OPTIMIZATION IN SCILAB

CONSORTIUM SCILAB
Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex France
digiteo
Recherche en sciences & technologies de l'information

This document has been written by Michaël Baudin and Vincent Couvert from the Scilab Consortium and by Serge Steer from INRIA Paris - Rocquencourt.

Abstract

In this document, we make an overview of optimization features in Scilab. The goal of this document is to present all existing and non-existing features, such that a user who wants to solve a particular optimization problem can know what to look for. In the introduction, we analyse a classification of optimization problems. In the first chapter, we analyse the flagship of Scilab in terms of nonlinear optimization: the `optim` function. We analyse its features, the management of the cost function, the linear algebra and the management of the memory. Then we consider the algorithms which are used behind `optim`, depending on the type of algorithm and the constraints. In the remaining chapters, we present the algorithms available to solve quadratic problems, nonlinear least squares problems, semidefinite programming, genetic algorithms, simulated annealing and linear matrix inequalities. A chapter focus on optimization data files managed by Scilab, especially MPS and SIF files. Some optimization features are available in the form of toolboxes, the most important of which are the Quapro and CUTEr toolboxes. The final chapter is devoted to missing optimization features in Scilab.

Contents

Introduction

This document aims at giving Scilab users a complete overview of optimization features in Scilab. It is written for Scilab partners needs in OMD project (http://omd.lri.fr/tiki-index.php). The core of this document is an analysis of current Scilab optimization features. In the final part, we give a short list of new features which would be interesting to find in Scilab. Above all the embedded functionalities of Scilab itself, some contributions (toolboxes) have been written to improve Scilab capabilities. Many of these toolboxes are interfaces to optimization libraries, such as FSQP for example.

In this document, we consider optimization problems in which we try to minimize a cost function $f(x)$ with or without constraints. These problems are partly illustrated in figure 1. Several properties of the problem to solve may be taken into account by the numerical algorithms :

- The unknown may be a vector of real values or integer values.
- The number of unknowns may be small (from 1 to 10 - 100), medium (from 10 to 100 - 1 000) or large (from 1 000 - 10 000 and above), leading to dense or sparse linear systems.
- There may be one or several cost functions (multi-objective optimization).
- The cost function may be smooth or non-smooth.
- There may be constraints or no constraints.
- The constraints may be bounds constraints, linear or non-linear constraints.
- The cost function can be linear, quadratic or a general non linear function.

An overview of Scilab optimization tools is showed in figure 2.
In this document, we present the following optimization features of Scilab.

- nonlinear optimization with the `optim` function,
- quadratic optimization with the `qpsolve` function,
- nonlinear least-square optimization with the `lsqrsolve` function,
- semidefinite programming with the `semidef` function,
- genetic algorithms with the `optim_ga` function,
- simulated annealing with the `optim_sa` function,
- linear matrix inequalities with the `lmisolver` function,

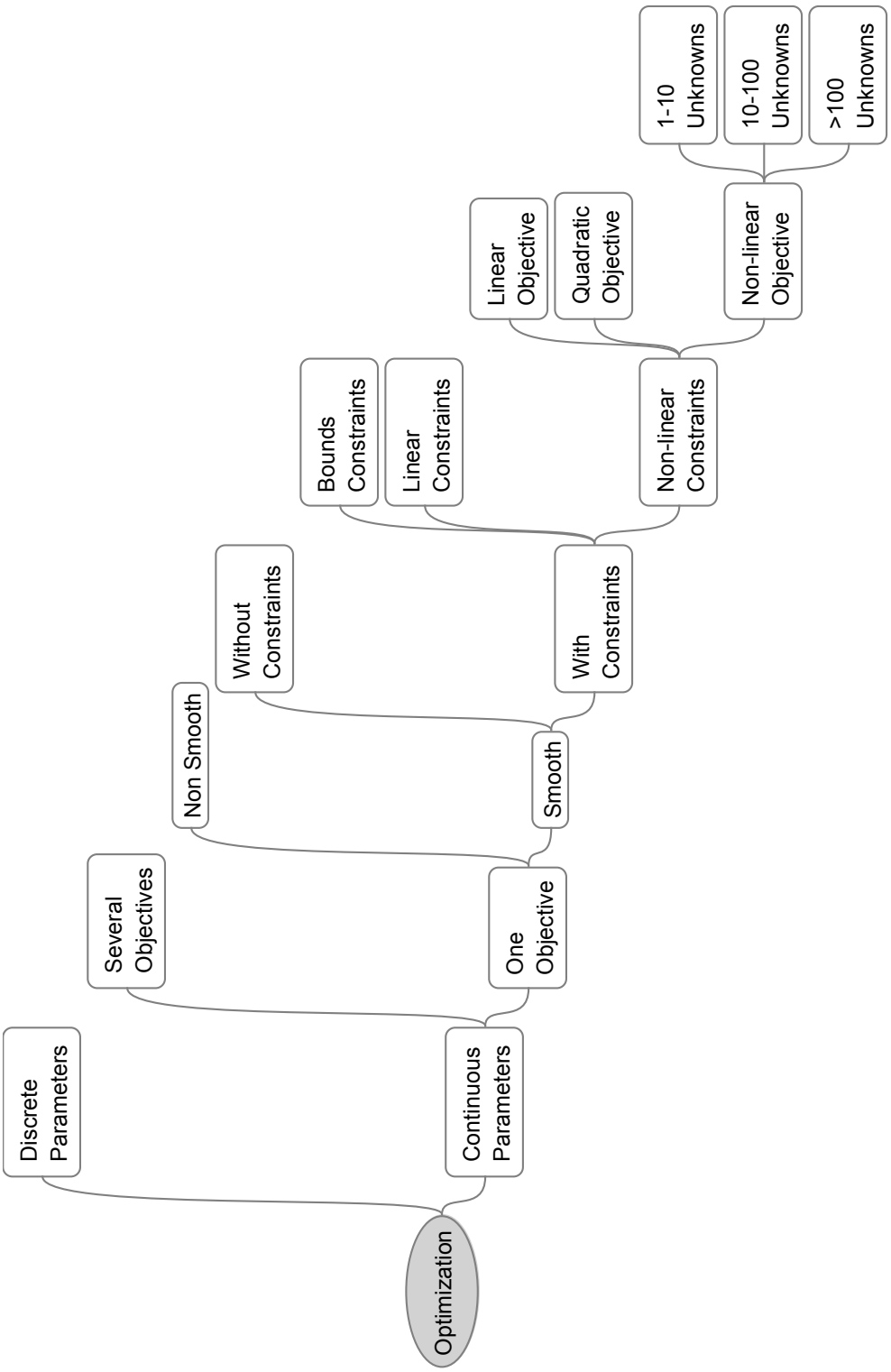

Figure 1: Classes of optimization problems

	Optimization Problem	**Scilab Feature**	**Scilab Toolbox**
Integer variable	integer variable, linear objective		Interface to LPSOLVE
Non linear objective	real variable, non linear objective, without constraints	optim	
	real variable, non linear objective, with bounds constraints	optim, GA, SA	
	real variable, non linear objective, with nonlinear constraints		Interface to CONMIN, Interface to FSQP, Interface to IPOPT
	real variable, non differentiable objective, without constraints		Interface to NEWUOA
Linear objective	Real variable, linear objective, without constraints, with/wo bound constraints, with/wo equality constraints		quapro
	real variable, linear objective, bounds constraints		Interface to LIPSOL
Quadratic objective	Real variable, quadratic objective, with/wo bound constraints, with/wo equality constraints	qpsolve, qp_solve	
	Real variable, quadratic objective, without constraints, with/wo bound constraints, with/wo equality constraints		quapro
Multi-objective optimization	real variable, non linear multi-objective, with bounds constraints	GA	
Semi-definite programming	real variable, linear objective, linear positivity matrix constraint	semidef	
	real variable, linear objective, linear matrix equality and matrix inequality	lmisolver	

Figure 2: Scilab Optimization Tools

- reading of MPS and SIF files with the `quapro` and `CUTEr` toolboxes.

Scilab v5.2 provides the `fminsearch` function, which a derivative-free algorithm for small problems. The `fminsearch` function is based on the simplex algorithm of Nelder and Mead (not to be confused with Dantzig's simplex for linear optimization). This unconstrained algorithm does not require the gradient of the cost function. It is efficient for small problems, i.e. up to 10 parameters and its memory requirement is only $O(n)$. This algorithm is known to be able to manage "noisy" functions, i.e. situations where the cost function is the sum of a general nonlinear function and a low magnitude function. The `neldermead` component provides three simplex-based algorithms which allow to solve unconstrained and nonlinearly constrained optimization problems. It provides an object oriented access to the options. The `fminsearch` function is, in fact, a specialized use of the `neldermead` component. This component is presented in in depth in [2].

An analysis of optimization in Scilab, including performance tests, is presented in "Optimization with Scilab, present and future" [3]. The following is the abstract of the paper :

"We present in this paper an overview of optimization algorithms available in theScilab software. We focus on the user's point of view, that is, we have to minimize or maximize an objective function and must find a solver suitable for the problem. The aim of this paper is to give a simple but accurate view of what problems can be solved by Scilab and what behavior can be expected for those solvers. For each solver, we analyze the type of problems that it can solve as well as its advantages and limitations. In order to compare the respective performances of the algorithms, we use the CUTEr library, which is available in Scilab. Numerical experiments are presented, which indicates that there is no cure-for-all solvers."

Several existing optimization features are not presented in this document. We especially mention the following tools.

- The `fsqp` toolbox provides an interface for a Sequential Quadratic Programming algorithm. This algorithm is very efficient but is not free (but is provided by the authors, free of charge, for an academic use).

- Multi-objective optimization is available in Scilab with the genetic algorithm component.

The organization of this document is as following.

In the first chapter, we analyse the flagship of Scilab in terms of nonlinear optimization: the `optim` function. This function allows to solve nonlinear optimization problems without constraints or with bound constraints. It provides a Quasi-Newton method, a Limited Memory BFGS algorithm and a bundle method for non-smooth functions. We analyse its features, the management of the cost function, the linear algebra and the management of the memory. Then we consider the algorithms which are used behind `optim`, depending on the type of algorithm and the constraints.

In the second chapter we present the `qpsolve` and `qp_solve` functions which allows to solve quadratic problems. We describe the solvers which are used, the memory requirements and the internal design of the tool.

The chapter 3 and 4 briefly present non-linear least squares problems and semidefinite programming.

The chapter 5 focuses on genetic algorithms. We give a tutorial example of the `optim_ga` function in the case of the Rastrigin function. We also analyse the support functions which allow to configure the behavior of the algorithm and describe the algorithm which is used.

The simulated annealing is presented in chapter 6, which gives an overview of the algorithm used in `optim_sa`. We present an example of use of this method and shows the convergence of the algorithm. Then we analyse the support functions and present the neighbor functions, the acceptance functions and the temperature laws. In the final section, we analyse the structure of the algorithm used in `optim_sa`.

The LMITOOL module is presented in chapter 7. This tool allows to solve linear matrix inequalities. This chapter was written by Nikoukhah, Delebecque and Ghaoui. The syntax of the `lmisolver` function is analysed and several examples are analysed in depth.

