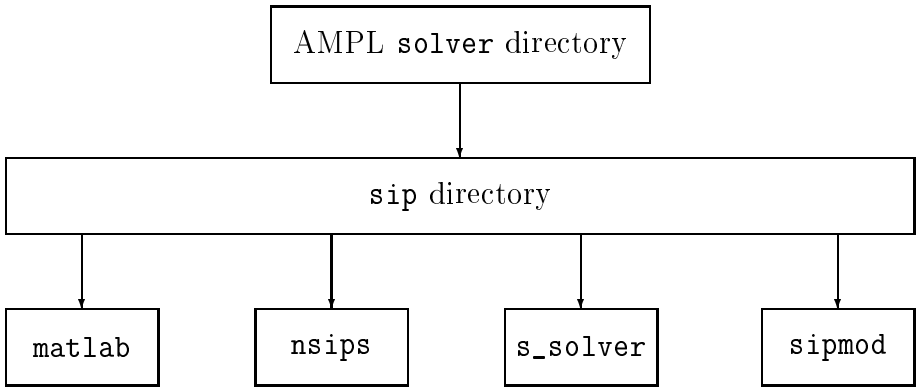

Figure 3.1: Organization of SIPAMPL and NSIPS directories

The NSIPS source package provides three makefiles. Two makefiles for the Linux Operating System: `makefile.nps` which compiles the NSIPS with the NPSOL (links NSIPS with `libopt.a`) and `makefile.std` which produces the version of NSIPS (without NPSOL). To use the makefiles you can either

- copy `makefile.xxx` to `makefile` and just run `make`, or
- run `make -f makefile.xxx`

being `xxx` the appropriate extension for the makefile corresponding to the desired version. The makefile `make_std.vc` is for the Windows Operating System with the Microsoft Visual C/C++ compiler which produces the distributed version of NSIPS. To use the makefile you can either

- copy `make_xxx.vc` to `makefile` and just run `nmake`, or
- run `nmake -f make_xxx.vc`

being `xxx` the appropriate string for the makefile corresponding to the desired version.

The NSIPS uses the SIPAMPL and AMPL interface routines (see Figure 3.2). `libsip.a` (`sipampl.lib`) (SIPAMPL interface routines) and `amplsolver.a` (`amplsolv.lib`) (AMPL interface routines) are linked with the NSIPS routines.

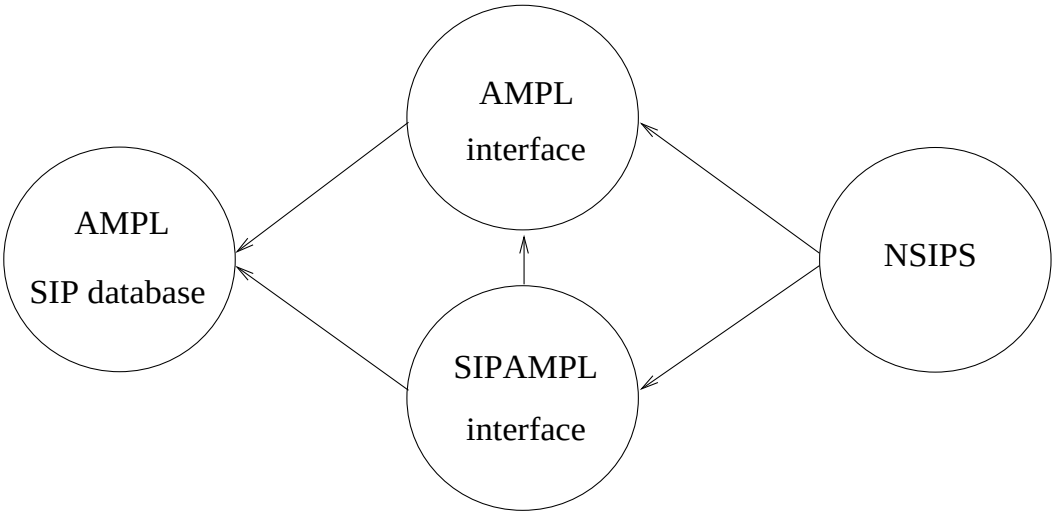

Figure 3.2: NSIPS interconnected with the SIPAMPL interface and AMPL

3.3 Passing options to the solver

The options are passed to the NSIPS in the standard way of passing options to any solver connected with the AMPL. Options to solvers can be given

- as command line arguments,
- in shell environment variables,

- in an options file,
- in the modeling file (AMPL option command).

The following examples will be given with respect to the NSIPS.
In the command line argument, if the problem is in a *stub* form,

```
% nsips stub [option=value]
```

where *option* is the option to set with the value *value*. The [x] brackets means that x is optional and may be repeated as many times as desired.

We can set the `nsips_options` environment variable with the shell (bash) commands

```
% nsips_options='[option=value]'
% export nsip_options
```

In the Windows Operating system the `set` command to change an environment variable gives a syntax error if the sign = is used in the variable declaration. Instead the environment variable `OPTIONS_IN` can be set. AMPL will read the file pointed by this variable before processing the model. The MSDOS command

```
% set OPTIONS_IN=.\nsipsopt.ini
```

will instruct AMPL to read the file `nsipsopt.ini` before reading the problem. The file `nsipsopt.ini` may have additional options to the solver as if they were inside the model file.

The `option` AMPL command can also be used to set the same options

```
option nsips_options '[option=value]';
```

directly in the AMPL command prompt or in the SIPAMPL problem file, before the `solve` command.

3.4 Selecting the method

The default method is the penalty method described in Chapter 5. This solver is the default since it does not need the NPSOL routines.

The method can be changed be using the option `method` where the `value` can be one of

- `disc_hett` for the Hettich modified discretization method (Chapter 4);
- `disc_halt` for the Hettich modified discretization method with the Halton pseudo-random points (Chapter 4);

- `disc_reem` for the Reemtsen modified discretization method (Chapter 4);
- `penalty` for the Penalty method (Chapter 5);
- `penalty_m` for the Penalty method with the Augmented Lagrangian penalty functions (Chapter 5);
- `sqp` for the Sequential Quadratic Programming method (Chapter 6);
- `intp` for the Interior Point method (Chapter 7).

If one wishes to use the Hettich modified method, the environment variable could be used in the following form

```
% nsips_options='method=disc_hett'
% export nsips_options
```

for the Linux Operating system, or

```
% type nsipsopt.ini
option nsips_options 'method=disc_hett';
% set OPTIONS_IN=.\nsipsopt.ini
```

for the Windows Operating system and then invoking the solver.

3.5 Solver output

The coded problems in the SIPAMPL database include some display commands to print the solution found by the solver. One can edit the `.mod` file to obtain the values for the objective function, constraints functions and variables. The output from each method will be described in the corresponding chapter. Each method will append a line to a file (`results`) in the working directory.

Chapter 4

Discretization method

The discretization method is selected with the option

```
nsips_options='method=disc_hett'
```

for the Hettich modified version,

```
nsips_options='method=disc_halt'
```

for the Hettich modified version with Halton pseudo-random points, or

```
nsips_options='method=disc_reem'
```

for the Reemtsen modified version.

Numerical results with the discretization method were presented in [37]. We need to transcribe some parts of [37] in order to present the options to the solver. Section 4.1 presents some notation and definitions required to the description of the implemented algorithms implemented (Section 4.2). In Section 4.3 we describe the options to the discretization method. Section 4.4 shows the solver output.

4.1 Definitions and some Notation

The following definitions are needed to describe the discretization algorithms presented in the next section.

Definition 1 *A grid is a set of the form* $T[h = (h_1, h_2, \ldots, h_p)] = T \cap \{t = (t_1, t_2, \ldots, t_p) : t_i = \alpha_i + jh_i, j = 0, \ldots, n_i; i = 1, \ldots, p\}$, *where* $n_i = (\beta_i - \alpha_i)/h_i$.

Definition 2 *NLP(T[h]) is the following nonlinear programming sub-problem:*

$$\begin{aligned} & \min_{x \in R^n} f(x) \\ s.t.\ \ & g_i(x,t) \leq 0,\ i = 1, \ldots, m \\ & h_i(x) \leq 0,\ i = 1, \ldots, o \\ & h_i(x) = 0,\ i = o+1, \ldots, q \\ & \forall t \in T[h]\ , \end{aligned} \tag{4.1}$$

The following sets are crucial in the algorithms procedure.

$R(x_k)$ denotes the set of all points of the grid at iteration k that makes an infinite constraint active , i.e., $R(x_k) = \{t \in T[h^k] : g_i(x_k, t) = 0\}$.

$S(x_k)$ is the set of all points of the grid at iteration k that makes an infinite constraint active or violated, i.e., $S(x_k) = \{t \in T[h^k] : g_i(x_k, t) \geq 0\}$.

4.2 The Algorithms

Discretization methods try to solve a SIP problem by replacing the infinite set T by a finite one that is, usually, a grid of points. These methods do not guarantee an exact or near exact solution to the SIP but will solve the SIP in the final grid. These methods start with an initial approximation to the solution and in order to converge to an approximate solution the methods solve the problem in a number of intermediate grids. In each iteration the methods refine the grid in a predefined way. To minimize the number of constraints in the sub-problem, the algorithms use only a selected set of points in each grid. Hettich [15, 16] and Reemtsen [29] define two discretization methods, that we have modified to solve a nonlinear SIP problem.

We have also implemented a Hettich version where the grid is replaced by a set of pseudo-random numbers. These pseudo-random numbers are known as an Halton (see [27]) sequence of points. The algorithm will start with a predefined number of pseudo-random points and, in each refinement, a number of points is added until a maximum allowed number of points has been reached.

The Hettich modified algorithm is as follows:

Algorithm 3 *Hettich modified version*

step 0: *Define $T[h^0]$, set $\widetilde{T}[h^0] = T[h^0]$. Solve NLP($\widetilde{T}[h^0]$) and let x_0 be the solution. Define $R = R(x_0)$.*

step k: *If $S(x_{k-1}) \nsubseteq \widetilde{T}[h^{k-1}]$*

then: *Make $\widetilde{T}[h^{k-1}] = R \cup S(x_{k-1})$. Solve NLP($\widetilde{T}[h^{k-1}]$) and let x_{k-1} be the solution. If $R(x_{k-1}) \nsubseteq R$*

then: *Set* $R = R \cup R(x_{k-1})$.

else: *Add one point from* $\widetilde{T}[h^{k-1}] \backslash R$ *to* R.

Continue with Step k.

else: *If* $k > r$ *stop.* $\widetilde{T}[h^k] = R \cup S(x_{k-1}) \cup N(S(x_{k-1}))$. *Solve NLP(*$\widetilde{T}[h^k]$*) and let* x_k *be the solution. Set* $R = R \cup R(x_k)$. *Go to Step k+1.*

where r is the number of refinements and $N(S(x_{k-1}))$ contains the neighbors of points of $S(x_{k-1})$ in the full grid $T[h^k]$ that make an infinite constraint active or violated.

The modified Reemtsen algorithm is presented below with some changes in the notation to meet our requirements.

Algorithm 4 *Reemtsen modified version*

step 0: *Choose* $\widetilde{\varepsilon}_k \in (0,1)$ *and* $\widetilde{\delta}_k \geq 0$ $(k = 1, \ldots, r)$. *Set* $D_0 = T[h^0]$. *Set also* $i = 0$ *and* $k = 1$.

step 1: *Solve* $NLP(D_i)$ *and let* x_i *be the solution.*

step 2: *If k>r stop. Set* $B_{i+1} = T[h^k]$, $\varepsilon_{i+1} = \widetilde{\varepsilon}_k$ *and* $\delta_{i+1} = \widetilde{\delta}_k$.
If x_i *solves the full grid, i.e.,* $g_j(x_i, t) \leq \delta_{i+1}$ *for all* $t \in B_{i+1}$ *and* $j = 1, \ldots, m$

then: *Increment k by one and continue with step 2.*

else: *Set* $D_{i+1} = \{t \in B_{i+1} : g_j(x_i, t) \geq -\varepsilon_{i+1}|f(x_i)|, j = 1, \ldots, m\}$. *Increment i by one and go to step 1.*

$f(x_i)$ is the objective value in x_i (solution of NLP(D_i)).