The chapter 8 focuses on optimization data files managed by Scilab, especially MPS and SIF files.

Some optimization features are available in the form of toolboxes, the most important of which are the Quapro, CUTEr and the Unconstrained Optimization Problems toolboxes. These modules are presented in the chapter 9, along with other modules including the interface to CONMIN, to FSQP, to LIPSOL, to LPSOLVE, to NEWUOA.

The chapter 10 is devoted to missing optimization features in Scilab.

Chapter 1

Non-linear optimization

The goal of this chapter is to present the current features of the `optim` primitive Scilab. The `optim` primitive allows to optimize a problem with a nonlinear objective without constraints or with bound constraints.

In this chapter, we describe both the internal design of the `optim` primitive. We analyse in detail the management of the cost function. The cost function and its gradient can be computed using a Scilab function, a C function or a Fortran 77 function. The linear algebra components are analysed, since they are used at many places in the algorithms. Since the management of memory is a crucial feature of optimization solvers, the current behaviour of Scilab with respect to memory is detailed here.

Three non-linear solvers are connected to the `optim` primitive, namely, a BFGS Quasi-Newton solver, a L-BFGS solver, and a Non-Differentiable solver. In this chapter we analyse each solver and present the following features :

- the reference articles or reports,
- the author,
- the management of memory,
- the linear algebra system, especially the algorithm name and if dense/sparse cases are taken into account,
- the line search method.

The Scilab online help is a good entry point for this function.

1.1 Mathematical point of view

The problem of the non linear optimization is to find the solution of

$$\min_x f(x)$$

with bounds constraints or without constraints and with $f : \mathbb{R}^n \to \mathbb{R}$ the cost function.

1.2 Optimization features

Scilab offers three non-linear optimization methods:

- Quasi-Newton method with BFGS formula without constraints or with bound constraints,
- Quasi-Newton with limited memory BFGS (L-BGFS) without constraints or with bound constraints,
- Bundle method for non smooth functions (half derivable functions, non-differentiable problem) without constraints.

Problems involving non linear constraints cannot be solved with the current optimization methods implemented in Scilab. Non smooth problems with bounds constraints cannot be solved with the methods currently implemented in Scilab.

1.3 Optimization routines

Non-linear optimization in Scilab is based on a subset of the MODULOPT library, developed at INRIA. The library which is used by optim was created by the Modulopt project at INRIA and developped by Bonnans, Gilbert and Lemaréchal [5].

This section lists the routines used according to the optimization method used.

The following is the list of solvers currently available in Scilab, and the corresponding fortran routine :

- "qn" without constraints : a Quasi-Newton method without constraints, n1qn1,
- "qn" with bounds constraints : Quasi-Newton method with bounds constraints, qnbd,
- "gc" without constraints : a Limited memory BGFS methoud without constraints, n1qn3,
- "gc" with bounds constraints : a Limited memory BGFS with bounds constraints, gcbd,
- "nd" without constraints : a Non smooth method without constraints, n1fc1.

1.4 The cost function

The communication protocol used by optim is direct, that is, the cost function must be passed as a callback argument to the "optim" primitive. The cost function must compute the objective and/or the gradient of the objective, depending on the input integer flag "ind".

In the most simple use-case, the cost function is a Scilab function, with the following header :

```
[f,g,ind]=costf(x,ind)
```

where "x" is the current value of the unknown and "ind" is the integer flag which states if "f", "g" or both are to be computed.

The cost function is passed to the optimization solver as a callback, which is managed with the fortran 77 callback system. In that case, the name of the routine to call back is declared as "external" in the source code. The cost function may be provided in the following ways :

- the cost function is provided as a Scilab script,
- the cost function is provided as a C or fortran 77 compiled routine.

If the cost function is a C or fortran 77 source code, the cost function can be statically or dynamically linked against Scilab. Indeed, Scilab dynamic link features, such as ilib_for_link for example, can be used to provide a compiled cost function.

In the following paragraph, we analyse the very internal aspects of the management of the cost function.

This switch is managed at the gateway level, in the sci_f_optim routine, with a "if" statement :

- if the cost function is compiled, the "foptim" symbol is passed,
- if not, the "boptim" symbol is passed.

In the case where the cost function is a Scilab script, the "boptim" routine performs the copy of the input local arguments into Scilab internal memory, calls the script, and finally copy back the output argument from Scilab internal memory into local output variables. In the case where the cost function is compiled, the computation is based on function pointers, which are managed at the C level in optimtable.c. The optimization function is configured by the "setfoptim" C service, which takes as argument the name of the routine to callback. The services implemented in AddFunctionInTable.c are used, especially the function "AddFunctionInTable", which takes the name of the function as input argument and searches the corresponding function address, be it in statically compiled libraries or in dynamically compiled libraries. This allows the optimization solvers to callback dynamically linked libraries. These names and addresses are stored in the hashmap FTab_foptim, which maps function names to function pointer. The static field foptimfonc with type foptimf is then set to the address of the function to be called back. When the optimization solver needs to compute the cost function, it calls the "foptim" C void function which in returns calls the compiled cost function associated to the configured address (*foptimfonc).

1.5 Linear algebra

The linear algebra which is used in the "optim" primitive is dense. Generally, the linear algebra is inlined and there is no use the BLAS API. This applies to all optimization methods, except "gcbd". This limits the performance of the optimization, because optimized libraries like ATLAS cannot not used. There is only one exception : the L-BFGS with bounds constraints routine gcbd uses the "dcopy" routine of the BLAS API.

1.6 Management of memory

The optimization solvers requires memory, especially to store the value of the cost function, the gradient, the descent direction, but most importantly, the approximated *Hessian of the cost function.*

Most of the memory is required by the approximation of the Hessian matrix. If the full approximated Hessian is stored, as in the BFGS quasi-Newton method, the amount of memory is the square of the dimension of the problem $O(n^2)$, where n is the size of the unknown. When

a quasi-Newton method with limited memory is used, only a given number m of vectors of size n are stored.

This memory is allocated by Scilab, inside the stack and the storage area is passed to the solvers as an input argument. This large storage memory is then split into pieces like a piece of cake by each solver to store the data. The memory system used by the fortran solvers is the fortran 77 "assumed-size-dummy-arrays" mechanism, based on "real arrayname(*)" statements.

The management of memory is very important for large-scale problems, where n is from 100 to 1000. One main feature of one of the L-BFGS algorithms is to limit the memory required. More precisely, the following is a map from the algorithm to the memory required, as the number of required double precision values.

- Quasi-Newton BFGS "qn" without constraints (n1qn1) : $n(n+13)/2$,
- Quasi-Newton BFGS "qn" with bounds constraints (qnbd) : $n(n+1)/2+4n+1$,
- Limited Memory BFGS "gc" without constraints (n1qn3) : $4n+m(2n+1)$,
- Limited Memory BFGS "gc" with bounds constraints (gcbd) : $n(5+3nt)+2nt+1$ with $nt=max(1,m/3)$,
- Non smooth method without constraints (n1fc1) : $(n+4)m/2+(m+9)*m+8+5n/2$.

Note that n1fc1 requires an additionnal $2(m+1)$ array of integers. Simplifying these array sizes leads to the following map, which clearly shows why Limited Memory BFGS algorithms in Scilab are more suitable for large problems. This explains why the name "cg" was chosen: it refers to the Conjugate Gradient method, which stores only one vector in memory. But the name "cg" is wrongly chosen and this is why we consistently use L-BFGS to identify this algorithm.

- Quasi-Newton "qn" without constraints (n1qn1) : $O(n^2)$,
- Quasi-Newton "qn" with bounds constraints (qnbd) : $O(n^2)$,
- Limited Memory BFGS "gc" without constraints (n1qn3) : $O(n)$,
- Limited Memory BFGS "gc" with bounds constraints (gcbd) : $O(n)$,
- Non smooth method without constraints (n1fc1) : $O(n)$.

That explains why L-BFGS methods associated with the "gc" option of the optim primitive are recommended for large-scale optimization. It is known that L-BFGS convergence may be slow for large-scale problems (see [21], chap. 9).

1.7 Quasi-Newton "qn" without constraints : n1qn1

The author is C. Lemarechal, 1987. There is no reference report for this solver.

The following is the header for the n1qn1 routine :

```
      subroutine n1qn1 (simul,n,x,f,g,var,eps,
     1     mode,niter,nsim,imp,lp,zm,izs,rzs,dzs)
c!but
c     minimisation d une fonction reguliere sans contraintes
c!origine
c     c. lemarechal, inria, 1987
c     Copyright INRIA
c!methode
c     direction de descente calculee par une methode de quasi-newton
c     recherche lineaire de type wolfe
```

The following is a description of the arguments of this routine.

- simul : point d'entree au module de simulation (cf normes modulopt i). n1qn1 appelle toujours simul avec indic = 4 ; le module de simulation doit se presenter sous la forme subroutine simul(n,x, f, g, izs, rzs, dzs) et être declare en external dans le programme appelant n1qn1.
- n (e) : nombre de variables dont depend f.
- x (e-s) : vecteur de dimension n ; en entree le point initial ; en sortie : le point final calcule par n1qn1.
- f (e-s) : scalaire ; en entree valeur de f en x (initial), en sortie valeur de f en x (final).
- g (e-s) : vecteur de dimension n : en entree valeur du gradient en x (initial), en sortie valeur du gradient en x (final).
- var (e) : vecteur strictement positif de dimension n. amplitude de la modif souhaitee a la premiere iteration sur x(i).une bonne valeur est 10% de la difference (en valeur absolue) avec la coordonee x(i) optimale
- eps (e-s) : en entree scalaire definit la precision du test d'arret. Le programme considere que la convergence est obtenue lorque il lui est impossible de diminuer f en attribuant à au moins une coordonnée x(i) une variation superieure a eps*var(i). En sortie, eps contient le carré de la norme du gradient en x (final).
- mode (e) : definit l'approximation initiale du hessien
  - =1 n1qn1 l'initialise lui-meme
  - =2 le hessien est fourni dans zm sous forme compressee (zm contient les colonnes de la partie inferieure du hessien)
- niter (e-s) : en entree nombre maximal d'iterations : en sortie nombre d'iterations reellement effectuees.
- nsim (e-s) : en entree nombre maximal d'appels a simul (c'est a dire avec indic = 4). en sortie le nombre de tels appels reellement faits.
- imp (e) : contrôle les messages d'impression :

- = 0 rien n'est imprime
- = 1 impressions initiales et finales
- = 2 une impression par iteration (nombre d'iterations, nombre d'appels a simul, valeur courante de f).
- >=3 informations supplementaires sur les recherches lineaires ; tres utile pour detecter les erreurs dans le gradient.

- lp (e) : le numero du canal de sortie, i.e. les impressions commandees par imp sont faites par write (lp, format).
- zm : memoire de travail pour n1qn1 de dimension n*(n+13)/2.
- izs,rzs,dzs memoires reservees au simulateur (cf doc)

The n1qn1 solver is an interface to the n1qn1a routine, which really implements the optimization method. The n1qn1a file counts approximately 300 lines. The n1qn1a routine is based on the following routines :

- simul : computes the cost function,
- majour : probably (there is no comment) an update of the BFGS matrix.

Many algorithms are in-lined, especially the line search and the linear algebra.

1.7.1 Management of the approximated Hessian matrix

The current BFGS implementation is based on a approximation of the Hessian [21], which is based on Cholesky decomposition, i.e. the approximated Hessian matrix is decomposed as $G = LDL^T$, where D is a diagonal $n \times n$ matrix and L is a lower triangular $n \times n$ matrix with unit diagonal. To compute the descent direction, the linear system $Gd = LDL^T d = -g$ is solved.

The memory requirements for this method is $O(n^2)$ because the approximated Hessian matrix computed from the BFGS formula is stored in compressed form so that only the lower part of the approximated Hessian matrix is stored. This is why this method is not recommended for large-scale problems (see [21], chap.9, introduction).

The approximated hessian $H \in \mathbb{R}^{n \times n}$ is stored as the vector $h \in \mathbb{R}^{n_h}$ which has size $n_h = n(n+1)/2$. The matrix is stored in factored form as following

$$h = (D_{11}L_{21} \dots L_{n1} | H_{21}D_{22} \dots L_{n2} | \dots | D_{n-1n-1}L_{nn-1} | D_{nn}) \,. \tag{1.1}$$

Instead of a direct acces to the factors of D and L, integers algebra is necessary to access to the data stored in the vector h.

The algorithm presented in figure 1.1 is used to set the diagonal terms the diagonal terms of D, the diagonal matrix of the Cholesky decomposition of the approximated Hessian. The right-hand side $\frac{0.01 c_{max}}{v_i^2}$ of this initialization is analysed in the next section of this document.

$k \leftarrow 1$
for $i = 1$ to n **do**
  $h(k) = \frac{0.01 c_{max}}{v_i^2}$
  $k \leftarrow k + n + 1 - i$
end for

Figure 1.1: Loop over the diagonal terms of the Cholesky decomposition of the approximated Hessian

Solving the linear system of equations

The linear system of equations $Gd = LDL^T d = -g$ must be solved to computed the descent direction $d \in \mathbb{R}^n$. This direction is computed by the following algorithm

- compute w so that $Lw = -g$,
- computed d so that $DL^T d = w$.

This algorithm requires $O(n^2)$ operations.

1.7.2 Line search

The line search is based on the algorithms developped by Lemaréchal [26]. It uses a cubic interpolation.

The Armijo condition for sufficient decrease is used in the following form

$$f(x_k + \alpha p_k) - f(x_k) \leq c_1 \alpha \nabla f_k^T p_k \tag{1.2}$$

with $c_1 = 0.1$. The following fortran source code illustrates this condition

```
if (fb-fa.le.0.10d+0*c*dga) go to 280
```

where $fb = f(x_k + \alpha p_k)$, $fa = f(x_k)$, $c = \alpha$ and $dga = \nabla f_k^T p_k$ is the local directional derivative.

1.7.3 Initial Hessian matrix

Several modes are available to compute the initial Hessian matrix, depending on the value of the *mode* variable

- if mode = 1, n1qn1 initializes the matrix by itself,
- if mode = 2, the hessian is provided in compressed form, where only the lower part of the symetric hessian matrix is stored.

An additionnal mode=3 is provided but the feature is not clearly documented. In Scilab, the n1qn1 routine is called with mode = 1 by default. In the case where a hot-restart is performed, the mode = 3 is enabled.

If mode = 1 is chosen, the initial Hessian matrix H^0 is computed by scaling the identity matrix

$$H^0 = I\delta \tag{1.3}$$

where $\delta \in \mathbb{R}^n$ is a n-vector and I is the $n \times n$ identity matrix. The scaling vector $\delta \in \mathbb{R}^n$ is based on the gradient at the initial guess $g^0 = g(x_0) = \nabla f(x_0) \in \mathbb{R}^n$ and a scaling vector $v \in \mathbb{R}^n$, given by the user

$$\delta_i = \frac{0.01 c_{max}}{v_i^2} \tag{1.4}$$

where $c_{max} > 0$ is computed by

$$c_{max} = \max\left(1.0, \max_{i=1,n}(|g_i^0)|)\right) \tag{1.5}$$

In the Scilab interface for *optim*, the scaling vector is set to 0.1 :

$$v_i = 0.1, \qquad i = 1, n. \tag{1.6}$$

1.7.4 Termination criteria

The following list of parameters are taken into account by the solver

- `niter`, the maximum number of iterations (default value is 100),
- `nap`, the maximum number of function evaluations (default value is 100),
- `epsg`, the minimum length of the search direction (default value is $\%eps \approx 2.22e-16$).

The other parameters `epsf` and `epsx` are not used. The termination condition is not based on the gradient, as the name `epsg` would indicate.

The following is a list of termination conditions which are taken into account in the source code.

- The iteration is greater than the maximum.

```
if (itr.gt.niter) go to 250
```

- The number of function evaluations is greater than the maximum.

```
if (nfun.ge.nsim) go to 250
```

- The directionnal derivative is positive, so that the direction d is not a descent direction for f.

```
if (dga.ge.0.0d+0) go to 240
```

- The cost function set the indic flag (the `ind` parameter) to 0, indicating that the optimization must terminate.

```
call simul (indic,n,xb,fb,gb,izs,rzs,dzs)
[...]
go to 250
```

- The cost function set the indic flag to a negative value indicating that the function cannot be evaluated for the given x. The step is reduced by a factor 10, but gets below a limit so that the algorithm terminates.

```
call simul (indic,n,xb,fb,gb,izs,rzs,dzs)
[...]
step=step/10.0d+0
[...]
if (stepbd.gt.steplb) goto 170
[...]
go to 250
```

- The Armijo condition is not satisfied and step size is below a limit during the line search.

```
if (fb-fa.le.0.10d+0*c*dga) go to 280
[...]
if (step.gt.steplb) go to 270
```

- During the line search, a cubic interpolation is computed and the computed minimum is associated with a zero step length.

```
if(c.eq.0.0d+0) goto 250
```

- During the line search, the step length is lesser than a computed limit.

```
if (stmin+step.le.steplb) go to 240
```

- The rank of the approximated Hessian matrix is lesser than n after the update of the Cholesky factors.

```
if (ir.lt.n) go to 250
```

1.7.5 An example

The following script illustrates that the gradient may be very slow, but the algorithm continues. This shows that the termination criteria is not based on the gradient, but on the length of the step. The problem has two parameters so that $n = 2$. The cost function is the following

$$f(\mathbf{x}) = x_1^p + x_2^p \tag{1.7}$$

where $p \geq 0$ is an even integer. Here we choose $p = 10$. The gradient of the function is

$$g(\mathbf{x}) = \nabla f(\mathbf{x}) = (px_1^{p-1}, px_2^{p-1})^T \tag{1.8}$$

and the Hessian matrix is

$$H(\mathbf{x}) = \begin{pmatrix} p(p-1)x_1^{p-2} & 0 \\ 0 & p(p-1)x_2^{p-2} \end{pmatrix} \tag{1.9}$$

The optimum of this optimization problem is at

$$\mathbf{x}^\star = (0,0)^T. \tag{1.10}$$

The following Scilab script defines the cost function, checks that the derivatives are correctly computed and performs an optimization. At each iteration, the norm of the gradient of the cost function is displayed so that one can see if the algorithm terminates when the gradient is small.

```
function [ f , g , ind ] = myquadratic ( x , ind )
  p = 10
  if ind == 1 | ind == 2 | ind == 4 then
    f = x(1)^p + x(2)^p;
  end
  if ind == 1 | ind == 2 | ind == 4 then
    g(1) = p * x(1)^(p-1)
    g(2) = p * x(2)^(p-1)
  end
  if ind == 1 then
    mprintf("|x|=%e, f=%e, |g|=%e\n",norm(x),f,norm(g))
  end
endfunction
function f = quadfornumdiff ( x )
  f = myquadratic ( x , 2 )
endfunction
x0 = [-1.2 1.0];
[ f , g ] = myquadratic ( x0 , 4 );
mprintf ( "Computed f(x0) = %f\n", f);
mprintf ( "Computed g(x0) = \n");disp(g');
mprintf ( "Expected g(x0) = \n");disp(derivative(quadfornumdiff,x0'))
nap = 100
iter = 100
epsg = %eps
[ fopt , xopt , gradopt ] = optim ( myquadratic , x0 , ...
  "ar" , nap , iter , epsg , imp = -1)
```

The script produces the following output.

```
-->[ fopt , xopt , gradopt ] = optim ( myquadratic , x0 , ...
   "ar" , nap , iter , epsg , imp = -1)
|x|=1.562050e+000, f=7.191736e+000, |g|=5.255790e+001
|x|=1.473640e+000, f=3.415994e+000, |g|=2.502599e+001
|x|=1.098367e+000, f=2.458198e+000, |g|=2.246752e+001
|x|=1.013227e+000, f=1.092124e+000, |g|=1.082542e+001
|x|=9.340864e-001, f=4.817592e-001, |g|=5.182592e+000
[...]
|x|=1.280564e-002, f=7.432396e-021, |g|=5.817126e-018
|x|=1.179966e-002, f=3.279551e-021, |g|=2.785663e-018
|x|=1.087526e-002, f=1.450507e-021, |g|=1.336802e-018
|x|=1.002237e-002, f=6.409611e-022, |g|=6.409898e-019
|x|=9.236694e-003, f=2.833319e-022, |g|=3.074485e-019
Norm of projected gradient lower than    0.3074485D-18.
```

```
gradopt  =
  1.0D-18 *
   0.2332982      0.2002412
xopt  =
   0.0065865      0.0064757
fopt  =
   2.833D-22
```

One can see that the algorithm terminates when the gradient is extremely small $g(\mathbf{x}) \approx 10^{-18}$. The cost function is very near zero $f(\mathbf{x}) \approx 10^{-22}$, but the solution is not accurate only up to the 3d digit.

This is a very difficult test case for optimization solvers. The difficulty is because the function is extremely *flat* near the optimum. If the termination criteria was based on the gradient, the algorithm would stop in the early iterations. Because this is not the case, the algorithm performs significant iterations which are associated with relatively large steps.

1.8 Quasi-Newton "qn" with bounds constraints : qnbd

The comments state that the reference report is an INRIA report by F. Bonnans [4]. The solver qnbd is an interface to the zqnbd routine. The zqnbd routine is based on the following routines :

- calmaj : calls majour, which updates the BFGS matrix,
- proj : projects the current iterate into the bounds constraints,
- ajour : probably (there is no comment) an update of the BFGS matrix,
- rlbd : line search method with bound constraints,
- simul : computes the cost function

The rlbd routine is documented as using an extrapolation method to computed a range for the optimal t parameter. The range is then reduced depending on the situation by :

- a dichotomy method,
- a linear interpolation,
- a cubic interpolation.

The stopping criteria is commented as "an extension of the Wolfe criteria". The linear algebra does not use the BLAS API. It is in-lined, so that connecting the BLAS may be difficult. The memory requirements for this method are $O(n^2)$, which shows why this method is not recommended for large-scale problems (see [21], chap.9, introduction).