Two grid refinements are made and they are as Hettich proposed in [15], i.e., $h^1 = h^0/2$, $h^{k+1} = h^k/3$. In Reemtsen version of the algorithm $\widetilde{\epsilon}_k$ is updated by the formula

$$\begin{cases} \widetilde{\epsilon}_1 = \widetilde{\epsilon}_0/(2)^p & \text{for } k = 0 \\ \widetilde{\epsilon}_{k+1} = \widetilde{\epsilon}_k/(3)^p & \text{for } k \neq 0 \end{cases} .$$

4.3 Discretization method options

The discretization method options are presented in Table 4.1. The first column ("Option") presents the ***option*** name. Column "Type" presents the ***value*** that the option can take: "Integer" is an integer value (example: 1, 10, 100); "Double" is a real number in double precision (example: 1.0, 1, 1e+1, -1e-1) and "Doubles" is a comma separated list of "Double" type (example: 0.1,0.2 0.3,0.4 0.1). Column "Default" displays the default value for the option and in the last column a brief description of the option. Whenever the option has a correspondence to the notation in the algorithms, it will be given.

Option	Type	Default	Description
disc_dist	Double	0.1	Two points are neighbors if $\|x - y\|_2 <$disc_dist.
disc_eps	Double	0.01	$\widetilde{\epsilon}_0$.
disc_h	Doubles	0.1	h_i, $i = 1, \ldots, p$.
disc_h_mx	Integer	$100 \times p$	Number of Halton points added in each refinement.
disc_ha_mx	Integer	$1000 \times p$	Maximum number of Halton points.
disc_inner	Integer	100	Maximum of $NLP(\widetilde{T}[h^{k-1}])$ solved for a given k.
disc_k	Integer	3	Number of refinements, r.
zero	Double	10^{-6}	Approximation to zero. Used to find active point and to compare two points, $\widetilde{\delta}_k$ for all k.

Table 4.1: Options for discretization method

4.4 Discretization method output

The discretization method will append the line

& ig & fg & avg & nsub & fx \\

to the output file. "ig" is the number of points in the initial grid; "fg" is the number of points in the final grid; "avg" is the average number of points in the finite sub-problems solved (number of finite constraints) - the number of constraints in the first finite sub-problem solved is not counted; "nsub" is the number of finite sub-problems solved; "fx" is the objective function value in the solution found. The first & is provided to include the problem name and to build a LaTeX tabular environment to produce a table with results. The solver (if executed by the AMPL) has no information about the problem name and can not provide the first tabular field. One can use the following bash shell commands

```
% echo -n "problem" >> ./results
% ampl problem
```

to produce the following tabular complete line

problem & ig & fg & avg & nsub & fx \\

and after including the begin and end of the tabular, the line looks like this

$$\| \text{ problem } | \text{ ig } | \text{ fg } | \text{ avg } | \text{ nsub } | \text{ fx } \| \ .$$

Chapter 5

Penalty method

The penalty method includes two versions, both based on a quasi-Newton method applied to penalty functions. The first method solves the unconstrained problem (based on penalty functions), where no reference to Lagrange multipliers is made. This method is selected with the solver option

`nsips_options='method=penalty'`.

The second method solves the unconstrained problem using an Augmented Lagrangian penalty function or a multiplier penalty function. An estimation of the Lagrange multipliers is obtained through an updating formula. This method is selected with the solver option

`nsips_options='method=penalty_m'`.

Section 5.1 presents an overview on penalty functions. In Section 5.2 we present the finite problem obtained from the semi-infinite problem after applying a constraint transcription. The penalty functions are presented in Section 5.3. The updating formula for the Lagrange multipliers are shown in Section 5.4. The new constraints are rewritten as integrals of the infinite constraints. We show how we have numerically computed these integrals in Section 5.5. Section 5.6 presents the integral computational options. Section 5.7 describes the implemented algorithm, Section 5.8 presents the penalty method options and Section 5.9 presents the penalty method output.

5.1 Overview on penalty functions

We introduce now some notation that will be used in this chapter.

For $z \in R$, let

$$z_+ = max\{0, z\}$$

and

$$sgn(z) = \begin{cases} +1 & \text{if} \quad z > 0 \\ 0 & \text{if} \quad z = 0 \\ -1 & \text{if} \quad z < 0. \end{cases}$$

Teo and Goh in [32] proposed a constraint transcription from problem (2.1) into a non-linear finite problem (NLP) where the infinite inequality constraints are transformed into equality finite constraints of the form:

$$G_i(x) \equiv c \int_T [g_i(x,t)]_+^2 \, dt = 0, \quad i = 1, \ldots, m$$

where c is an empirical weighting factor which is used to improve the numerical accuracy. However the new constraints of the NLP problem do not satisfy the usual constraint qualification, and therefore the convergence, with typical methods for NLP, is not guaranteed. In spite of this Teo and Goh reported successful numerical experience with two examples.

Note 5 *For an exact penalty function there exists $\overline{\mu} > 0$, such that a local minimizer of $\phi(x, \mu)$, $x^*(\mu)$, is a solution of (2.1), x^*, for $\mu > \overline{\mu}$. So, a unique minimization of $\phi(x, \mu)$ is required.*

Conn and Gould in [6] proposed an exact penalty function for semi-infinite programming problems with the following form

$$\phi_{CG}(x, \mu) = f(x) + \mu \sum_{i=1}^{m} \left(\frac{\sum_{j=1}^{s_i} \int_{\Omega_{ij}(x)} g_i(x,t) dt}{\sum_{j=1}^{s_i} \int_{\Omega_{ij}(x)} dt} \right) \tag{5.1}$$

where, for any x, there is a finite set of sets $\Omega_{ij}(x)$ such that

(i) $\Omega_{ij} \subseteq T$, $1 \leq j \leq s_i = s_i(x) < \infty$,

(ii) $g_i(x,t) \geq 0$, $\forall t \in \Omega_{ij}(x)$ and $g_i(x,t) < 0$, $\forall t \in T \setminus \cup_{j=1}^{s_i} \Omega_{ij}(x)$,

(iii) $\Omega_{ij}(x) \cap \Omega_{ik}(x) = \{\emptyset\}$ if $j \neq k$, and

(iv) $\Omega_{ij}(x)$ is connected and non-trivial, i.e., $\int_{\Omega_{ij}(x)} dt > 0$.

The denominator has been included to make the penalty for infeasibility strong enough in order for the penalty function to be exact (see Note 5). The computation of the sets Ω_{ij}, $i = 1, \ldots, m$, $j = 1, \ldots, s_i$ for the penalty function (5.1) evaluation is a major drawback of this definition. Instead the following definition can be used

$$\phi_{CG}(x, \mu) = f(x) + \mu \sum_{i=1}^{m} \frac{\int_T [g_i(x,t)]_+ dt}{\int_T [sgn\,(g_i(x,t))]_+ dt} \tag{5.2}$$

which is in some way equivalent to (5.1) [27].

However the penalty function (5.2) is not smooth and a nondifferentiable or derivative free algorithm must be used. See, for example, Polak [26] for a review on nondifferentiable optimization and Wolfe [52] for the derivative free Powell method.

In a later paper Jennings and Teo [19] showed how problem (2.1) can be replaced by an approximate problem (in a similar form as in [32]) with continuously differentiable constraints in x. The approximate problem is then

$$\begin{aligned} &\min_{x \in R^n} \; f(x) \\ s.t. \;\; &G_{i,\epsilon}(x) \equiv \int_T g_{i,\epsilon}(x,t)dt = 0, \quad i = 1, \ldots, m \end{aligned} \tag{5.3}$$

with

$$g_{i,\epsilon}(x,t) = \begin{cases} 0, & \text{if } \; g_i(x,t) < -\epsilon; \\ (g_i(x,t) + \epsilon)^2 / 4\epsilon, & \text{if } \; -\epsilon \leq g_i(x,t) \leq \epsilon; \\ g_i(x,t), & \text{if } \; g_i(x,t) > \epsilon, \end{cases} \tag{5.4}$$

where ϵ is a small real scalar.

In fact, the equality constraints $G_{i,\epsilon}(x) = 0$ in problem (5.3) do not, again, satisfy the usual constraint qualification and it is not advisable to solve the problem in formulation (5.3). The following approximate problem is then solved

$$\begin{aligned} &\min_{x \in R^n} \; f(x) \\ s.t. \;\; &G_{i,\epsilon}(x) < \tau, \\ &\qquad i = 1, \ldots, m \end{aligned} \tag{5.5}$$

being τ a small real scalar. The routine E04VCF in the NAG library [22] was used, in [19], to solve problem (5.5). ϵ and τ are updated during the process and we will herein omit its description.

Teo *et al.* in [33] developed a new algorithm based on the L_1 exact penalty function technique

$$\phi_1(x, \mu) = f(x) + \mu \sum_{i=1}^{m} \int_T g_{i,\epsilon}(x,t)dt.$$

Other local smoothing technique consists of replacing the $[g_i(x,t)]_+$ term by $\frac{g_i(x,t)+\sqrt{(g_i(x,t))^2+\epsilon^2}}{2}$.

While the smoothing technique used by Teo is only C^1 ($g_{i,\epsilon}(x,t)$ is not twice continuously differentiable in $-\epsilon$ and ϵ) the one presented here is C^∞ everywhere.

5.2 Problem with constraints transcription

We will use the following problem to approximate (2.1)

$$\begin{aligned} &\min_{x\in R^n} f(x) \\ s.t. \quad &\int_T [g_i(x,t)]_+ \, dt \le \tau, \ i = 1,\dots,m. \end{aligned} \tag{5.6}$$

This problem will satisfy the usual constraint qualification.

The Lagrangian of problem (5.6) is

$$\mathcal{L}(x,\lambda) = f(x) + \sum_{i=1}^{m} \lambda_i \left(\int_T [g_i(x,t)]_+ \, dt - \tau \right) \tag{5.7}$$

and

$$\nabla_x \mathcal{L}(x,\lambda) = \nabla_x f(x) + \sum_{i=1}^{m} \lambda_i \int_T [\nabla_x g_i(x,t)]_+ \, dt \tag{5.8}$$

where

$$[\nabla_x g_i(x,t)]_+ = \begin{cases} \nabla_x g_i(x,t) & \text{if } \ g_i(x,t) > 0 \\ 0 & \text{if } \ g_i(x,t) \le 0 \end{cases} \tag{5.9}$$

5.3 The penalty functions

In this section we describe the implemented penalty functions. We applied primal methods to the various penalty functions.

In the use of primal methods one tries to solve

$$\min_{x\in R^n} \phi_S(x,\mu), \min_{x\in R^n} \phi_{AL}(x,\lambda,\mu) \ \text{ or } \ \min_{x\in R^n} \phi_{EMP}(x,\lambda,\mu) \tag{5.10}$$

where $\phi_S(x,\mu)$, $\phi_{AL}(x,\lambda,\mu)$ and $\phi_{EMP}(x,\lambda,\mu)$ are

- Simple penalty functions

$$\phi_S^1(x,\mu) = f(x) + \mu \sum_{i=1}^{m} \int_T g_{i,\epsilon}(x,t) dt \tag{5.11}$$

$$\phi_S^2(x,\mu) = f(x) + \frac{\mu}{2}\sum_{i=1}^{m}\int_T g_{i,\epsilon}(x,t)^2 dt \tag{5.12}$$

and

$$\phi_S^3(x,\mu) = f(x) + \mu\sum_{i=1}^{m}\int_T \left(e^{g_{i,\epsilon}(x,t)} - 1\right) dt \tag{5.13}$$

- A penalty function based on the Augmented Lagrangian with updating formula (multiplier method) for estimated Lagrange multipliers

$$\begin{aligned}\phi_{AL}(x,\lambda,\mu) =& f(x) + \sum_{i=1}^{m}\lambda_i\left(\int_T g_{i,\epsilon}(x,t)dt - \tau\right)\\ &+ \frac{\mu}{2}\sum_{i=1}^{m}\left(\int_T g_{i,\epsilon}(x,t)dt\right)^2\end{aligned} \tag{5.14}$$

  where $\lambda = (\lambda_1,\ldots,\lambda_m)^T$ is the Lagrange multiplier vector.

- An exponential penalty function based on a multiplier method

$$\phi_{EMP}(x,\lambda,\mu) = f(x) + \frac{1}{\mu}\sum_{i=1}^{m}\lambda_i\left(e^{\mu\left(\int_T g_{i,\epsilon}(x,t)dt-\tau\right)} - 1\right) \tag{5.15}$$

5.4 Updating the multipliers

During the optimization procedure the optimal Lagrange multipliers vector is not known and so an updating formula for the multipliers is needed.