1.9 L-BFGS "gc" without constraints : n1qn3

The comments in this solver are clearly written. The authors are Jean Charles Gilbert, Claude Lemarechal, 1988. The BFGS update is based on the article [34]. The solver n1qn3 is an interface to the n1qn3a routine. The architecture is clear and the source code is well commented. The n1qn3a routine is based on the following routines :

- prosca : performs a vector x vector scalar product,
- simul : computes the cost function,
- nlis0 : line search based on Wolfe criteria, extrapolation and cubic interpolation,
- ctonb : copies array u into v,
- ddd2 : computed the descent direction by performing the product hxg.

The linear algebra is dense, which limits the feature to small size optimization problems. The linear algebra does not use the BLAS API but is based on the prosca and ctonb routines. The prosca routine is a call back input argument of the n1qn3 routine, connected to the fuclid routine. This implements the scalar product, but without optimizaion. Connecting BLAS may be easy for n1qn3. The algorithm is a limited memory BFGS method with m levels, so that the memory cost is $O(n)$. It is well suited for medium-scale problems, although convergence may be slow (see [21], chap. 9, p.227).

1.10 L-BFGS "gc" with bounds constraints : gcbd

The author is F. Bonnans, 1985. There is no reference report for gcbd. The gcbd solver is an interface to the zgcbd routine, which really implements the optimization method. The zgcbd routine is based on the following routines :

- simul : computes the cost function
- proj : projects the current iterate into the bounds constraints,
- majysa : updates the vectors $y = g(k+1) - g(k)$, $s = x(k+1) - x(k)$, ys,
- bfgsd : updates the diagonal by Powell diagonal BFGS,
- shanph : scalse the diagonal by Shanno-Phua method,
- majz : updates z,zs,
- relvar : computes the variables to relax by Bertsekas method,
- gcp : gradient conjugate method for $Ax = b$,
- dcopy : performs a vector copy (BLAS API),
- rlbd : line search method with bound constraints.

The linear algebra is dense. But zgcbd uses the "dcopy" BLAS routine, which allows for some optimizations. The algorithm is a limited memory BFGS method with m levels, so that the memory cost is $O(n)$. It is well suited for medium-scale problems, although the convergence may be slow (see [21], chap. 9, p.227).

1.11 Non smooth method without constraints : n1fc1

This routine is probably due to Lemaréchal, who is an expert of this topic. References for this algorithm include the "Part II, Nonsmooth optimization" in [5], and the in-depth presentation in [18, 19].

The n1fc1 solver is an interface to the n1fc1a routine, which really implements the optimization method. The n1fc1a routine is based on the following routines :

- simul : computes the cost function,
- fprf2 : computes the direction,
- frdf1 : reduction du faisceau
- nlis2 : line search,
- prosca : performs a vector x vector scalar product.

It is designed for functions which have a non-continuous derivative (e.g. the objective function is the maximum of several continously differentiable functions).

Chapter 2

Quadratic optimization

Quadratic problems can be solved with the `qpsolve` Scilab macro and the `qp_solve` Scilab primitive. In this chapter, we presents these two primitives, which are meant to be a replacement for the former Quapro solver (which has be transformed into a Scilab toolbox). We especially analyse the management of the linear algebra as well as the memory requirements of these solvers.

2.1 Mathematical point of view

This kind of optimization is the minimization of function $f(x)$ with

$$f(x) = \frac{1}{2}x^T Q x + p^T x$$

under the constraints :

$$C_1^T x = b_1 \quad (2.1)$$
$$C_2^T x \geq b_2 \quad (2.2)$$
$$(2.3)$$

2.2 qpsolve

The Scilab function `qpsolve` is a solver for quadratic problems when Q is symmetric positive definite.

The `qpsolve` function is a Scilab macro which aims at providing the same interface (that is, the same input/output arguments) as the quapro solver.

For more details about this solver, please refer to Scilab online help for `qpsolve`

The `qpsolve` Scilab macro is based on the work by Berwin A Turlach from The University of Western Australia, Crawley [38]. The solver is implemented in Fortran 77. This routine uses the Goldfarb/Idnani algorithm [11, 12].

The constraints matrix can be dense or sparse.

The `qpsolve` macro calls the `qp_solve` compiled primitive. The internal source code for `qpsolve` manages the equality and inequality constraints so that it can be processed by the `qp_solve` primitive.

2.3 qp_solve

The `qp_solve` compiled primitive is an interface for the fortran 77 solver. The interface is implemented in the C source code sci_qp_solve.c. Depending on the constraints matrix, a dense or sparse solver is used :

- If the constraints matrix C is dense, the qpgen2 fortran 77 routine is used. The qpgen2 routine is the original, unmodified algorithm which was written by Turlach (the original name was solve.QP.f)
- If the constraints matrix C is a Scilab sparse matrix, the qpgen1sci routine is called. This routine is a modification of the original routine qpgen1, in order to adapt to the specific Scilab sparse matrices storage.

2.4 Memory requirements

Suppose that n is the dimension of the quadratic matrix Q and m is the sum of the number of equality constraints me and inequality constraints md. Then, the temporary work array which is allocated in the primitive has the size

$$r = \min(n, m), \tag{2.4}$$
$$lw = 2n + r(r + 5)/2 + 2m + 1. \tag{2.5}$$

This temporary array is de-allocated when the qpsolve primitive returns.
This formulae may be simplified in the following cases :

- if $n \gg m$, that is the number of constraints m is negligible with respect to the number of unknowns n, then the memory required is $O(n)$,
- if $m \gg n$, that is the number of unknowns n is negligible with respect to the number of constraints m, then the memory required is $O(m)$,
- if $m = n$, then the memory required is $O(n^2)$.

2.5 Internal design

The original Goldfarb/Idnani algorithm [11, 12] was designed to solve the following minimization problem:

$$\min_x -d^T x + \frac{1}{2} x^T D x$$

where

$$A_1^T x = b_1 \tag{2.6}$$
$$A_2^T x >= b_2 \tag{2.7}$$

where the matrix D is assumed to be symmetric positive definite. It was considered as a building block for a Sequential Quadratic Programming solver. The original package provides two routines :

- solve.QP.f containing routine qpgen2 which implements the algorithm for dense matrices,
- solve.QP.compact.f containing routine qpgen1 which implements the algorithm for sparse matrices.

Chapter 3

Non-linear least square

3.1 Mathematical point of view

The problem of the non linear least-quare optimization is to find the solution of

$$\min_x \sum_x f(x)^2$$

with bounds constraints or without constraints and with $f : \mathbb{R}^n \to \mathbb{R}^m$ the cost function.

3.2 Scilab function

Scilab function called *lsqrsolve* is designed for the minimization of the sum of the squares of non-linear functions using a Levenberg-Marquardt algorithm. For more details about this function, please refer to Scilab online help

3.3 Optimization routines

Scilab *lsqrsolve* function is based on the routines *lmdif* and *lmder* of the library MINPACK (Argonne National Laboratory).

Chapter 4

Semidefinite programming

4.1 Mathematical point of view

This kind of optimization is the minimization of $f(x) = c^T x$ under the constraint:

$$F_0 + x_1 F_1 + ... + x_m F_m \geq 0 \tag{4.1}$$

or its dual problem, the maximization or $-Trace(F_0, Z)$ under the constraints:

$$Trace(F_i, Z) = c_i, i = 1, ..., m \tag{4.2}$$

$$Z \geq 0 \tag{4.3}$$

4.2 Scilab function

The Scilab function called *semidef* is designed for this kind of optimization problems. For more details about this function, please refer to Scilab online help

4.3 Optimization routines

Scilab *semidef* function is based on a routine from L. Vandenberghe and Stephen Boyd.

Chapter 5

Genetic algorithms

5.1 Introduction

Genetic algorithms are search algorithms based on the mechanics on natural selection and natural genetics [17, 28]. Genetic algorithms have been introduced in Scilab v5 thanks to a work by Yann Collette [9]. The solver is made of Scilab macros, which enables a high-level programming model for this optimization solver.

The problems solved by the current genetic algorithms in Scilab are the following :

- minimization of a cost function with bound constraints,
- multi-objective non linear minimization with bound constraints.

Genetic algorithms are different from more normal optimization and search procedures in four ways :

- GAs work with a coding of the parameter set, not the parameters themselves,
- GAs search from a population of points, not a single point,
- GAs use payoff (objective function) information, not derivativess or other auxiliary knowledge,
- GAs use probabilistic transition rules, not deterministic rules.

A simple genetic algorithm that yields good results in many practical problems is composed of three operators [17] :

- reproduction,
- cross-over,
- mutation.

Many articles on this subject have been collected by Carlos A. Coello Coello on his website [7]. A brief introduction to GAs is done in [43].

The GA macros are based on the "parameters" Scilab module for the management of the (many) optional parameters.

5.2 Example

In the current section, we give an example of the user of the GA algorithms.

The following is the definition of the Rastrigin function.

```
function Res = min_bd_rastrigin()
Res = [-1 -1]';
endfunction
function Res = max_bd_rastrigin()
Res = [1 1]';
endfunction
function Res = opti_rastrigin()
Res = [0 0]';
endfunction
function y = rastrigin(x)
  y = x(1)^2+x(2)^2-cos(12*x(1))-cos(18*x(2));
endfunction
```

This cost function is then defined with the generic name "f". Other algorithmic parameters, such as the size of the population, are defined in the following sample Scilab script.

```
func = 'rastrigin';
deff('y=f(x)','y = '+func+'(x)');
PopSize     = 100;
Proba_cross = 0.7;
Proba_mut   = 0.1;
NbGen       = 10;
NbCouples   = 110;
Log         = %T;
nb_disp     = 10;
pressure    = 0.05;
```

Genetic Algorithms require many settings, which are cleanly handled by the "parameters" module. This module provides the `nit_param` function, which returns a new, empty, set of parameters. The `add_param` function allows to set individual named parameters, which are configure with key-value pairs.

```
ga_params = init_param();
// Parameters to adapt to the shape of the optimization problem
ga_params = add_param(ga_params,'minbound',eval('min_bd_'+func+'()'));
ga_params = add_param(ga_params,'maxbound',eval('max_bd_'+func+'()'));
ga_params = add_param(ga_params,'dimension',2);
ga_params = add_param(ga_params,'beta',0);
ga_params = add_param(ga_params,'delta',0.1);
// Parameters to fine tune the Genetic algorithm.
ga_params = add_param(ga_params,'init_func',init_ga_default);
ga_params = add_param(ga_params,'crossover_func',crossover_ga_default);
ga_params = add_param(ga_params,'mutation_func',mutation_ga_default);
ga_params = add_param(ga_params,'codage_func',coding_ga_identity);
```

```
13 ga_params = add_param(ga_params,'selection_func',selection_ga_elitist);
14 ga_params = add_param(ga_params,'nb_couples',NbCouples);
15 ga_params = add_param(ga_params,'pressure',pressure);
```