The multiplier updating formula for the augmented Lagrangian (5.14) is

$$\lambda_i^{k+1} = \lambda_i^k + \mu\int_T g_{i,\epsilon}(x^k,t)dt, \quad i = 1,\ldots,m. \tag{5.16}$$

where k is the iteration counter.

The updating formula for the exponential penalty function (5.15) is

$$\lambda_i^{k+1} = \lambda_i^k e^{\mu\left(\int_T \nabla_x g_{i,\epsilon}(x^k,t)dt-\tau\right)}, \quad i = 1,\ldots,m. \tag{5.17}$$

5.5 Evaluating the penalty function

The integrals in the penalty functions are numerically computed. We use adaptative trapezoid or Gaussian formulae.

A trapezoid formula to evaluate a simple integral on $T = [a, b]$ based on two points is

$$\int_a^b h(x)dx \approx \frac{b-a}{2}\left[h(a) + h(b)\right].$$

The evaluation of the integral with a given precision ϵ is done recursively in the following way. We start by computing the integral using a two point trapezoid formula in the set $[a, b]$ and in the subsets $\left[a, \frac{a+b}{2}\right]$, $\left[\frac{a+b}{2}, b\right]$. If the error between the approximations is greater then ϵ_T, i.e.,

$$\left| \int_a^b h(x)dx - \left(\int_a^{\frac{a+b}{2}} h(x)dx + \int_{\frac{a+b}{2}}^b h(x)dx \right) \right| > \epsilon_T$$

then we proceed recursively in the subsets $\left[a, \frac{a+b}{2}\right]$ and $\left[\frac{a+b}{2}, b\right]$. Otherwise we accept $\int_a^{\frac{a+b}{2}} h(x)dx + \int_{\frac{a+b}{2}}^b h(x)dx$ as a good approximation to the integral.

In this way the function evaluations are mostly required where the function is less smooth.

This adaptative idea can also be used with a Gaussian formula. In spite of the Gaussian formula having in general a better accuracy, with some careful programming the trapezoid formula requires less function evaluations, since all function evaluations in the Gaussian formula are discarded if a subdivision is needed in a given set.

In figure 5.1 the first infinite constraint for *hettich2* SIPAMPL [36] problem is used to illustrate the integral evaluation by a Gaussian adaptative formula. The solid line represents the function

$$\left[-\left(t^2 - \left(p_1 t + p_2 e^t\right)\right) - d\right]_+$$

with $p_1 = 0.675751$, $p_2 = 0.285805$, $d = 0.536671$ and $t \in [0, 2]$. The $+$ signs are the points where the set $[0, 2]$ was splited with $*$ the corresponding function value. Please note that in the set $[1, 2]$ the function is constant and only three function evaluations are needed.

In the adaptative trapezoid formula the initial interval $[a, b]$ is divided into a predefined number of sub-intervals ni and a recursive formula is applied in each subset $\left[a + i * \frac{b-a}{ni}, a + (i+1) * \frac{b-a}{ni}\right]$, with $i = 0, \ldots, ni - 1$.

The number of function evaluations is not limited, but the amplitude of the subset is. The interval $[x, y]$ will not be splited if $y - x < \rho$.

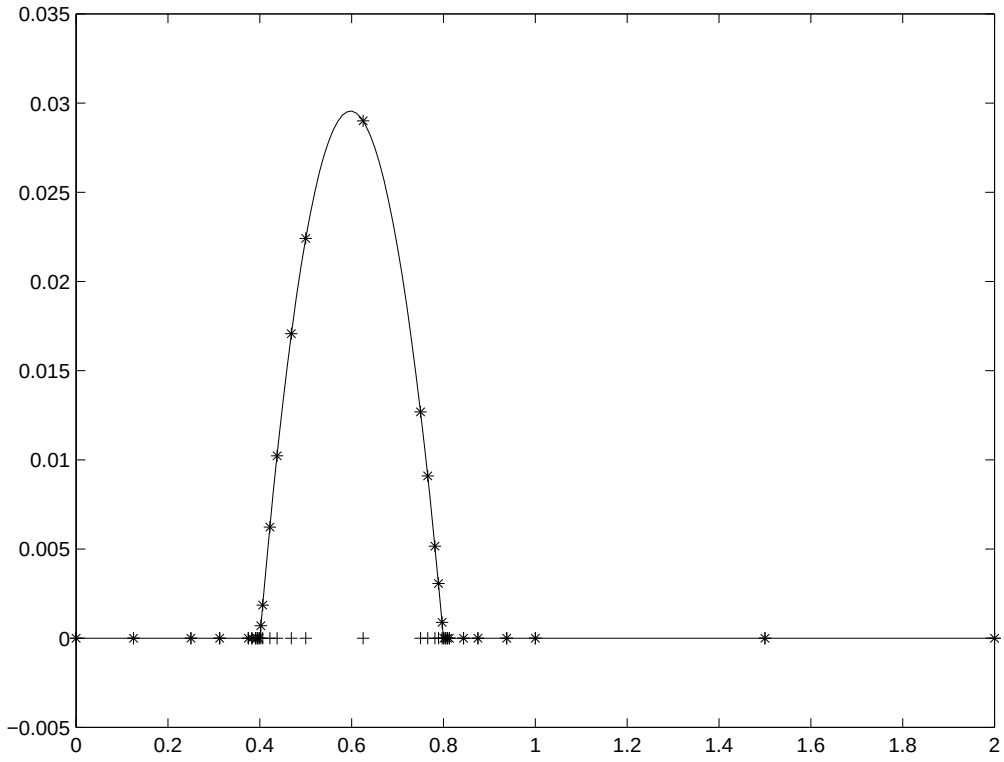

Figure 5.1: Gaussian adaptative formula in first infinite constraint for *hettich2* SIPAMPL problem

5.6 Options on integral computation

Option

`pf_int=gaussian`

selects the Gaussian adaptative formula (each sub-interval estimation is done by a Gaussian formula). Option

`pf_int=trapezoid`

selects the Trapezoid adaptative formula (each sub-interval estimation is done by a Trapezoid formula). Table 5.1 presents other options for tunning the integral computation.

The columns in Table 5.1 have the same meaning as in Table 4.1.

Option	Type	Default	Description
int_amp	Double	10^{-2}	Minimum integral amplitude ρ
int_error	Double	10^{-8}	Precision in integral computation ϵ_T
int_n	Integer	20	Number of initial sub-intervals in adaptative integration ni, or number of Gaussian points in Gaussian adaptative formula. 6, 8, 16 are the allowed number of Gaussian points.

Table 5.1: Options for integral computation

5.7 The algorithm

For the primal method

$$\min_{x \in R^n} \phi_S(x, \mu) \quad , \quad \min_{x \in R^n} \phi_{AL}(x, \lambda, \mu) \quad \text{or} \quad \min_{x \in R^n} \phi_{EMP}$$

the sequential penalty algorithm used was a quasi-Newton (QN) type algorithm based on the BFGS updating formula.

A description of the algorithm follows.

Algorithm 6 *(Primal method)*

Step (a) Given x_0, μ, ϵ δ_1, δ_2 *and* λ *(if* ϕ *is* ϕ_{AL} *or* ϕ_{EMP}*). Let* $i = 0$, $j = 0$, $k = 0$, $y_0 = x_0$.

Step (b) Outer iteration. Let $x_0 = y_k$.

Method	pf_type	Penalty
penalty	p1	(5.11)
	p2	(5.12)
	p3	(5.13)
penalty_m	p1	(5.14)
	p3	(5.15)

Table 5.2: Relation between the options and the penalty function selected

Step (c) *Inner iteration. Update H by the BFGS formula (if $i = 0$ then H = Identity matrix). Let $d_i = -H\nabla_x\phi$.*

Step (d) *Let α_i be the step size computed by an Armijo like rule that sufficiently decreases the penalty function ϕ.*

Step (e) *Set $x_{i+1} = x_i + \alpha_i d_i$.*

Step (f) *If there is not a significant evolution from x_i to x_{i+1} $\left(\frac{\|x_{i+1}-x_i\|}{\|x_{i+1}\|} < \delta_1\right)$ then set $k = k+1$, $y_k = x_{i+1}$, $i = 0$, and go to Step (g). Else set $i = i+1$ and go to Step (c).*

Step (g) *If $\int_T g_{i,\epsilon}(y_k, t)dt \neq 0$ then update λ (if ϕ is ϕ_{AL} or ϕ_{EMP}), μ and go to Step (b). Else, if $j > 0$ and there is not a significant evolution from y_{eps} and y_k $\left(\frac{\|y_{eps}-y_k\|}{\|y_k\|} < \delta_2\right)$ then stop with y_k as an approximatted solution. Else set $j = j+1$, $y_{eps} = y_k$, update ϵ and go to Step (b)*

5.8 Penalty method options

The option

```
pf_type=[p1|p2|p3]
```

will select the penalty function to be used. Table 5.2 presents the relation between the options `p1`, `p2` and `p3`.

The options to the penalty method are presented in Table 5.3.

The columns in Table 5.3 have the same meaning as in Table 4.1.

Option	Type	Default	Description
armijo	Double	10^{-1}	Constant η in Armijo rule
damped	Integer	1	0 if no damped BFGS updating formula is used. For any other integer value the damped formula is used. See [25]
maxiteri	Integer	400	Maximum allowed number of inner iterations
maxitero	Integer	400	Maximum allowed number of outer iterations
mu0	Double	1	Initial approximation to the penalty parameter μ_0
muf	Double	10	Multiplying factor for the penalty parameter
pf_preci	Double	10^{-4}	Inner iteration stopping criteria δ_1
pf_preco	Double	10^{-4}	Outer iteration stopping criteria δ_2
pf_eps	Double	10^{-4}	Initial smoothing parameter for differentiability ϵ_0
reset	Integer	0	0 if no reset of the estimation of the Hessian inverse if requested. Any integer value for reset.
scale	Integer	0	0 if no scaling is wanted. Any value otherwise. If the norm of the direction is not in range $[10^{-2}, 10^2]$ the direction will be scaled to fit the values.

Table 5.3: Options for penalty method

5.9 The penalty method output

The penalty method will append the line

& ti & no & npf & ngpf & fx & miu & eps \\

to the output file. "ti" is the number of total inner iterations; "no" is the number of outer iterations; "npf" is the number of penalty function evaluations; "ngpf" is the number of penalty gradient evaluations; "fx" is the objective function value in the solution found; "miu" is the last penalty parameter used and "eps" is the last ϵ used (differentiability parameter).

The same comment of Section 4.4 about building a LaTeX tabular environment applies.

Chapter 6

Sequential quadratic programming method

The sequential quadratic method is selected with the option

`nsips_options='method=sqp'`.

Section 6.1 introduces the SQP technique. The approximation to the dual functions and the dual problem are described in Section 6.2. The merit function used to measure progress toward the solution is described in Section 6.3. Section 6.4 presents the full implemented algorithm. The options to the SQP methods are shown in Section 6.5 and Section 6.6 shows the SQP method output.

6.1 Sequential quadratic programming technique

The Lagrangian function associated with problem (2.1) can be equated in the following way

$$\mathcal{L}(x, v) = f(x) + \sum_{j=1}^{m} \int_a^b g_j(x, t) dv_j(t) \tag{6.1}$$

where the integrations are Lebesgue-Stieltjes (see for example [51]) integrals with measures induced by $v(t)$ (vector function $v(t) = (v_1(t), \ldots, v_m(t))^T$).

The first derivative of the Lagrangian function (5.7) with respect to x can be obtained by the following formula

$$\nabla_x \mathcal{L}(x, v) = \nabla f(x) + \sum_{j=1}^{m} \int_a^b \nabla_x g_j(x, t) dv_j(t) \tag{6.2}$$

Given an x_k (x at iteration k), a local quadratic approximation to problem (2.1), here denoted by quadratic semi-infinite programming (QSIP) is:

$$\begin{aligned} \min_{d \in R^n} \; & f_Q(d) = \frac{1}{2} d^T H_k d + d^T \nabla f(x_k) \\ s.t. \; & d^T \nabla_x g_j(x_k, t) + g_j(x_k, t) \leq 0, \\ & j = 1, \ldots, m, \; \forall t \in [a, b] \; , \end{aligned} \tag{6.3}$$

where H_k is a symmetric positive definite matrix, $\nabla f(x_k)$ is the gradient of the objective function and $\nabla_x g_j(x_k, t)$, $j = 1, \ldots, m$ are the gradients of the infinite constraints, with respect to the x variable.