The `optim_ga` function search a population solution of a single-objective problem with bound constraints.

```
1 [pop_opt, fobj_pop_opt, pop_init, fobj_pop_init] = ...
2    optim_ga(f, PopSize, NbGen, Proba_mut, Proba_cross, Log, ga_params);
```

The following are the messages which are displayed in the Scilab console :

```
optim_ga: Initialization of the population
optim_ga: iteration 1 / 10 - min / max value found = -1.682413 / 0.081632
optim_ga: iteration 2 / 10 - min / max value found = -1.984184 / -0.853613
optim_ga: iteration 3 / 10 - min / max value found = -1.984184 / -1.314217
optim_ga: iteration 4 / 10 - min / max value found = -1.984543 / -1.513463
optim_ga: iteration 5 / 10 - min / max value found = -1.998183 / -1.691332
optim_ga: iteration 6 / 10 - min / max value found = -1.999551 / -1.871632
optim_ga: iteration 7 / 10 - min / max value found = -1.999977 / -1.980356
optim_ga: iteration 8 / 10 - min / max value found = -1.999979 / -1.994628
optim_ga: iteration 9 / 10 - min / max value found = -1.999989 / -1.998123
optim_ga: iteration 10 / 10 - min / max value found = -1.999989 / -1.999534
```

The initial and final populations for this simulation are shown in 5.1.
The following script is a loop over the optimum individuals of the population.

```
1 printf('Genetic Algorithm: %d points from pop_opt\n', nb_disp);
2 for i=1:nb_disp
3 printf('Individual %d: x(1) = %f x(2) = %f -> f = %f\n', ...
4   i, pop_opt(i)(1), pop_opt(i)(2), fobj_pop_opt(i));
5 end
```

The previous script make the following lines appear in the Scilab console.

```
Individual 1: x(1) = -0.000101 x(2) = 0.000252 -> f = -1.999989
Individual 2: x(1) = -0.000118 x(2) = 0.000268 -> f = -1.999987
Individual 3: x(1) = 0.000034 x(2) = -0.000335 -> f = -1.999982
Individual 4: x(1) = -0.000497 x(2) = -0.000136 -> f = -1.999979
Individual 5: x(1) = 0.000215 x(2) = -0.000351 -> f = -1.999977
Individual 6: x(1) = -0.000519 x(2) = -0.000197 -> f = -1.999974
Individual 7: x(1) = 0.000188 x(2) = -0.000409 -> f = -1.999970
Individual 8: x(1) = -0.000193 x(2) = -0.000427 -> f = -1.999968
Individual 9: x(1) = 0.000558 x(2) = 0.000260 -> f = -1.999966
Individual 10: x(1) = 0.000235 x(2) = -0.000442 -> f = -1.999964
```

5.3 Support functions

In this section, we analyze the GA services to configure a GA computation.

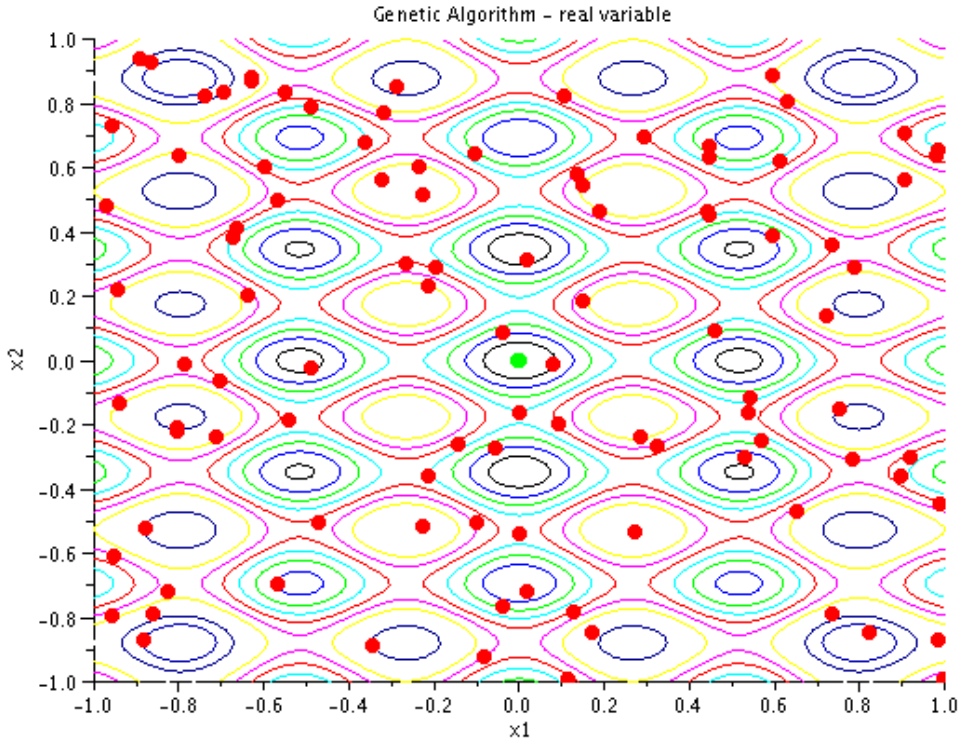

Figure 5.1: Optimum of the Rastrigin function – Initial population is in red, Optimum population is accumulated on the blue dot

5.3.1 Coding

The following is the list of coding functions available in Scilab's GA :

- `coding_ga_binary` : A function which performs conversion between binary and continuous representation
- `coding_ga_identity` : A "no-operation" conversion function

The user may configure the GA parameters so that the algorithm uses a customized coding function.

5.3.2 Cross-over

The crossover function is used when mates have been computed, based on the Wheel algorithm : the crossover algorithm is a loop over the couples, which modifies both elements of each couple.

The following is the list of crossover functions available in Scilab :

- `crossover_ga_default` : A crossover function for continuous variable functions.
- `crossover_ga_binary` : A crossover function for binary code

5.3.3 Selection

The selection function is used in the loop over the generations, when the new population is computed by processing a selection over the individuals.

The following is the list of selection functions available in Scilab :

- `selection_ga_random` : A function which performs a random selection of individuals. We select pop_size individuals in the set of parents and childs individuals at random.
- `selection_ga_elitist` : An 'elitist' selection function. We select the best individuals in the set of parents and childs individuals.

5.3.4 Initialization

The initialization function returns a population as a list made of "pop_size" individuals. The Scilab macro `init_ga_default` computes this population by performing a randomized discretization of the domain defined by the bounds as minimum and maximum arrays. This randomization is based on the Scilab primitive `rand`.

5.4 Solvers

In this section, we analyze the 4 GA solvers which are available in Scilab :

- `optim_ga` : flexible genetic algorithm
- `optim_moga` : multi-objective genetic algorithm
- `optim_nsga` : multi-objective Niched Sharing Genetic Algorithm

- `optim_nsga2` : multi-objective Niched Sharing Genetic Algorithm version 2

While `optim_ga` is designed for one objective, the 3 other solvers are designed for multi-objective optimization.

5.4.1 optim_ga

The Scilab macro `optim_ga` implements a Genetic Algorithm to find the solution of an optimization problem with one objective function and bound constraints.

The following is an overview of the steps in the GA algorithm.

- processing of input arguments

  In the case where the input cost function is a list, one defines the "hidden" function `_ga_f` which computes the cost function. If the input cost function is a regular Scilab function, the "hidden" function `_ga_f` simply encapsulate the input function.

- initialization

  One computes the initial population with the `init_func` callback function (the default value for `init_func` is `init_ga_default`)

- coding

  One encodes the initial population with the `codage_func` callback function (default : `coding_ga_ide`

- evolutionary algorithm as a loop over the generations

- decoding

  One decodes the optimum population back to the original variable system

The loop over the generation is made of the following steps.

- reproduction : two list of children populations are computed, based on a randomized Wheel,

- crossover : the two populations are processed through the `crossover_func` callback function (default : `crossover_ga_default`)

- mutation : the two populations are processed throught the `mutation_func` callback function (default : `mutation_ga_default`)

- computation of cost functions : the `_ga_f` function is called to compute the fitness for all individuals of the two populations

- selection : the new generation is computed by processing the two populations through the `selection_func` callback function (default : `selection_ga_elitist`)

5.4.2 optim_moga, pareto_filter

The `optim_moga` function is a multi-objective genetic algorithm. The method is based on [15].

The function `pareto_filter` extracts non dominated solution from a set.

5.4.3 optim_nsga

The `optim_nsga` function is a multi-objective Niched Sharing Genetic Algorithm. The method is based on [37].

5.4.4 optim_nsga2

The function `optim_nsga2` is a multi-objective Niched Sharing Genetic Algorithm. The method is based on [14].

Chapter 6

Simulated Annealing

In this document, we describe the Simulated Annealing optimization methods, a new feature available in Scilab v5 .

6.1 Introduction

Simulated annealing (SA) is a generic probabilistic meta-algorithm for the global optimization problem, namely locating a good approximation to the global optimum of a given function in a large search space. It is often used when the search space is discrete (e.g., all tours that visit a given set of cities) [42].

Genetic algorithms have been introduced in Scilab v5 thanks to the work by Yann Collette [9].

The current Simulated Annealing solver aims at finding the solution of

$$\min_x f(x)$$

with bounds constraints and with $f : \mathbb{R}^n \to \mathbb{R}$ the cost function.

Reference books on the subject are [24, 25, 13].

6.2 Overview

The solver is made of Scilab macros, which enables a high-level programming model for this optimization solver. The GA macros are based on the "parameters" Scilab module for the management of the (many) optional parameters.