The sequential quadratic semi-infinite programming (SQSIP) technique solves a sequence of QSIP problems (6.3) for d_k, the search direction used to compute a new approximation, $x_{k+1} = x_k + \alpha_k d_k$, to the solution of the original problem (2.1), where α_k is obtained through a line search procedure.

The SQSIP technique is described in the following algorithm.

Algorithm 7 *SQSIP algorithm*

Step (a) Given x_0. Let $k = 0$.

Step (b) Compute H_k.

Step (c) Solve the QSIP problem to obtain the search direction d_k.

Step (d) If $d_k = \mathbf{0}$ then stop.

Step (e) Line search. Find α_k such that $x_{k+1} = x_k + \alpha_k d_k$ gives a sufficient decrease in a merit function.

Step (f) If there is not a sufficient difference from x_{k+1} and x_k then stop with x_{k+1} as an approximated solution. Else let $k = k + 1$ and go to Step (b).

6.2 Solving the QSIP problem

Some of the theory presented in [23] will be used to show how the QSIP problem is solved.

The Lagrangian function associated with problem (6.3) is

$$\mathcal{L}(d, v) = \frac{1}{2} d^T H_k d + d^T \nabla f(x_k) + \sum_{i=1}^{m} \int_a^b \left(d^T \nabla_x g_j(x_k, t) + g_j(x_k, t) \right) dv_j(t) \tag{6.4}$$

where once again the integrations are Lebesgue-Stieltjes integrals with measures induced by $v(t)$.

Let $\mathcal{V}[a,b;R^m]$ be the space of all normalized bounded variation functions from $[a,b]$ to R^m, which are right-continuous on (a,b) and vanish at $t=a$. Let $\mathcal{V}^*$ be the set of all the nondecreasing functions of $\mathcal{V}[a,b;R^m]$.

Assumption 8 *The interior of the feasible set of problem* (6.3) *is nonempty.*

Lemma 9 *If H_k is positive definite then problem* (6.3) *has a unique solution d^*.*

The following theorem follows from the Duality Theory.

Theorem 10 *(Theorem 3.2 in [23]) Let d^* be the unique solution to problem* (6.3). *Then*

$$f_Q(d^*) = \max_{v\in\mathcal{V}^*} \min_{d\in R^n} \mathcal{L}(d,v) \tag{6.5}$$

and the maximum on the right hand side of (6.5) *is achieved at some $v^*\in\mathcal{V}^*$. Furthermore,*

$$f_Q(d^*) = \min_{d\in R^n} \mathcal{L}(d,v^*)\ . \tag{6.6}$$

For a given $v\in\mathcal{V}[a,b;R^m]$ the unique solution $d(v)$ to the problem

$$\min_{d\in R^n}\ \mathcal{L}(d,v) \tag{6.7}$$

is given by

$$d(v) = -H_k^{-1}\left(\nabla f(x_k) + \sum_{j=1}^{m}\int_a^b \nabla_x g_j(x_k,t)dv_j(t)\right) \tag{6.8}$$

which can be easily seen by solving the equation

$$\nabla_d \mathcal{L}(d,v) = 0\ .$$

The right hand side of (6.5) can be written as a dual problem

$$\min_{v\in\mathcal{V}^*}\ \mathcal{L}^*(v) \tag{6.9}$$

where $\mathcal{L}^*(v)$ is the dual functional given by

$$\begin{aligned}\mathcal{L}^*(v) =& -\mathcal{L}(d(v),v)\\ =&\frac{1}{2}\left(\nabla f(x_k) + \sum_{j=1}^{m}\int_a^b \nabla_x g_j(x_k,t)dv_j(t)\right)^T H_k^{-1}\\ &\left(\nabla f(x_k) + \sum_{j=1}^{m}\int_a^b \nabla_x g_j(x_k,t)dv_j(t)\right)\\ &-\sum_{j=1}^{m}\int_a^b g_j(x_k,t)dv_j(t)\end{aligned} \tag{6.10}$$

As the dual variables v in (6.9) are measures of the integration, to solve problem (6.9) an approximation to $v(t)$ by piecewise linear functions in $[a,b]$ as considered. Let $\mathcal{F}^* \subset \mathcal{V}^*$ be the set of all these functions. Let

$$a = t_0 < t_1 < \cdots < t_l < t_{l+1} = b$$

be a partition of the set $[a,b]$. Let the vector function $v(t) \in \mathcal{F}_l^* \subset \mathcal{F}^*$ be defined by

$$v_j(t) = \begin{cases} w_{1j}(t-a), & \text{for } t \in [a,t_1)\,; \\ a_{ij} + w_{i+1j}(t-t_i), & \text{for } t \in [t_i,t_{i+1})\,, i = 1,2,\ldots,l-1; \\ a_{lj} + w_{l+1j}(t-t_l), & \text{for } t \in [t_l,b]\,; \end{cases} \tag{6.11}$$

$j = 1,\ldots,m$, where

$$a_{ij} = \sum_{p=1}^{i} h_{pj} + \sum_{p=1}^{i} w_{pj}(t_p - t_{p-1}),\ i = 1,\ldots,l \tag{6.12}$$

being t_i, $i = 1,\ldots,l$ the partition points, h_{ij}, $i = 1,\ldots,l$, $j = 1,\ldots,m$ the discontinuity jumps and w_{ij}, $i = 1,\ldots,l+1$, $j = 1,\ldots,m$ line segment slopes (see Figure 6.1).

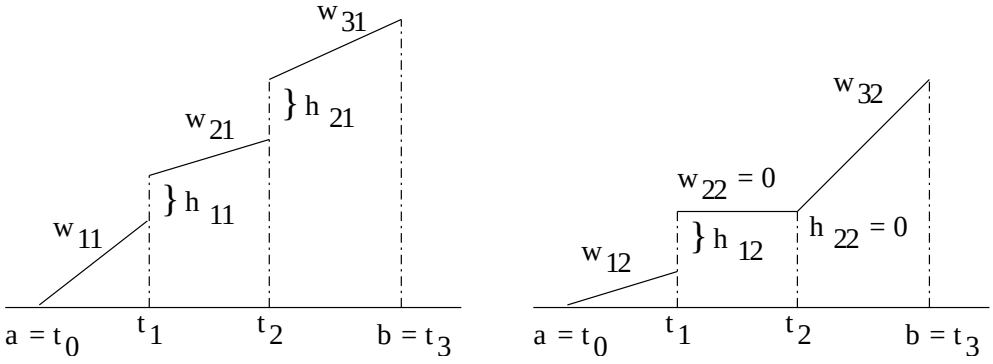

Figure 6.1: Linear segments to approximate $v(t)$ with $l = 2$ and $m = 2$.

With approximation (6.11) to $v(t)$ and the properties of Lebesgue-Stieltjes integration, the integrals in (6.10) can be approximated by the following formulae

$$\int_a^b \nabla_x g_j(x,t)dv_j(t) = \sum_{i=1}^{l} \nabla_x g_j(x,t_i)h_{ij} + \sum_{i=1}^{l+1} w_{ij} \int_{t_{i-1}}^{t_i} \nabla_x g_j(x,t)dt \tag{6.13a}$$

and

$$\int_a^b g_j(x,t)dv_j(t) = \sum_{i=1}^{l} g_j(x,t_i)h_{ij} + \sum_{i=1}^{l+1} w_{ij} \int_{t_{i-1}}^{t_i} g_j(x,t)dt \tag{6.13b}$$

and the solution of problem (6.9) by

$$\min_{v \in \mathcal{F}_l^*} \mathcal{L}_l^*(v) \tag{6.14}$$

with l suitable large, where $\mathcal{L}_l^*(v)$ is given by

$$\begin{aligned} \mathcal{L}_l^*(v) =& \frac{1}{2} \left(\nabla f(x_k) + \sum_{j=1}^{m} c_j \right)^T H_k^{-1} \left(\nabla f(x_k) + \sum_{j=1}^{m} c_j \right) \\ & - \sum_{j=1}^{m} \left(\sum_{i=1}^{l} g_j(x_k, t_i) h_{ij} + \sum_{i=1}^{l+1} w_{ij} \int_{t_{i-1}}^{t_i} g_j(x_k, t) dt \right) \end{aligned} \tag{6.15}$$

with

$$c_j = \sum_{i=1}^{l} \nabla_x g_j(x_k, t_i) h_{ij} + \sum_{i=1}^{l+1} w_{ij} \int_{t_{i-1}}^{t_i} \nabla_x g_j(x_k, t) dt.$$

See [23] for details.

Note 11 *In the problem* (6.14) *the constraint* $v(t) \in \mathcal{F}_l^*$ *is equivalent to the following constraints*

$$\begin{cases} w_{ij} \geq 0 & i = 1, \ldots, l+1, \quad j = 1, \ldots, m \\ h_{ij} \geq 0 & i = 1, \ldots, l, \quad j = 1, \ldots, m \\ t_i \leq t_{i+1} & i = 0, \ldots, l. \end{cases} \tag{6.16}$$

Note 12 *This new problem based on* (6.15) *and* (6.16) *has* $m + lm + (l+1)m$ *variables,* $(l+1)m + lm + 2$ *simple bound constraints and* $l - 1$ *linear constraints and is no longer quadratic.*

6.3 Merit function

We measure progress to the solution of the SIP problem by means of the following merit function

$$\phi(x, \mu) = f(x) + \frac{1}{2} \mu \sum_{j=1}^{m} \int_a^b [g_j(x, t)]_+^2 \, dt. \tag{6.17}$$

where μ is a positive parameter and $[c]_+ = max\{c, 0\}$.

We use the following procedure to compute the penalty parameter μ_{k+1} which ensures that d from (6.8) is a descent direction for (6.17), at x_k.

Algorithm 13 *Computation of the penalty parameter*

Step (a) *Compute* $-d^T H_k d - \sum_{j=1}^{m} \int_a^b d^T \nabla_x g_j(x_k, t) dv_j(t)$. *If it is negative then set* $\mu_{k+1} = \mu_k$, *go to Step* (d).

Step (b) *Compute*

$$\sum_{j=1}^{m} \int_a^b [g_j(x_k, t)]_+^2 \, dt. \tag{6.18}$$

If (6.18) *is zero, as a safeguarded procedure take* $d = -\nabla_x \phi(x_k, \mu_k)$ *and set* $\mu_{k+1} = \mu_k$, *go to Step* (d).

Step (c) *Set*

$$\mu_{k+1} = 10 \max \left\{ \frac{-d^T H_k d - \sum_{j=1}^{m} \int_a^b d^T \nabla_x g_j(x_k, t) dv_j(t)}{\sum_{j=1}^{m} \int_a^b [g_j(x_k, t)]_+^2 \, dt}, \mu_k \right\} \tag{6.19}$$

Step (d) *Proceed.*

The step size α_k used is the first member of the sequence $\{1, \beta, \beta^2, \dots\}$, $\beta = 0.5$, that satisfies the Armijo rule

$$\phi(x_k + \alpha_k d, \mu_{k+1}) \leq \phi(x_k, \mu_{k+1}) + \eta \alpha_k d^T \nabla_x \phi(x_k, \mu_{k+1}), \tag{6.20}$$

with $\eta \in (0, 1)$, fixed.

6.4 The complete algorithm

We are now in position to present the full implemented algorithm.

Algorithm 14 *SQSIP full algorithm*

Step (a) *Given* x_0, δ_1 *and* δ_2. *Let* $k = 0$ *and* $l = 1$.

Step (b) *Update* H_k^{-1} *by a BFGS formula (if* $k = 0$ *then* $H_k =$*Identity matrix).*

Step (c) *Let* $b = \nabla f(x_k)$.