To use the SA algorithm, one must perform the following steps :

- configure the parameters with calls to "init_param" and "add_param" especially,
  - the neighbor function,
  - the acceptance function,
  - the temperature law,
- compute an initial temperature with a call to "compute_initial_temp"
- find an optimum by using the "optim_sa" solver

6.3 Example

The following example is extracted from the SA examples. The Rastrigin functin is used as an example of a dimension 2 problem because it has many local optima but only one global optimum.

```
1  //
2  // Rastrigin function
3  //
4  function Res = min_bd_rastrigin()
5  Res = [-1  -1]';
6  endfunction
7  function Res = max_bd_rastrigin()
8  Res = [1  1]';
9  endfunction
10 function Res = opti_rastrigin()
11 Res = [0  0]';
12 endfunction
13 function y = rastrigin(x)
14   y = x(1)^2+x(2)^2-cos(12*x(1))-cos(18*x(2));
15 endfunction
16 //
17 // Set parameters
18 //
19 func = 'rastrigin';
20 Proba_start = 0.8;
21 It_intern = 1000;
22 It_extern = 30;
23 It_Pre    = 100;
24 Min = eval('min_bd_'+func+'()');
25 Max = eval('max_bd_'+func+'()');
26 x0  = (Max - Min).*rand(size(Min,1),size(Min,2)) + Min;
27 deff('y=f(x)','y='+func+'(x)');
28 //
29 // Simulated Annealing with default parameters
30 //
31   printf('SA: geometrical decrease temperature law\n');
32
33   sa_params = init_param();
34   sa_params = add_param(sa_params,'min_delta',-0.1*(Max-Min));
35   sa_params = add_param(sa_params,'max_delta', 0.1*(Max-Min));
36   sa_params = add_param(sa_params,'neigh_func', neigh_func_default);
37   sa_params = add_param(sa_params,'accept_func', accept_func_default);
38   sa_params = add_param(sa_params,'temp_law', temp_law_default);
39   sa_params = add_param(sa_params,'min_bound',Min);
40   sa_params = add_param(sa_params,'max_bound',Max);
41
42   T0 = compute_initial_temp(x0, f, Proba_start, It_Pre, sa_params);
43   printf('Initial temperature T0 = %f\n', T0);
44
45   [x_opt, f_opt, sa_mean_list, sa_var_list, temp_list] = ...
```

```
46        optim_sa(x0, f, It_extern, It_intern, T0, Log = %T, sa_params);
47
48    printf('optimal solution:\n'); disp(x_opt);
49    printf('value of the objective function = %f\n', f_opt);
50
51    scf();
52    drawlater;
53    subplot(2,1,1);
54    xtitle('Geometrical annealing','Iteration','Mean / Variance');
55    t = 1:length(sa_mean_list);
56    plot(t,sa_mean_list,'r',t,sa_var_list,'g');
57    legend(['Mean','Variance']);
58    subplot(2,1,2);
59    xtitle('Temperature evolution','Iteration','Temperature');
60    for i=1:length(t)-1
61      plot([t(i) t(i+1)], [temp_list(i) temp_list(i)],'k-');
62    end
63    drawnow;
```

After some time, the following messages appear in the Scilab console.

```
optimal solution:
  - 0.0006975
  - 0.0000935
value of the objective function = -1.999963
```

The figure 6.1 presents the evolution of Mean, Variance and Temperature depending on the iteration.

6.4 Neighbor functions

In the simulated annealing algorithm, a neighbour function is used in order to explore the domain [43].

The prototype of a neighborhood function is the following :

```
1  function x_neigh = neigh_func_default(x_current, T, param)
```

where:

- x_current represents the current point,
- T represents the current temperature,
- param is a list of parameters.

The following is a list of the neighbour functions available in the SA context :

- `neigh_func_default` : SA function which computes a neighbor of a given point. For example, for a continuous vector, a neighbor will be produced by adding some noise to each component of the vector. For a binary string, a neighbor will be produced by changing one bit from 0 to 1 or from 1 to 0.

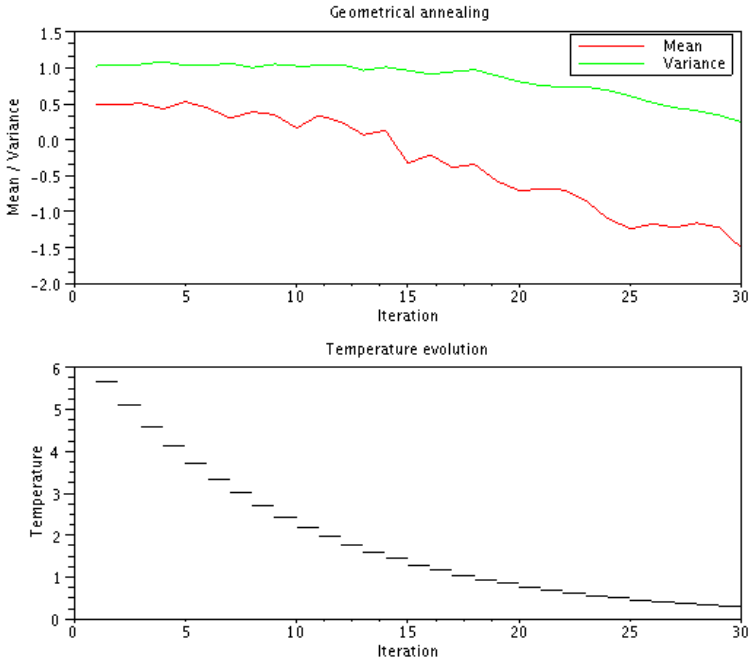

Figure 6.1: Convergence of the simulated annealing algorithm

- `neigh_func_csa` : The classical neighborhood relationship for the simulated annealing. The neighbors distribution is a gaussian distribution which is more and more peaked as the temperature decrease.
- `neigh_func_fsa` : The Fast Simulated Annealing neghborhood relationship. The corresponding distribution is a Cauchy distribution which is more and more peaked as the temperature decreases.
- `neigh_func_vfsa` : The Very Fast Simulated Annealing neighborhood relationship. This distribution is more and more peaked as the temperature decreases.

6.5 Acceptance functions

There exist several kind of simulated annealing optimization methods:

- the Fast Simulated Annealing,
- the simulated annealing based on metropolis-hasting acceptance function,
- etc...

To implement these various simulated annealing optimization methods, you only need to change the acceptance function. For common optimization, you need not to change the default acceptance function.

The following is a list of acceptance functions available in Scilab SAs :

- `accept_func_default` : is the default acceptance function, based on the exponential function

$$level = exp\left(-\frac{F_{neigh} - F_{current}}{T}\right)$$

- `accept_func_vfsa` : is the Very Fast Simulated Annealing function, defined by :

$$Level = \frac{1}{1 + exp\left(-\frac{F_{current} - F_{neigh}}{T}\right)}$$

6.6 Temperature laws

In the simulated annealing algorithm, a temperature law is used in a statistical criteria for the update of the optimum [43]. If the new (neighbor) point improves the current optimum, the update is done with the new point replacing the old optimum. If not, the update may still be processed, provided that a statistical criteria is satisfied. The statistical law decreases while the iterations are processed.

There are 5 temperature laws available in the SA context :

- `temp_law_default` : A SA function which computes the temperature of the next temperature stage
- `temp_law_csa` : The classical temperature decrease law, the one for which the convergence of the simulated annealing has been proven
- `temp_law_fsa` : The Szu and Hartley Fast simulated annealing
- `temp_law_huang` : The Huang temperature decrease law for the simulated annealing
- `temp_law_vfsa` : This function implements the Very Fast Simulated Annealing from L. Ingber

6.7 optim_sa

The `optim_sa` macro implements the simulated annealing solver. It allows to find the solution of an minimization problem with bound constraints.

It is based on an iterative update of two points :

- the current point is updated by taking into account the neighbour function and the acceptance criterium,
- the best point is the point which achieved the minimum of the objective function over the iterations.

While the current point is used internally to explore the domain, only the best point is returned as the algorithm output.

The algorithm is based on the following steps, which include a main, external loop over the temperature decreases, and an internal loop.

- processing of input arguments,
- initialization,
- loop over the number of temperature decreases.

For each iteration over the temperature decreases, the following steps are processed.

- loop over internal iterations, with constant temperature,
- if history is required by user, store the temperature, the x iterates, the values of f,
- update the temperature with the temperature law.

The internal loop allows to explore the domain and is based on the neighbour function. It is based on the following steps.

- compute a neighbour of the current point,
- compute the objective function for that neighbour
- if the objective decreases or if the acceptance criterium is true, then overwrite the current point with the neighbour
- if the cost of the best point is greater than the cost of the current point, overwrite the best point by the current point.

Chapter 7

LMITOOL: a Package for LMI Optimization in Scilab

R. Nikoukhah Ramine.Nikoukhah@inria.fr
F. Delebecque Francois.Delebecque@inria.fr
L. El Ghaoui ENSTA, 32, Bvd. Victor, 75739 Paris, France. Internet: elghaoui@ensta.fr.
Research supported in part by DRET under contract 92017-BC14

This chapter describes a user-friendly Scilab package, and in particular its two main functions `lmisolver` and `lmitool` for solving Linear Matrix Inequalities problems. This package uses Scilab function `semidef`, an interface to the program Semidefinite Programming **SP** (Copyright ©1994 by Lieven Vandenberghe and Stephen Boyd) distributed with Scilab.

7.1 Purpose

Many problems in systems and control can be formulated as follows (see [6]):

$$\Sigma : \begin{cases} \text{minimize} & f(X_1, \ldots, X_M) \\ \text{subject to} & \begin{cases} G_i(X_1, \ldots, X_M) = 0, & i = 1, 2, ..., p, \\ H_j(X_1, \ldots, X_M) \geq 0, & j = 1, 2, .., q. \end{cases} \end{cases}$$

where

- $X_1, \ldots, X_M$ are unknown real matrices, referred to as the *unknown matrices,*
- f is a real linear scalar function of the entries of the unknown matrices $X_1, \ldots, X_M$; it is referred to as the *objective function*,
- G_i's are real matrices with entries which are affine functions of the entries of the unknown matrices, $X_1, \ldots, X_M$; they are referred to as "Linear Matrix Equality" (LME) functions,
- H_j's are real symmetric matrices with entries which are affine functions of the entries of the unknown matrices $X_1, \ldots, X_M$; they are referred to as "Linear Matrix Inequality" (LMI) functions. (In this report, the $V \geq 0$ stands for V positive semi-definite unless stated otherwise).

The purpose of LMITOOL is to solve problem Σ in a user-friendly manner in Scilab, using the code SP [23]. This code is intended for small and medium-sized problems (say, up to a few hundred variables).

7.2 Function lmisolver

LMITOOL is built around the Scilab function lmisolver. This function computes the solution $X_1, \ldots, X_M$ of problem Σ, given functions f, G_i and H_j. To solve Σ, user must provide an evaluation function which "evaluates" f, G_i and H_j as a function the unknown matrices, as well as an initial guess on the values of the unknown matrices. User can either invoke lmisolver directly, by providing the necessary information in a special format or he can use the interactive function lmitool described in Section 7.3.

7.2.1 Syntax

```
[XLISTF[,OPT]] = lmisolver(XLIST0,EVALFUNC[,options])
```

where

- XLIST0: a list structure including matrices and/or list of matrices. It contains initial guess on the values of the unknown matrices. In general, the ith element of XLIST0 is the initial guess on the value of the unknown matrix X_i. In some cases however it is more convenient to define one or more elements of XLIST0 to be lists (of unknown matrices) themselves. This is a useful feature when the number of unknown matrices is not fixed a priori (see Example of Section 7.2.2).

  The values of the matrices in XLIST0, if compatible with the LME functions, are used as intial condition for the optimization algorithm; they are ignored otherwise. The size and structure of XLIST0 are used to set up the problem and determine the size and structure of the output XLISTF.

- EVALFUNC: a Scilab function called *evaluation function* (supplied by the user) which evaluates the LME, LMI and objective functions, given the values of the unknown matrices. The syntax is:

  ```
  [LME,LMI,OBJ]=EVALFUNC(XLIST)
  ```

  where

  - XLIST: a list, identical in size and structure to XLIST0.
  - LME: a list of matrices containing values of the LME functions G_i's for X values in XLIST. LME can be a matrix in case there is only one LME function to be evaluated (instead of a list containing this matrix as unique element). It can also be a list of a mixture of matrices and lists which in turn contain values of LME's, and so on.
  - LMI: a list of matrices containing the values of the LMI functions H_j's for X values in XLIST. LMI can also be a matrix (in case there is only one LMI function to be evaluated). It can also be a list of a mixture of matrices and lists which in turn contain values of of LMI's, and so on.