Step (d) Solve the following problem

$$\min_{\substack{t_i,h_{ij},\ i=1,\ldots,l \\ w_{ij},\ i=1,\ldots,l+1 \\ j=1,\ldots,m}} \mathcal{L}_l^*(t_i,h_{ij},w_{ij}) = \frac{1}{2}\left(b+\sum_{j=1}^{m} c_j\right)^T H_k^{-1}\left(b+\sum_{j=1}^{m} c_j\right) - \sum_{j=1}^{m}\left(\sum_{i=1}^{l} g_j(x_k,t_i)h_{ij} + \sum_{i=1}^{l+1} w_{ij}\int_{t_{i-1}}^{t_i} g_j(x_k,t)dt\right) \tag{6.21}$$

$$s.t.$$

$$w \geq 0, \quad h \geq 0, \quad t_{i+1}-t_i \geq 0, \quad i=0,\ldots,l$$

with

$$c_j = \sum_{i=1}^{l} \nabla_x g_j(x_k,t_i)h_{ij} + \sum_{i=1}^{l+1} w_{ij}\int_{t_{i-1}}^{t_i} \nabla_x g_j(x_k,t)dt$$

to obtain the $v(t)$ unknowns.

Step (e) Obtain the search direction $\tilde{d}_l$ from

$$\tilde{d}_l(t_i,h_{ij},w_{ij}) = -H_k^{-1}\left(b+\sum_{j=1}^{m} c_j\right) \tag{6.22}$$

where t, h and w define the solution to problem (6.21) at iteration l.

Step (f) If there is a sufficient difference from $\tilde{d}_{l-1}$ and $\tilde{d}_l$ ($\frac{\|\tilde{d}_{l-1}-\tilde{d}_l\|_2}{\|\tilde{d}_l\|_2} > \delta_1$) and $l < n$, then let $l = l+1$ and go to Step (d). Else stop with $d_k = \tilde{d}_l$ as an appropriate solution to problem (6.3).

Step (g) if $d_k = \mathbf{0}$ then stop with x_k as the solution.

Step (h) Penalty parameter. Compute μ_{k+1} using the Algorithm 13.

Step (i) Line search. Find α_k the first element of the sequence $\{1,\beta,\beta^2,\ldots\}$ that satisfies the Armijo rule (6.20).Set $x_{k+1} = x_k + \alpha_k d_k$.

Step (j) If there is not a sufficient difference from x_{k+1} and x_k ($\frac{\|x_{k+1}-x_k\|_2}{\|x_{k+1}\|_2} < \delta_2$) then stop with x_{k+1} as an approximated solution. Else if $l = n$ go to Step (b), else let $l = \max\{1, l-1\}$ and go to Step (b).

The initial guess to the problem (6.21) is the following

$$\begin{cases} t_i = i\frac{b-a}{l+1}, & i=1,\ldots,l \\ h_{ij} = 1.0, & i=1,\ldots,l, \quad j=1,\ldots,m \\ w_{ij} = 1.0, & i=1,\ldots,l+1, \quad j=1,\ldots,m \end{cases} \tag{6.23}$$

Since the solutions of the several solved QSIP subproblems approximate (by linear segments) the multiplier functions ($v_j(t)$, $j = 1, \ldots, m$) of problem (2.1) the algorithm implemented has the ability to use the previous solution (for a given l) as initial guess to the next QSIP sub-problem.

Problem (6.21) was solved with the NPSOL software package. With a low integral computation accuracy the use of finite differences to approximate the first derivatives of $\mathcal{L}_l^*$ with respect to the variables t, h, and w is not recommended. We have coded the first derivatives instead.

For the matrix H_k we have implemented a BFGS updating scheme to approximate the Hessian of the Lagrangian function, $\nabla^2_{xx}\mathcal{L}(x_k, v)$.

Another issue concerning problem (6.21) is the one dimensional integral computations. Since the intervals $[t_{i-1}, t_i]$ may be small and to reduce the number of integral computations, the $l + 1$ integrals

$$w_{ij} \int_{t_{i-1}}^{t_i} \nabla_x g_j(x_k, t)dt \quad \text{and} \quad w_{ij} \int_{t_{i-1}}^{t_i} g_j(x_k, t)dt \tag{6.24}$$

are replaced by

$$\int_a^b \omega_j(t)\nabla_x g_j(x_k, t)dt \quad \text{and} \quad \int_a^b \omega_j(t) g_j(x_k, t)dt \tag{6.25}$$

respectively where

$$\omega_j(t) = w_{ij} \;\text{ if }\; t \in [t_{i-1}, t_i]\,. \tag{6.26}$$

We approximated the integrals (6.25) by a adaptative trapezoid formula as in [40] (see Section 5.5).

6.5 The SQP method options

In the computation of the integrals the options of Section 5.6 are still valid for tuning the integral computation. The options to the SQP method are presented in Table 6.1.

The columns in Table 6.1 have the same meaning as in Table 4.1.

6.6 The SQP method output

The SQP method will append the line

& no & nmf & ngmf & nc & ngc & fx \\

to the output file. “no” is the number of outer iterations; “nmf” is the number of merit function evaluations; “ngmf” is the number of merit gradient evaluations; “nc” is the number

Option	Type	Default	Description
armijo	Double	10^{-1}	Constant η in Armijo rule.
damped	Integer	1	0 if no damped BFGS updating formula is used. For any other integer value the damped formula is used. See [25].
maxiteri	Integer	400	Maximum allowed number of inner iterations.
maxitero	Integer	400	Maximum allowed number of outer iterations.
pf_preci	Double	10^{-4}	Inner iteration stopping criteria δ_1.
pf_preco	Double	10^{-4}	Outer iteration stopping criteria δ_2.
reset	Integer	0	0 if no reset of the estimation of the Hessian inverse if requested. Any integer value for reset.
scale	Integer	0	0 if no scaling is wanted. Any value otherwise. If the norm of the direction is not in range $[10^{-2}, 10^2]$ the direction will be scaled to fit the values.
dual_ini	Integer	1	0 if no reset on the initial guess to the QSIP is wanted. Any other value otherwise (always use the initial guess in (6.23)).

Table 6.1: Options for the SQP method

of constraint evaluations; "ngc" is the number of constraint gradient evaluations and "fx" is the objective function value in the solution found.

The same comment of Section 4.4 about building a LaTeX tabular environment applies.

Chapter 7

Interior Point method

The interior point method is selected with the option

`nsips_options='method=intp'`.

Section 7.1 presents the interior point paradigm. Section 7.2 is devoted to the implementation details and Section 7.3 presents the full implemented algorithm. Section 7.4 presents the method options and Section 7.5 shows the output results from the method.

Proof of theorems are omitted. The user is referred to [46] for details.

7.1 The interior point method

We will use the following approximate problem to solve (2.1)

$$\begin{aligned} &\min_{x \in R^n} f(x) \\ s.t. \quad &\int_T g_{i,\epsilon}(x,t)dt \leq \tau, \ i = 1, \ldots, m \end{aligned} \tag{7.1}$$

for a positive decreasing sequence of τ values and

$$g_{i,\epsilon}(x,t) = \frac{g_i(x,t) + \sqrt{(g_i(x,t))^2 + \epsilon^2}}{2}. \tag{7.2}$$

We now follow the main ideas in [31, 34, 35].

Adding nonnegative slack variables, ω_i, $i = 1, \ldots, m$, we reformulate problem (7.1) as

$$\begin{aligned} &\min_{x \in R^n, \omega \in R^m} f(x) \\ s.t. \quad &\int_T g_{i,\epsilon}(x,t)dt - \tau + \omega_i = 0 \\ &\omega_i \geq 0, \ i = 1, \ldots, m. \end{aligned} \tag{7.3}$$

being ω the vector with ω_i components.

We eliminate the inequality constraints in (7.3) by placing them in barrier terms in the objective function, resulting in the barrier problem

$$\begin{aligned} \min_{x \in R^n, s \in R^m} \; & f(x) - \mu \sum_{i=1}^{m} \log(s_i + \tau) \\ s.t. \; & \int_T g_{i,\epsilon}(x,t)dt + s_i = 0, \\ & i = 1, \ldots, m \end{aligned} \tag{7.4}$$

where s is the vector of the $s_i = \omega_i - \tau$ variables and $\mu > 0$ is the barrier parameter. The associated Lagrangian function is

$$\mathcal{L}_\mu(x, s, \lambda) = f(x) - \mu \sum_{i=1}^{m} \log(s_i + \tau) - \sum_{i=1}^{m} \lambda_i \left(\int_T g_{i,\epsilon}(x,t)dt + s_i \right) \tag{7.5}$$

and the first-order KKT conditions for a minimum are

$$\nabla_x \mathcal{L}_\mu(x, s, \lambda) = \nabla f(x) - \sum_{i=1}^{m} \lambda_i \int_T \nabla_x g_{i,\epsilon}(x,t)dt = 0 \tag{7.6a}$$

$$\nabla_s \mathcal{L}_\mu(x, s, \lambda) = -\mu S^{-1} e - \lambda = 0 \tag{7.6b}$$

$$\nabla_\lambda \mathcal{L}_\mu(x, s, \lambda) = -\overline{g}(x) - s = 0 \tag{7.6c}$$

where S is a diagonal matrix with elements $s_i + \tau$, $i = 1, \ldots, m$, e is the unit vector, $\overline{g}(x)$ is a vector with elements $\int_T g_{i,\epsilon}(x,t)dt$, s is the vector of the s_i and λ is the vector of the λ_i.

We now modify (7.6) by multiplying equation (7.6b) by S, giving the following primal-dual system

$$\nabla f(x) - \sum_{i=1}^{m} \lambda_i \int_T \nabla_x g_{i,\epsilon}(x,t)dt = 0 \tag{7.7a}$$

$$-\mu e - S\Lambda e = 0 \tag{7.7b}$$

$$-\overline{g}(x) - s = 0 \tag{7.7c}$$

where Λ is a diagonal matrix with elements λ_i.

Applying Newton's method to (7.7) we obtain the following system

$$\begin{pmatrix} H(x,\lambda) & 0 & -J(x) \\ 0 & -\Lambda & -S \\ -J^T(x) & -I & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta s \\ \Delta\lambda \end{pmatrix} = \begin{pmatrix} -\nabla f(x) + \sum_{i=1}^{m} \lambda_i \int_T \nabla_x g_{i,\epsilon}(x,t)dt \\ \mu e + S\Lambda e \\ \overline{g}(x) + s \end{pmatrix} \tag{7.8}$$

where

$$H(x,\lambda) = \nabla^2 f(x) - \sum_{i=1}^{m} \lambda_i \int_T \nabla^2_{xx} g_{i,\epsilon}(x,t)dt, \tag{7.9}$$

$$J(x) = \left(\int_T \nabla_x g_{1,\epsilon}(x,t)dt, \ldots, \int_T \nabla_x g_{m,\epsilon}(x,t)dt \right) \tag{7.10}$$

and $(\Delta x, \Delta s, \Delta\lambda)$ is the Newton direction.

System (7.8) can be rewritten in a shorter form as

$$\begin{pmatrix} H & 0 & -J \\ 0 & -\Lambda & -S \\ -J^T & -I & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta s \\ \Delta\lambda \end{pmatrix} = \begin{pmatrix} \sigma \\ \gamma \\ \rho \end{pmatrix} \tag{7.11}$$

where

$$\sigma = -\nabla f + J\lambda \tag{7.12a}$$

$$\gamma = \mu e + S\Lambda e \tag{7.12b}$$

$$\rho = \bar{g} + s. \tag{7.12c}$$

The vector σ measures dual infeasibility and vector ρ measures primal infeasibility.

Theorem 15 *(Similar to Theorem 1 in [35].)*
If $N = H - JS^{-1}\Lambda J^T$ is nonsingular, system (7.11) has a unique solution given by

$$\Delta x = -N^{-1}\nabla f - \mu N^{-1} J S^{-1} e + N^{-1} J S^{-1} \Lambda \rho \tag{7.13a}$$

$$\Delta s = J^T N^{-1} \nabla f + \mu J^T N^{-1} J S^{-1} e - (I + J^T N^{-1} J S^{-1} \Lambda)\rho \tag{7.13b}$$

$$\Delta\lambda = S^{-1}\Lambda\rho + S^{-1}\Lambda J^T N^{-1}\sigma - S^{-1}\Lambda J^T N^{-1} J S^{-1}\gamma + S^{-1}\Lambda J^T N^{-1} J S^{-1}\Lambda\rho - S^{-1}\gamma. \tag{7.13c}$$

Proof. [46, Theorem 3.1] ■

Given an initial approximation (x^0, s^0, λ^0), the method proceeds iteratively by computing the next approximation in the following way

$$\begin{cases} x^{k+1} = x^k + \alpha^k \Delta x^k \\ s^{k+1} = x^k + \alpha^k \Delta s^k \\ \lambda^{k+1} = \lambda^k + \alpha^k \Delta \lambda^k \end{cases} \tag{7.14}$$

where k is the iteration counter and α^k is the step length selected to ensure that the vectors $s + \tau e$ and λ remain nonnegative and nonpositive respectively, and to guarantee convergence.

7.2 Implementation details

7.2.1 The merit function

For nonlinear optimization problems, a merit function or a filter (see [9] or [3] in the interior point context) should be used to ensure progress toward a local minimizer and feasibility. This progress is achieved selecting the step length along the search direction so that a sufficient reduction in the merit function is obtained.

In [8] the author propose a merit function based in the squared l_2-norm of the residual, but this merit function can drive the algorithm to local minima, maxima or saddle points [31].

We have implemented two merit functions. The l_2 merit function which for problem (7.4) has the following form

$$\phi(x, s; \mu, \beta) = f(x) - \mu \sum_{i=1}^{m} \log(s_i + \tau) + \frac{\beta}{2} \rho^T \rho, \tag{7.15}$$

where ρ is given by (7.12c) and $\beta > 0$ is the penalty parameter. The augmented Lagrangian merit function which for problem (7.4) has the following form

$$\mathcal{L}_A(x, s, \lambda; \mu, \beta) = f(x) - \mu \sum_{i=1}^{m} \log(s_i + \tau) - \lambda^T \rho + \frac{\beta}{2} \rho^T \rho, \tag{7.16}$$

and once again ρ is given by (7.12c) and $\beta > 0$ is the penalty parameter.

We are able to prove that for sufficiently large β, the step defined by (7.8) is a descent direction for the l_2 and the augmented Lagrangian merit function when the problem is strictly convex.

Theorem 16 *If N is positive definite, then there exists $\beta^{\phi}_{min} > 0$, such that the search direction $(\Delta x, \Delta s)$ satisfies*

$$\begin{pmatrix} \nabla_x \phi \\ \nabla_s \phi \end{pmatrix}^T \begin{pmatrix} \Delta x \\ \Delta s \end{pmatrix} \leq 0$$

for every $\beta > \beta^{\phi}_{min}$. The equality holds if and only if (x, s) satisfies (7.7) for some λ.

Proof. [46, Theorem 4.1] ■

Theorem 17 *If N is positive definite, then there exists $\beta^{\mathcal{L}_A}_{min} > 0$, such that the search direction $(\Delta x, \Delta s, \Delta\lambda)$ satisfies*

$$\begin{pmatrix} \nabla_x \mathcal{L}_A \\ \nabla_s \mathcal{L}_A \\ \nabla_\lambda \mathcal{L}_A \end{pmatrix}^T \begin{pmatrix} \Delta x \\ \Delta s \\ \Delta\lambda \end{pmatrix} \leq 0$$

for every $\beta > \beta^{\mathcal{L}_A}_{min}$. The equality holds if and only if (x, s) satisfies (7.7) for some λ.

Proof. [46, Theorem 4.2] ■

In spite of the formula for the β_{min} the algorithm proceeds in the following way to obtain a β such that the search direction is descent for the merit function.

Algorithm 18 *(β computation)*

Step(a) If $k = 0$ then let $\beta^k = 0$.

Step(b) If

$$\begin{pmatrix} \nabla_x \phi \\ \nabla_s \phi \end{pmatrix}^T \begin{pmatrix} \Delta x \\ \Delta s \end{pmatrix} < 0 \quad or \quad \begin{pmatrix} \nabla_x \mathcal{L}_A \\ \nabla_s \mathcal{L}_A \\ \nabla_\lambda \mathcal{L}_A \end{pmatrix}^T \begin{pmatrix} \Delta x \\ \Delta s \\ \Delta\lambda \end{pmatrix} < 0$$

then go to Step(d)*, otherwise go to Step*(c)

Step(c) If $\beta^k = 0$ then set $\beta^k = 0.1$ and go to Step(b)*, else $\beta^k = 10\beta^k$ and go to Step*(b)*.*

Step(d) Proceed with the line search.

7.2.2 The step length selection

Choosing $\alpha^k = 1$ could violate the property that the variables $\omega^k = s^k + \tau e$ and λ^k should remain nonnegative and nonpositive respectively. In the context of the discussion presented in Section 7.1, this means that $s_i^k + \alpha^k \Delta s_i^k$ must remain greater than $-\tau$ and $\lambda_i^k + \alpha^k \Delta \lambda_i^k$ must remain negative, for $i = 1, \ldots, m$. So the inequalities

$$\lambda + \alpha \Delta \lambda \leq 0 \tag{7.17}$$

and

$$s + \alpha \Delta s \geq -\tau e \tag{7.18}$$

imply that the maximum possible value for α is

$$\min \left\{ \frac{-\lambda_i}{\Delta \lambda_i}, \frac{-\tau - s_i}{\Delta s_i} \right\}, \tag{7.19}$$

for every i such that $\Delta \lambda_i > 0$ and $\Delta s_i < 0$.

As a safeguard procedure we select

$$\alpha_{max} = \min \left\{ 1, 0.95 \min \left\{ \frac{-\lambda_i}{\Delta \lambda_i}, \frac{-\tau - s_i}{\Delta s_i} \right\} \right\}, \tag{7.20}$$

to avoid dangerous proximity of the s variables from $-\tau e$.

A backtracking strategy is then implemented in the interval $(0, \alpha_{max}]$ to compute an α^k that gives a reduction in the merit function.

7.2.3 Initial values

Some heuristic could be defined in order to provide initial approximations to the slack and dual variables. The algorithm uses the initial guess for the primal variables proposed by the user. If the user does not provide an initial guess the procedure described in [37] which consists in solving a finite nonlinear problem with the infinite set T replaced by a grid of five equally spaced points is implemented.

As x^0 may lie very close to the boundary of the feasible region and s^0 would be very close to $-\tau e$, we require a $\theta > 0$ so that the initial s is at least as large as θ. So, for the success of the algorithm we set

$$s_i^0 = \max \left\{ \int_T g_{i,\epsilon}(x^0, t) dt, \theta \right\}, \quad i = 1, \ldots, m \tag{7.21}$$

and the dual variables are set to

$$\lambda = [J^T J]^{-1} J^T \nabla f. \tag{7.22}$$

Whenever we update the τ parameter, multiplying it by a reduction factor ξ_τ, we must recompute the slack variables and keep $s_i + \tau > 0$. The equation (7.21) is then used with θ replaced with $\xi_\tau^k \theta$, where k is the iteration counter.

7.2.4 Computing the barrier parameter μ

Using (7.7b), a value of μ could be obtained using $-(s_i+\tau)\lambda_i$, for any i. Alternatively, we can choose μ as

$$\mu = \delta_\mu \frac{-\lambda^T(s+\tau e)}{m}, \tag{7.23}$$

where $0 \leq \delta_\mu < 1$ is used to give a point which is closer to optimality than the current approximation.

7.2.5 The BFGS formula

In [2] the authors introduces and analyzes a BFGS quasi-Newton interior point algorithm. In [1] the strong convexity assumption of one of the objective and constraint functions is relaxed to assume strong convexity of the Lagrangian.

To avoid the computational burden required by the Hessian $H(x,\lambda)$, in (7.9), a quasi-Newton strategy (with the BFGS updating formula) was implemented in the algorithm.

To overcome a possible failure of the curvature condition, a damped strategy was implemented as proposed in [25]. If the damped strategy still does not satisfy the curvature condition, a skipping strategy is used.

7.2.6 Ordering and high ordering

In spite of system (7.11) having a unique solution, assuming that H is positive definite, the several numerical ways of solving it can indeed result in different directions that can affect the algorithm performance. We describe in the following subsections the two main approaches for solving (7.11), primal ordering and dual ordering. A predictor-corrector option is also implemented and may be selected.

Primal ordering

Assuming that H^{-1} is positive definite and $\Lambda^{-1}S$ is diagonal with negative elements we have that $J^TH^{-1}J-\Lambda^{-1}S$ is positive definite. The primal ordering is obtained from system (7.11) by solving the first equation in order to the primal variables. We solve the system for Δx, Δs and $\Delta\lambda$ using:

$$(J^TH^{-1}J - \Lambda^{-1}S)\Delta\lambda = \Lambda^{-1}\gamma - \rho - J^TH^{-1}\sigma \tag{7.24a}$$

by a modified Cholesky factorization,

$$\Delta x = H^{-1}(\sigma + J\Delta\lambda) \tag{7.24b}$$

and

$$\Delta s = -\Lambda^{-1}(S\Delta\lambda + \gamma). \tag{7.24c}$$

Dual ordering

Assuming H to be positive definite and noting from equation (7.7b) that the Lagrange multipliers are negative we have that $H - JS^{-1}\Lambda J^T$ is positive definite. The dual ordering is obtain by solving the system (7.11) for the dual variables in the last equation. We solve the system (7.11) for Δx, Δs and $\Delta\lambda$ using:

$$(H - JS^{-1}\Lambda J^T)\Delta x = -\nabla f + JS^{-1}(\Lambda\rho - \mu e) \tag{7.25a}$$

by a modified Cholesky factorization,

$$\Delta s = -\rho - J^T\Delta x \tag{7.25b}$$

and

$$\Delta\lambda = -S^{-1}(\Lambda\Delta s + \gamma). \tag{7.25c}$$

Predictor corrector

Mehrotra [24] proposed a predictor-corrector method for interior point applied to linear programming. The algorithm consists of computing two directions, the predictor and corrector, in the same iteration, both based on only one factorization. In [5] the authors describe an extension to the Mehrotra predictor-corrector method to multiple corrections in the context of linear and convex quadratic programs and the relation between the composite Newton method and the multiple predictor-corrector methods is established. In [18, 12] the authors study the use of multiple correction steps for the linear case. In [31] the authors describe the primal ordering, dual ordering and the predictor-corrector to the nonlinear case. The predictor-corrector proposed for NLP may not produce a descent direction for the l_2 merit function and so the algorithm proposed switchs to the standard direction if the penalty parameter increases too much or the step length is too short.

The predictor-corrector implemented herein first solves the unperturbed Newton system

$$\begin{pmatrix} H(x,\lambda) & 0 & -J(x) \\ 0 & -\Lambda & -S \\ -J^T(x) & -I & 0 \end{pmatrix} \begin{pmatrix} \Delta x_p \\ \Delta s_p \\ \Delta\lambda_p \end{pmatrix} = \begin{pmatrix} \sigma \\ S\Lambda e \\ \rho \end{pmatrix} \tag{7.26}$$

for the predictor direction Δx_p, Δs_p and $\Delta\lambda_p$, using the dual ordering.

The predictor-corrector step, for the i correction ($i = 1, \ldots, m_{pc}$) is then obtained from the perturbed Newton system

$$\begin{pmatrix} H(x,\lambda) & 0 & -J(x) \\ 0 & -\Lambda & -S \\ -J^T(x) & -I & 0 \end{pmatrix} \begin{pmatrix} \Delta x^i_{pc} \\ \Delta s^i_{pc} \\ \Delta\lambda^i_{pc} \end{pmatrix} = \begin{pmatrix} \sigma \\ S\Lambda e + \mu e + \Delta S^{i-1}\Delta\Lambda^{i-1}e \\ \rho \end{pmatrix} \tag{7.27}$$

where ΔS^{i-1}, $\Delta\Lambda^{i-1}$ are diagonal matrices with elements Δs^{i-1}_{pc} and $\Delta\lambda^{i-1}_{pc}$ respectively and ΔS^0, $\Delta\Lambda^0$ are diagonal matrices with elements Δs_p, $\Delta\lambda_p$ respectively.

The algorithm is described in the following way.