– `OBJ:` a scalar equal to the value of the objective function f for X values in `XLIST`.

If the Σ problem has no equality constraints then `LME` should be `[]`. Similarly for `LMI` and `OBJ`.

- `options:` a 5×1 vector containing optimization parameters `Mbound`, `abstol`, `nu`, `maxiters`, and `reltol`, see manual page for `semidef` for details (`Mbound` is a multiplicative coefficient for `M`). This argument is optional, if omitted, default parameters are used.
- `XLISTF:` a list, identical in size and structure to `XLIST0` containing the solution of the problem (optimal values of the unknown matrices).
- `OPT:` a scalar corresponding to the optimal value of the minimization problem Σ.

7.2.2 Examples

State-feedback with control saturation constraint

Consider the linear system

$$\dot{x} = Ax + Bu$$

where A is an $n \times n$ and B, an $n \times n_u$ matrix. There exists a stabilizing state feedback K such that for every initial condition $x(0)$ with $\|x(0)\| \leq 1$, the resulting control satisfies $\|u(t)\|$ for all $t \geq 0$, if and only if there exist an $n \times n$ matrix Q and an $n_u \times n$ matrix Y satisfying the equality constraint

$$Q - Q^T = 0$$

and the inequality constraints

$$\begin{aligned} Q &\geq 0 \\ -AQ - QA^T - BY - Y^T B^T &> 0 \\ \begin{pmatrix} u_{max}^2 I & Y \\ Y^T & Q \end{pmatrix} &\geq 0 \end{aligned}$$

in which case one such K can be constructed as $K = YQ^{-1}$.

To solve this problem using `lmisolver`, we first need to construct the evaluation function.

```
function [LME,LMI,OBJ]=sf_sat_eval(XLIST)
 [Q,Y]=XLIST(:)
 LME=Q-Q'
 LMI=list(-A*Q-Q*A'-B*Y-Y'*B',[umax^2*eye(Y*Y'),Y;Y',Q],Q-eye())
 OBJ=[]
```

Note that `OBJ=[]` indicates that the problem considered is a feasibility problem, i.e., we are only interested in finding a set of X's that satisfy LME and LMI functions.

Assuming `A`, `B` and `umax` already exist in the environment, we can call `lmisolver`, and reconstruct the solution in Scilab, as follows:

```
--> Q_init=zeros(A);
--> Y_init=zeros(B');
--> XLIST0=list(Q_init,Y_init);
--> XLIST=lmisolver(XLIST0,sf_sat_eval);
--> [Q,Y]=XLIST(:)
```

These Scilab commands can of course be encapsulated in a Scilab function, say `sf_sat`. Then, To solve this problem, all we need to do is type:

```
--> [Q,Y]=sf_sat(A,B,umax)
```

We call `sf_sat` the *solver function* for this problem.

Control of jump linear systems

We are given a linear system

$$\dot{x} = A(r(t))x + B(r(t))u,$$

where A is $n \times n$ and B is $n \times n_u$. The scalar parameter $r(t)$ is a continuous-time Markov process taking values in a finite set $\{1, \ldots, N\}$.

The transition probabilities of the process r are defined by a "transition matrix" $\Pi = (\pi_{ij})$, where π_{ij}'s are the transition probability rates from the i-th mode to the j-th. Such systems, referred to as "jump linear systems", can be used to model linear systems subject to failures.

We seek a state-feedback control law such that the resulting closed-loop system is mean-square stable. That is, for every initial condition $x(0)$, the resulting trajectory of the closed-loop system satisfies $\lim_{t\to\infty} \mathbf{E}\|x(t)\|^2 = 0$.

The control law we look for is a mode-dependent linear state-feedback, *i.e.* it has the form $u(t) = K(r(t))x(t)$; $K(i)$'s are $n_u \times n$ matrices (the unknowns of our control problem).

It can be shown that this problem has a solution if and only if there exist $n \times n$ matrices $Q(1), \ldots, Q(N)$, and $n_u \times n$ matrices $Y(1), \ldots, Y(N)$, such that

$$\begin{aligned} Q(i) - Q(i)^T &= 0, \\ \mathbf{Tr}Q(1) + \ldots + \mathbf{Tr}Q(N) - 1 &= 0. \end{aligned}$$

and

$$\begin{aligned} \begin{bmatrix} Q(i) & Y(i)^T \\ Y(i) & I \end{bmatrix} &> 0, \\ -\left[A(i)Q(i) + Q(i)A(i)^T + B(i)Y(i) + Y(i)^T B(i)^T + \sum_{j=1}^{N} \pi_{ji} Q(j)\right] &> 0, \quad i = 1, \ldots, N, \end{aligned}$$

If such matrices exist, a stabilizing state-feedback is given by $K(i) = Y(i)Q(i)^{-1}$, $i = 1, \ldots, N$.

In the above problem, the data matrices are $A(1), \ldots, A(N)$, $B(1), \ldots, B(N)$ and the transition matrix Π. The unknown matrices are $Q(i)$'s (which are symmetric $n \times n$ matrices) and $Y(i)$'s (which are $n_u \times n$ matrices). In this case, both the number of the data matrices and that of the unknown matrices are a-priori unknown.

The above problem is obviously a Σ problem. In this case, we can let `XLIST` be a list of two lists: one representing the Q's and the other, the Y's.

The evaluation function required for invoking `lmisolver` can be constructed as follows:

```
function [LME,LMI,OBJ]=jump_sf_eval(XLIST)
[Q,Y]=XLIST(:)
N=size(A); [n,nu]=size(B(1))
LME=list(); LMI1=list(); LMI2=list()
tr=0
for i=1:N
   tr=tr+trace(Q(i))
   LME(i)=Q(i)-Q(i)'
   LMI1(i)=[Q(i),Y(i)';Y(i),eye(nu,nu)]
   SUM=zeros(n,n)
   for j=1:N
     SUM=SUM+PI(j,i)*Q(j)
   end
   LMI2(i)= A(i)*Q(i)+Q(i)*A(i)'+B(i)*Y(i)+Y(i)'*B(i)'+SUM
end
LMI=list(LMI1,LMI2)
LME(N+1)=tr-1
OBJ=[]
```

Note that `LMI` is also a list of lists containing the values of the LMI matrices. This is just a matter of convenience.

Now, we can solve the problem in Scilab as follows (assuming lists `A` and `B`, and matrix `PI` have already been defined).

First we should initialize `Q` and `Y`.

```
--> N=size(A);  [n,nu]=size(B(1)); Q_init=list(); Y_init=list();
--> for i=1:N, Q_init(i)=zeros(n,n);Y_init(i)=zeros(nu,n);end
```

Then, we can use `lmisolver` as follows:

```
--> XLIST0=list(Q_init,Y_init)
--> XLISTF=lmisolver(XLIST0,jump_sf_eval)
--> [Q,Y]=XLISTF(:);
```

The above commands can be encapsulated in a solver function, say `jump_sf`, in which case we simply need to type:

```
--> [Q,Y]=jump_sf(A,B,PI)
```

to obtain the solution.

Descriptor Lyapunov inequalities

In the study of descriptor systems, it is sometimes necessary to find (or find out that it does not exist) an $n \times n$ matrix X satisfying

$$\begin{aligned} E^T X = X^T E &\geq 0 \\ A^T X + X^T A + I &\leq 0 \end{aligned}$$

where E and A are $n \times n$ matrices such that E, A is a regular pencil. In this problem, which clearly is a Σ problem, the LME functions play important role. The evaluation function can be written as follows

```
function [LME,LMI,OBJ]=dscr_lyap_eval(XLIST)
X=XLIST(:)
LME=E'*X-X'*E
LMI=list(-A'*X-X'*A-eye(),E'*X)
OBJ=[]
```

and the problem can be solved by (assuming E and A are already defined)

```
--> XLIST0=list(zeros(A))
--> XLISTF=lmisolver(XLIST0,dscr_lyap_eval)
--> X=XLISTF(:)
```

Mixed H_2/H_∞ Control

Consider the linear system

$$\begin{aligned} \dot{x} &= Ax + B_1 w + B_2 u \\ z_1 &= C_1 x + D_{11} w + D_{12} u \\ z_2 &= C_2 x + D_{22} u \end{aligned}$$

The mixed H_2/H_∞ control problem consists in finding a stabilizing feedback which yields $\|T_{z_1 w}\|_\infty < \gamma$ and minimizes $\|T_{z_2 w}\|_2$ where $\|T_{z_1 w}\|_\infty$ and $\|T_{z_2 w}\|_2$ denote respectively the closed-loop transfer functions from w to z_1 and z_2. In [22], it is shown that the solution to this problem can be expressed as $K = LX^{-1}$ where X and L are obtained from the problem of minimizing Trace(Y) subject to:

$$X - X^T = 0, \;\; Y - Y^T = 0,$$

and

$$\begin{aligned} -\begin{pmatrix} AX + B_2 L + (AX + B_2 L)^T + B_1 B_1^T & XC_1^T + L^T D_{12}^T + B_1 D_{11}^T \\ C_1 X + D_{12} L + D_{11} B_1^T & -\gamma^2 I + D_{11} D_{11}^T \end{pmatrix} &> 0 \\ \begin{pmatrix} Y & C_2 X + D_{22} L \\ (C_2 X + D_{22} L)^T & X \end{pmatrix} &> 0 \end{aligned}$$

To solve this problem with `lmisolver`, we define the evaluation function:

```
function [LME,LMI,OBJ]=h2hinf_eval(XLIST)
[X,Y,L]=XLIST(:)
LME=list(X-X',Y-Y');
LMI=list(-[A*X+B2*L+(A*X+B2*L)'+B1*B1',X*C1'+L'*D12'+B1*D11';...
              (X*C1'+L'*D12'+B1*D11')',-gamma^2*eye()+D11*D11'],...
                                  [Y,C2*X+D22*L;(C2*X+D22*L)',X])
OBJ=trace(Y);
```

and use it as follows:

```
--> X_init=zeros(A); Y_init=zeros(C2*C2'); L_init=zeros(B2')
--> XLIST0=list(X_init,Y_init,L_init);
--> XLISTF=lmisolver(XLIST0,h2hinf_eval);
--> [X,Y,L]=XLISTF(:)
```

Descriptor Riccati equations

In Kalman filtering for descriptor system

$$\begin{aligned} Ex(k+1) &= Ax(k)+u(k) \\ y(k+1) &= Cx(k+1)+r(k) \end{aligned}$$

where u and r are zero-mean, white Gaussian noise sequences with covariance Q and R respectively, one needs to obtain the positive solution to the descriptor Riccati equation (see [33])

$$P = -\begin{pmatrix} 0 & 0 & I \end{pmatrix} \begin{pmatrix} APA^T+Q & 0 & E \\ 0 & R & C \\ E^T & C^T & 0 \end{pmatrix}^{-1} \begin{pmatrix} 0 \\ 0 \\ I \end{pmatrix}.$$

It can be shown that this problem can be formulated as a Σ problem as follows: maximize Trace(P) under constraints

$$P - P^T = 0$$

and

$$\begin{pmatrix} APA^T+Q & 0 & EP \\ 0 & R & CP \\ P^TE^T & P^TC^T & P \end{pmatrix} \geq 0.$$

The evaluation function is:

```
function [LME,LMI,OBJ]=ric_dscr_eval(XLIST)
LME=P-P'
LMI=[A*P*A'+Q,zeros(A*C'),E*P;zeros(C*A'),R,C*P;P*E',P*C',P]
OBJ=-trace(P)
```

which can be used as follows (asuming E, A, C, Q and R are defined and have compatible sizes–note that E and A need not be square).

```
--> P_init=zeros(A'*A)
--> P=lmisolver(XLIST0,ric_dscr_eval)
```

Linear programming with equality constraints

Consider the following classical optimization problem

$$\begin{array}{ll} \text{minimize} & e^Tx \\ \text{subject to} & Ax+b \geq 0, \\ & Cx+d = 0. \end{array}$$

where A and C are matrices and e, b and d are vectors with appropriate dimensions. Here the sign $\geq$ is to be understood elementwise.