Algorithm 19 *Predictor-Corrector*

Step(a) *Solve system* (7.26) *to obtain the predictor direction* Δx_p, Δs_p *and* $\Delta\lambda_p$ *using the following formulae*

$$\left(H - JS^{-1}\Lambda J^T\right)\Delta x_p = -\nabla f + JS^{-1}\Lambda\rho \tag{7.28a}$$

$$\Delta s_p = -\left(\rho + J^T\Delta x_p\right) \tag{7.28b}$$

$$\Delta\lambda_p = -S^{-1}\left(\Lambda\Delta s_p + S\Lambda e\right) \tag{7.28c}$$

Step(b) *For* $i = 1, \ldots, m_{pc}$ *do*

Step(b.1) *Compute* $\mu = \frac{\left(\lambda+\alpha_{max}\Delta\lambda_{pc}^{i-1}\right)^T\left(s+\tau e+\alpha_{max}\Delta s_{pc}^{i-1}\right)}{m}$, *with* $\Delta x_{pc}^0 = \Delta x_p$, $\Delta s_{pc}^0 = \Delta s_p$, $\Delta\lambda_{pc}^0 = \Delta\lambda_p$ *and* α_{max} *is the maximum step size allowed given by* (7.20).

Step(b.2) *Solve system* (7.27) *to obtain the predictor-corrector direction* Δx_{pc}^i, Δs_{pc}^i *and* $\Delta\lambda_{pc}^i$ *using the following formulae*

$$\left(H - JS^{-1}\Lambda J^T\right)\Delta x_{pc}^i = -\nabla f + JS^{-1}\left(\Lambda\rho - \mu e - \Delta S_{pc}^{i-1}\Delta\Lambda_{pc}^{i-1}e\right) \tag{7.29a}$$

$$\Delta s_{pc}^i = -\left(\rho + J^T\Delta x_{pc}^i\right) \tag{7.29b}$$

$$\Delta\lambda_{pc}^i = -S^{-1}\left(\Lambda\Delta s_{pc}^i + S\Lambda e + \mu e + \Delta S_{pc}^{i-1}\Delta\Lambda_{pc}^{i-1}e\right) \tag{7.29c}$$

Step(c) *continue with* $\Delta x = \Delta x_{pc}^{m_{pc}}$, $\Delta s = \Delta s_{pc}^{m_{pc}}$ *and* $\Delta\lambda = \Delta\lambda_{pc}^{m_{pc}}$.

Proposition 7.2.1 *If* x *is a feasible solution to problem* (7.1) *then the predictor-corrector direction obtained by solving* (7.27) *may not be a descent direction to the* l_2 *merit function* (7.15).

Proof. [46, Proposition 4.1] ■

7.2.7 Computing the integrals

The integrals are computed in the same way as for the Penalty and SQP methods. See Section 5.5 for a description and Section 5.6 for the options.

7.2.8 Stopping rule

We measure proximity to the solution by means of the primal and dual infeasibility. The algorithm will stop with a primal-dual solution if

$$\|\rho\|_2 \leq \delta_f \tag{7.30}$$

and

$$\|\sigma\|_2 \leq \delta_f. \tag{7.31}$$

7.2.9 A watchdog technique

To overcome a possibly Maratos effect a second order correction or a watchdog technique could be used. In this particular case of constraints transcription the Jacobian can be a null matrix, and a second-order correction is unappropriate. A watchdog technique was implemented.

The watchdog technique consists of allowing an increase on the merit function for a given number of iterations, before forcing a decrease.

The watchdog technique can be describe by the following algorithm.

Algorithm 20 *(watchdog technique)*

Step (a) *If* $k = 0$ *then let* `watch`$=0$.

Step (b) *Compute* α^k_{max} *by formula* (7.20).

Step (c) *If* `watch`$<$`watchmax`

then *Let* $x^{k+1} = x^k + \alpha^k_{max}\Delta x^k$, $s^{k+1} = s^k + \alpha^k_{max}\Delta s^k$ *and* $\lambda^{k+1} = \lambda^k + \alpha^k_{max}\Delta\lambda^k$.
If $\phi(x^{k+1}, s^{k+1}; \mu^k, \beta^k) - \phi(x^k, s^k; \mu^k, \beta^k) < 0$
then *Save* x^{k+1}, s^{k+1}, λ^{k+1}, Δx^k, Δs^k, $\Delta\lambda^k$, α^k_{max}, β^k, B^k *and* $\phi(x^{k+1}, s^{k+1}; \mu^k, \beta^k)$.
Set `watch`$=0$ *and go to Step* (f).
else *If* `watch=0`
then *Save* x^k, s^k, λ^k, Δx^k, Δs^k, $\Delta\lambda^k$, α^k_{max}, β^k, B^k *and* $\phi(x^k, s^k; \mu^k, \beta^k)$.
Let `watch`$=$`watch`$+1$. *Go to Step* (f)
else *Restore* x^k, s^k, λ^k, Δx^k, Δs^k, $\Delta\lambda^k$, α^k_{max}, β^k, $\phi(x^k, s^k; \mu^k, \beta^k)$ *and* B^k. *Let* $\alpha^k = \frac{\alpha^k_{max}}{2}$ *and go to Step* (d)

Step (d) *Compute* β^{k+1} *using Algorithm 18.*

Step (e) *Compute* α^k *such that* $\phi(x^{k+1}, s^{k+1}; \mu^k, \beta^{k+1}) - \phi(x^k, s^k; \mu^k, \beta^{k+1}) < 0$, *with* $x^{k+1} = x^k + \alpha^k\Delta x^k$, $s^{k+1} = s^k + \alpha^k\Delta s^k$ *and* $\lambda^{k+1} = \lambda^k + \alpha^k\Delta\lambda^k$. *Save* x^{k+1}, s^{k+1}, λ^{k+1}, Δx^k, Δs^k, $\Delta\lambda^k$, α^k_{max}, β^{k+1}, B^k *and* $\phi(x^{k+1}, s^{k+1}; \mu^k, \beta^{k+1})$.

Step (f) *Continue.*

7.2.10 Jamming and stability

The S^{-1} matrix can produce some numerical instability in the solution of the linear system 7.25 (see [48]) when driven to zero too fast. A strategy described in [4] was implemented, i.e.,

$$s_i^k = \max\left\{s_i^k, 10^{-8} - \tau\right\}. \tag{7.32}$$

This strategy has proven to give better results than the harmonic average suggested in [4].

7.3 The full algorithm

We present in this section the full primal-dual interior point algorithm.

Algorithm 21 *The full implemented algorithm.*

Step (a) *Given* x^0, ϵ, τ, θ, δ_μ *and* δ_f.

Step (b) *Compute* s_i^0, $i = 1, \ldots, m$ *using* (7.21). *Compute* λ_i^0, $i = 1, \ldots, m$ *using* (7.22). *Let* $k = 0$.

Step (c) *Let* $y_{eps} = x^k$ *be the last* y *computed for a given* ϵ.

Step (d) *Compute or update* μ^k *using* (7.23).

Step (e) *Check the stopping criteria described in Subsection 7.2.8. If the stopping criteria is satisfied then if there is a sufficient difference between* y_{eps} *and* x^k *then decrease* ϵ, τ, *update the slack variables and continue, otherwise stop.*

Step (f) *Update* B^k *by a BFGS formula. If* $k = 0$ *then* B^k =*Identity matrix.*

Step (g) *Solve the KKT system to obtain the search direction* $(\Delta x^k, \Delta s^k, \Delta\lambda^k)$, *using* (7.25).

Step (h) *Compute* β^k *as described in Algorithm 18*

Step (i) *Compute* α_{max}^k *as described in* (7.20).

Step (j) *Use a backtracking strategy to find* α^k *that reduces the merit function* (7.15) *or* (7.16).

Step (k) *Compute* x^{k+1}, s^{k+1} *and* λ^{k+1} *as in* (7.14).

Step (l) *Go to Step* (d).

7.4 Interior Point method options

In the computation of the integrals the options of Section 5.6 are still valid for tuning the integral computation. The options to the Interior Point method are presented in Table 7.1.

The columns in Table 7.1 have the same meaning as in Table 4.1 and "String" in the column "Type" means that the option should be a sequence of characters starting with a letter.

7.5 The Interior Point method output

The Interior Point method will append the line

& iter & nlag & nmer & fx & $||\rho||_2$ & $||\sigma||_2$ & μ & ϵ \\

to the output file. "iter" is the number of iterations; "nlag" is the number of the Lagrangian function evaluations; "nmer" is the number of merit function evaluations; "fx" is the objective function value in the solution found; "$||\rho||_2$" and "$||\sigma||_2$" are the primal and dual infeasibility and "μ"/"ϵ" is the last μ/ϵ used.

The same comment of Section 4.4 about building a LaTeX tabular environment applies.

Option	Type	Default	Description
int_merit	String	`m1`	`m1` for the l_2 merit function (7.15). `m2` for the augmented Lagrangian merit function (7.16).
damped	Integer	1	0 if no damped BFGS updating formula is used. For any other integer value the damped formula is used. See Subsection 7.2.5 and [25].
int_maxit	Integer	1000	Maximum number of iterations allowed.
int_prec	Double	10^{-3}	Primal and dual infeasibility measures in the stopping criteria δ_f.
int_eps	Double	10^{-4}	Initial smoothing parameter for differentiability ϵ_0. $\tau_0 = \epsilon_0$.
int_epsfactor	Double	0.1	Reduction factor for the smoothing parameter. ξ_τ.
int_epslimit	Double	10^{-8}	Minimum bound allowed for the smoothing parameter.
reset	Integer	0	0 if no reset of the approximation of the Hessian if requested. Any integer value will make B^k =identity matrix if k is a multiple of n.
scale	Integer	0	0 if no scaling is needed. Any value otherwise. If the norm of the direction is not in range $[10^{-2}, 10^2]$ the direction will be scaled to fit the values.
theta	Double	1.0	Value used to avoid proximity of the s variable to the bound, θ in (7.21).
delta	Double	0.1	Scaling factor in computing the barrier parameter, δ_μ in (7.23).
watchmax	Integer	3	Maximum number of consecutive increases in the merit function.
watchdog	Integer	0	0 for no watchdog strategy and any other value otherwise.
bfgsskip	Double	10^{-8}	The BFGS update will be skipped if curvature condition is less than the given value, δ_c.
int_ord	String	`d`	`p` for Primal ordering, `d` for Dual ordering and `pc` for Predictor-Corrector.
int_corr	Integer	1	number of corrections done in the predictor-corrector direction, m_{pc}.

Table 7.1: Options for the quasi-Newton interior point method

Chapter 8

SIPAMPL

The SIPAMPL was made in the end of 1999. With SIPAMPL we aimed to create a database with SIP problems, to use the automatic differentiation procedures in AMPL and to provide an interface to connect any SIP solver. The database provides several already coded SIP problems and the natural modeling language of AMPL allows an easy extension of the database. We use Section 8.1 for a brief description of the database and describe how a SIP problem can be coded in the AMPL modeling language. To make the database useful we have made an interface to connect the database to any solver. Section 8.2 describes the SIPAMPL interface. The technical report [36] is used as the SIPAMPL manual and can be obtained together with the SIPAMPL software, from the web page indicated in the reference [36].

8.1 SIPAMPL database

Since 1999 we have been coding all the problems that were found in the literature related to SIP. As a good practice we have coded, as a comment, in the beginning of each file (problem model) the reference from where we have obtained the problem. We have used, in most cases, the name of the first author to identify the problem. The existence of this database will also allow the forthcoming authors to mention only the problem name under the SIPAMPL database, avoiding the need of writing its mathematical formulation which sometimes leads to mistakes.

To codify a SIP problem in the AMPL modeling language is an easy task and no derivatives are requested. AMPL automatic differentiation is used to provide first and second derivatives.

The database has now more than 130 coded problems and we expect to extend the database whenever possible. We welcome references to SIP problems or SIP problems already coded.

As an example of a coded problem consider the following SIP problem

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

The corresponding (SIP) AMPL code is presented below

```
#####################################################################
# Objective: Quadratic
# Constraints: Linear
#####################################################################
# Sample problem in the user manual
# aivaz@ci.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
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 (see Section 9.4 for more details on how to change it).

The codification requests for coding SIP problems in AMPL are:

- Infinite variables are coded with names starting with t and conversely any variable names starting with t are considered infinite;
- Infinite constraints are coded with names starting with t and conversely any constraint names starting with t are considered infinite;
- With the option *auxfiles* the `.row` and `.col` AMPL files must be provided.

The robotics problems codified use the `bspline.dll` dynamic library. The dynamic library must be loaded before using these problems (see [36] on how to load it).

8.2 SIPAMPL interface

The SIPAMPL interface extends the AMPL interface, allowing the user to evaluate the finite/infinite constraints and access to other data related with the SIP problems (number of finite variables, number of infinite variables, etc). The SIPAMPL interface calls the AMPL interface routines and with the extra information provided by the `.row` and `.col` files it fills the data in Table 8.1 with the appropriate values. In table the columns refer to: "Variable", the variable name and type; "Description", a brief description for the variable and "SIP", the notation used in the SIP definition (2.1).

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 finite component from the initial complete variable array.

Variable	Description	SIP
int nxsip	number of finite variables	n
int ntsip	number of infinite variables	p
int nxsipc	number of finite constraints	q
int ntsipc	number of infinite constraints	m
real *XBU	Array of upper bounds on finite variables	
real *XBL	Array of lower bounds on finite variables	
real *TBU	Array of upper bounds on infinite variables	α_j
real *TBL	Array of lower bounds on infinite variables	β_j
real *XCBU	Array of upper bounds on finite constraints	
real *XCBL	Array of lower bounds on finite constraints	
real *TCBU	Array of upper bounds on infinite constraints	
real *TCBL	Array of lower bounds on infinite constraints	

Table 8.1: Data provided by the SIPAMPL interface

- `sip_extractt` - As in `sip_extractx` but for the infinite component.
- `sip_joinxt` - Joins the finite and infinite components back into the complete variable array.
- `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 finite and infinite positions in the complete variable array (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).
- `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.
- `sip_objgrd` - Evaluates the objective function gradient vector.
- `sip_objhes` - Evaluates the objective function Hessian matrix.
- `sip_conval` - Evaluates the constraints.
- `sip_jacval` - Evaluates the constraint Jacobian.
- `sip_conxval` - Evaluates a finite constraint.
- `sip_contval` - Evaluates an infinite constraint.
- `sip_conxgrd` - Evaluates a finite constraint gradient.

- `sip_contgrd` - Evaluates an infinite constraint gradient.
- `sip_conxhes` - Evaluates a finite constraint Hessian matrix.
- `sip_conthes` - Evaluates an infinite constraint Hessian matrix.

The AMPL interface routines may also be called.

Chapter 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 9.1. 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 (2.1). The next section presents the installation instructions. Section 9.2 is devoted to the implementation details. Section 9.3 gives a session example of the *select* tool.

9.1 Installing the *select* tool

At this moment the *select* tool is available for a Linux and a Windows Operating System 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). For the Windows version we used the Microsoft Visual C/C++ compiler.

The *select* tool is provided in only two files: `select.h` and `select.c`. `makefile` and `makefile.vc` are also provided and to produce the `select` binary one just have to type `make` or `nmake -f makefile.vc` in the *select* directory.

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

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	$[0,+\infty]$	n
Number infinite variables	range	Nonnegative integers	$[0,+\infty]$	p
Number finite constraints	range	Nonnegative integers	$[0,+\infty]$	q
Number infinite constraints	range	Nonnegative integers	$[0,+\infty]$	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	$[\alpha, \beta]$
Initial guess	list	With initial guess Without initial guess With or without initial guess	With or without initial guess	

Table 9.1: Questionable problems characteristics

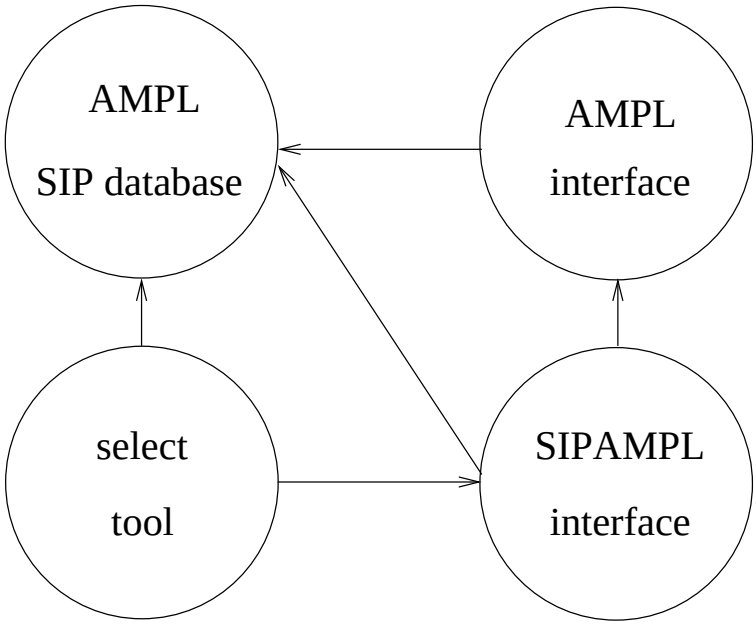

Figure 9.1: Interconnections between *select* and SIPAMPL

```
# Objective: Quadratic
# Constraints: Linear
```

in the example shown (Section 8.1) 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 files `.nl`, `.col` and `.row`. To produce them *select* copies the `.mod` file to a temporary file and replaces the `solve;` AMPL command with

```
option auxfiles rc;
write gtmpfile;
```

The great amount of time spent for processing a file in the database is due to the great 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 appearing 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 9.1.

9.3 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.

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 with the `./select` command. The default database directory and AMPL binary are accepted as being correct. A quadratic objective function with only one infinite variable will be requested. No finite simple bounds on the finite variables are requested. After the *select* tool having found the problems in the database, that match the user options the file `select.res` is written which contains all the found problems.

```
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>
```

9.4 Changing the solver name in the database problems

In each problem file the `nsips` 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 so.
- 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

This 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.

Bibliography

[1] P. Armand, J.C. Gilbert, and S. Jan-Jégou, *A BFGS-IP algorithm for solving strongly convex optimization problems with feasibility enforced by an exact penalty approach*, Tech. Report 4087, INRIA - Institut National de Recherche en Informatique et en Automatique, December 2000, Downloadable from Optimization Online.

[2] ———, *A feasible BFGS interior point algorithm for solving strongly convex minimization problems*, SIAM Journal on Optimization **11** (2000), 199–222.

[3] H.Y. Benson, D.F. Shanno, and R.J. Vanderbei, *Interior-point methods for nonconvex nonlinear programming: Filter methods and merit functions*, Tech. Report ORFE-00-06, Princeton University, 2000.

[4] ———, *Interior-point methods for nonconvex nonlinear programming: Jamming and comparative numerical testing*, Tech. Report ORFE-00-02, Princeton University, 2000.

[5] T.J. Carpenter, I.J. Lustig, J.M. Mulvey, and D.F. Shanno, *Higher-order predictor-corrector interior point methods with application to quadratic objectives*, SIAM Journal on Optimization **3** (1993), no. 4, 696–725.

[6] A.R. Conn and N.I.M. Gould, *An exact penalty function for semi-infinite programming*, Mathematical Programming **37** (1987), 19–40.

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

[8] A.S. El-Bakry, R.A. Tapia, T. Tsuchiya, and Y. Zhang, *On the formulation and theory of the Newton interior-point method for nonlinear programming*, Journal of Optimization Theory and Applications **89** (1996), no. 3, 507–541.

[9] R. Fletcher and S. Leyffer, *Nonlinear programming without a penalty function*, Numerical analysis report, NA/171, University of Dundee, September 1997.

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

[11] P.E. Gill, W. Murray, M.A. Saunders, and M.H. Wright, *User's guide for NPSOL: A fortran package for nonlinear programming*, Stanford University, 1986.

[12] J. Gondzio, *Multiple centrality corrections in a primal-dual method for linear programming*, Tech. Report 1994.20, University of Geneva, Switzerland, 1995.

[13] P.R. Gribik, *A central-cutting-plane algorithm for semi-infinite programming problems*, in [7] (1979), 66–82.

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

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

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

[17] R. Hettich and K.O. Kortanek, *Semi-infinite programming: Theory, methods, and applications*, SIAM Review **35** (1993), no. 3, 380–429.

[18] F. Jarre and M. Wechs, *Extending Mehrotra's corrector for linear programs*, Tech. report, Universität Würzburg, Federal Republic of Germany, 1999.

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

[20] J. Kaliski, D. Haglin, C. Roos, and T. Terlaky, *Logarithmic barrier decomposition methods for semi-infinite programming*, International Transactions in Operational Research **4** (1997), no. 4, 285–303.

[21] T. León, S. Sanmatías, and E. Vercher, *On the numerical treatment of linearly constrained semi-infinite optimization problems*, European Journal of Operational Research **121** (2000), 78–91.

[22] The NAG Library, *Numerical algorithms group*, Mayfield House, 256 Banbury Rd., Oxford OX2 7DE, U.K.

[23] 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.

[24] S. Mehrotra, *On the implementation of a primal-dual interior point method*, SIAM Journal on Optimization **2** (1992), no. 4, 575–601.

[25] J. Nocedal and S.J. Wright, *Numerical Optimization*, Springer Series in Operations Research, Springer, 1999.

[26] E. Polak, *On the mathematical foundations of nondifferentiable optimization in engineering design*, SIAM Review **29** (1987), no. 1, 21–89.

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

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

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

[30] K. Roleff, *A stable multiple exchange algorithm for linear SIP*, in [7] (1979), 83–96.

[31] D.F. Shanno and R.J. Vanderbei, *Interior-point methods for nonconvex nonlinear programming: Orderings and higher-order methods*, Mathematical Programming **87** (2000), 303–316.

[32] 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.

[33] 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.

[34] R.J. Vanderbei, *LOQO: An interior point code for quadratic programming*, Tech. Report SOR-94-15, Statistics and Operation Research, Princeton University, 1998.

[35] R.J. Vanderbei and D.F. Shanno, *An interior-point algorithm for nonconvex nonlinear programming*, Computational Optimization and Applications **13** (1999), 231–252.

[36] A.I.F. Vaz, E.M.G.P. Fernandes, and M.P.S.F. Gomes, *SIPAMPL: Semi-Infinite Programming with AMPL*, Technical Report ALG/EF/1-2000, Universidade do Minho, Braga, Portugal, April 2000, http://www.eng.uminho.pt/∼dps/aivaz/, sent for publication.

[37] ———, *Discretization methods for semi-infinite programming*, Investigação Operacional **21** (2001), no. 1, 37–46.

[38] ———, *NSIPS: Nonlinear Semi-Infinite Programming Solver*, Technical Report ALG/EF/1-2001 (2001), http://www.eng.uminho.pt/∼dps/aivaz/.

[39] ———, *Penalty function algorithms for semi-infinite programming*, EURO 2001, July 2001, p. 91.

[40] ———, *Penalty function algorithms for semi-infinite programming*, Internal report (to be published) (2001).

[41] ———, *A penalty technique for semi-infinite programming*, Proceedings of V-SGAPEIO (Ferrol, Spain), 2001, pp. 235–238.

[42] ———, *Robot trajectory planning with semi-infinite programming*, Proceedings of the ORP3 conference (7 pages) (2001), Paris.

[43] ———, *A sequential quadratic method with a dual parametrization approach to semi-infinite programming*, to be published (2001).

[44] ———, *A sequential quadratic programming for nonlinear semi-infinite programming*, Optimization 2001, July 2001, p. 27.

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

[46] ———, *A quasi-newton interior point method for semi-infinite programming*, Technical Report, Submitted (2002).

[47] ———, *SIPAMPL v2.0: Semi-Infinite Programming with AMPL*, Technical Report ALG/EF/4-2002, Universidade do Minho, Braga, Portugal, December 2002, http://www.eng.uminho.pt/∼dps/aivaz/, sent for publication.

[48] Andreas Wachter and Lorenz T. Biegler, *Failure of global convergence for a class of interior point methods for nonlinear programming*, Mathematical Programming **88** (2000), 565–574.

[49] M.-H. Wang and Y.-E. Kuo, *A perturbation method for solving linear semi-infinite programming problems*, Computers and Mathematic with Applications **37** (1999), 181–198.

[50] G.A. Watson, *Globally convergent methods for semi-infinite programming*, BIT **21** (1981), 362–373.

[51] R.L. Wheeden and A. Zygmund, *Measure and Integral, an Introduction to Real Analysis*, Marcel Dekker, Inc., 1977.

[52] M.A. Wolfe, *Numerical Methods for Unconstrained Optimization, an Introduction*, Van Nostrand Reinhold Company, 1978.

[53] S.-Y. Wu and S.-C. Fang, *Solving convex programs with infinitely many constraints by a relaxed cutting plane method*, Computers and Mathematics with Applications **38** (1999), 23–33.