This problem can be formulated in LMITOOL as follows:

```
function [LME,LMI,OBJ]=linprog_eval(XLIST)
 [x]=XLIST(:)
 [m,n]=size(A)
```

```
LME=C*x+d
LMI=list()
tmp=A*x+b
for i=1:m
   LMI(i)=tmp(i)
end
OBJ=e'*x
```

and solved in Scilab by (assuming A, C, e, b and d and an initial guess x0 exist in the environment):

```
--> x=lmisolver(x0,linprog_eval)
```

Sylvester Equation

The problem of finding matrix X satisfying

$$AX + XB = C$$

or

$$AXB = C$$

where A and B are square matrices (of possibly different sizes) is a well-known problem. We refer to the first equation as the continuous Sylvester equation and the second, the discrete Sylvester equation.

These two problems can easily be formulated as Σ problems as follows:

```
function [LME,LMI,OBJ]=sylvester_eval(XLIST)
[X]=XLIST(:)
if flag=='c' then
  LME=A*X+X*B-C
else
  LME=A*X*B-C
end
LMI=[]
OBJ=[]
```

with a solver function such as:

```
function [X]=sylvester(A,B,C,flag)
[na,ma]=size(A);[nb,mb]=size(B);[nc,mc]=size(C);
if ma<>na|mb<>nb|nc<>na|mc<>nb then error("invalid dimensions");end
XLISTF=lmisolver(zeros(nc,mc),sylvester_eval)
X=XLISTF(:)
```

Then, to solve the problem, all we need to do is to (assuming A, B and C are defined)

```
--> X=sylvester(A,B,C,'c')
```

for the continuous problem and

```
--> X=sylvester(A,B,C,'d')
```

for the discrete problem.

7.3 Function LMITOOL

The purpose of LMITOOL is to automate most of the steps required before invoking lmisolver. In particular, it generates a *.sci file including the solver function and the evaluation function or at least their skeleton. The solver function is used to define the initial guess and to modify optimization parameters (if needed).

lmitool can be invoked with zero, one or three arguments.

7.3.1 Non-interactive mode

lmitool can be invoked with three input arguments as follows:

Syntax

```
txt=lmitool(probname,varlist,datalist)
```

where

- probname: a string containing the name of the problem,
- xlist: a string containing the names of the unknown matrices (separated by commas if there are more than one).
- dlist: a string containing the names of data matrices (separated by commas if there are more than one).
- txt: a string providing information on what the user should do next.

In this mode, lmitool generates a file in the current directory. The name of this file is obtained by adding ".sci" to the end of probname. This file is the skeleton of a solver function and the corresponding evaluation function.

Example

Suppose we want to use lmitool to solve the problem presented in Section 7.2.2. Invoking

```
-->txt=lmitool('sf_sat','Q,Y','A,B,umax')
```

yields the output

```
--> txt  =

!    To solve your problem, you need to                          !
!                                                                !
!1- edit file /usr/home/DrScilab/sf_sat.sci                      !
!                                                                !
!2- load (and compile) your functions:                           !
!                                                                !
!   getf('/usr/home/DrScilab/sf_sat.sci','c')                    !
```

```
!                                                                    !
!3- Define A,B,umax and call sf_sat function:                        !
!                                                                    !
!  [Q,Y]=sf_sat(A,B,umax)                                            !
!                                                                    !
!To check the result, use [LME,LMI,OBJ]=sf_sat_eval(list(Q,Y))  !
```

and results in the creation of the file '/usr/home/curdir/sf_sat.sci' with the following content:

```
 function [Q,Y]=sf_sat(A,B,umax)
// Generated by lmitool on Tue Feb 07 10:30:35 MET 1995

  Mbound = 1e3;
  abstol = 1e-10;
  nu = 10;
  maxiters = 100;
  reltol = 1e-10;
  options=[Mbound,abstol,nu,maxiters,reltol];

//////////DEFINE INITIAL GUESS BELOW
Q_init=...
Y_init=...
//////////

XLIST0=list(Q_init,Y_init)
XLIST=lmisolver(XLIST0,sf_sat_eval,options)
[Q,Y]=XLIST(:)



////////////////EVALUATION FUNCTION//////////////////////////

function [LME,LMI,OBJ]=sf_sat_eval(XLIST)
[Q,Y]=XLIST(:)

////////////////DEFINE LME, LMI and OBJ BELOW
LME=...
LMI=...
OBJ=...
```

It is easy to see how a small amount of editing can do the rest!

7.3.2 Interactive mode

`lmitool` can be invoked with zero or one input argument as follows:

Syntax

```
txt=lmitool()
txt=lmitool(file)
```

where

- `file`: is a string giving the name of an existing ".sci" file generated by `lmitool`.

In this mode, `lmitool` is fully interactive. Using a succession of dialogue boxes, user can completely define his problem. This mode is very easy to use and its operation completely self explanatory. Invoking `lmitool` with one argument allows the user to start off with an existing file. This mode is useful for modifying existing files or when the new problem is not too much different from a problem already treated by `lmitool`.

Example

Consider the following estimation problem

$$y = Hx + Vw$$

where x is unknown to be estimated, y is known, w is a unit-variance zero-mean Gaussian vector, and

$$H \in \mathbf{Co}\left\{H(1), ..., H(N)\right\}, \quad V \in \mathbf{Co}\left\{V(1), ..., V(N)\right\}$$

where **Co** denotes the convex hull and $H(i)$ and $V(i)$, $i = 1, ..., N$, are given matrices.

The objective is to find L such that the estimate

$$\hat{x} = Ly$$

is unbiased and the worst case estimation error variance $\mathrm{E}(\|x - \hat{x}\|^2)$ is minimized.

It can be shown that this problem can be formulated as a Σ problem as follows: minimize γ subject to

$$\begin{aligned} I - LH(i) &= 0, \quad i = 1, ..., N, \\ X(i) - X(i)^T &= 0, \quad i = 1, ..., N, \end{aligned}$$

and

$$\begin{pmatrix} I & (L(i)V(i))^T \\ L(i)V(i) & X(i) \end{pmatrix} \geq 0, \quad i = 1, ..., N,$$
$$\gamma - \mathrm{Trace}(X(i)) \geq 0, \quad i = 1, ..., N.$$

To use `lmitool` for this problem, we invoke it as follows:

```
--> lmitool()
```

This results is an interactive session which is partly illustrated in following figures.

7.4 How `lmisolver` works

The function `lmisolver` works essentially in four steps:

1. *Initial set-up.* The sizes and structure of the initial guess are used to set up the problem, and in particular the size of the unknown vector.

2. *Elimination of equality constraints.* Making repeated calls to the evaluation function, `lmisolver` generates a canonical representation of the form

$$\begin{array}{ll} \text{minimize} & \tilde{c}^T z \\ \text{subject to} & \tilde{F}_0 + z_1\tilde{F}_1 + \cdots + z_{\tilde{m}}\tilde{F}_{\tilde{m}} \geq 0, \quad Az + b = 0, \end{array}$$

   where z contains the coefficients of all matrix variables. This step uses extensively sparse matrices to speed up the computation and reduce memory requirement.

3. *Elimination of variables.* Then, `lmisolver` eliminates the redundant variables. The equality constraints are eliminated by computing the null space N of A and a solution z_0 (if any) of $Ax + b = 0$. At this stage, all solutions of the equality constraints are parametrized by

$$z = Nx + z_0,$$

   where x is a vector containing the independent variables. The computation of N, z_0 is done using sparse LU functions of Scilab.

   Once the equality constraints are eliminated, the problem is reformulated as

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & F_0 + x_1F_1 + \cdots + x_mF_m \geq 0, \end{array}$$

   where c is a vector, and $F_0, \ldots, F_m$ are symmetric matrices, and x contains the *independent* elements in the matrix variables $X_1, \ldots, X_M$. (If the F_i's are dependent, a column compression is performed.)

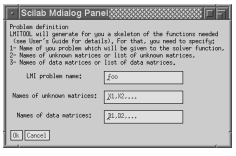

Figure 7.1: This window must be edited to define problem name and the name of variables used.

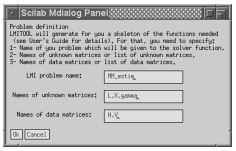

Figure 7.2: For the example at hand the result of the editing should look something like this.

4. *Optimization.* Finally, `lmisolver` makes a call to the function `semidef` (an interface to **SP** [23]). This phase is itself divided into a feasibility phase and a minimization phase (only if the linear objective function is not empty). The feasibility phase is avoided if the initial guess is found to be feasible.

   The function `semidef` is called with the optimization parameters `abstol`, `nu`, `maxiters`, `reltol`. The parameter `M` is set above the value

   ```
   Mbnd*max(sum(abs([F0 ... Fm])))
   ```

   For details about the optimization phase, and the meaning of the above optimization parameters see manual page for `semidef`.

7.5 Other versions

`LMITOOL` is also available on Matlab. The Matlab version can be obtained by anonymous ftp from `ftp.ensta.fr` under `/pub/elghaoui/lmitool`.

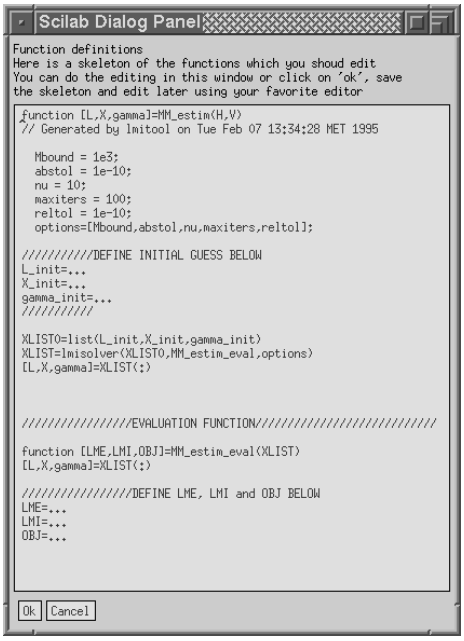

Figure 7.3: This is the skeleton of the solver function and the evaluation function generated by `LMITOOL` using the names defined previously.

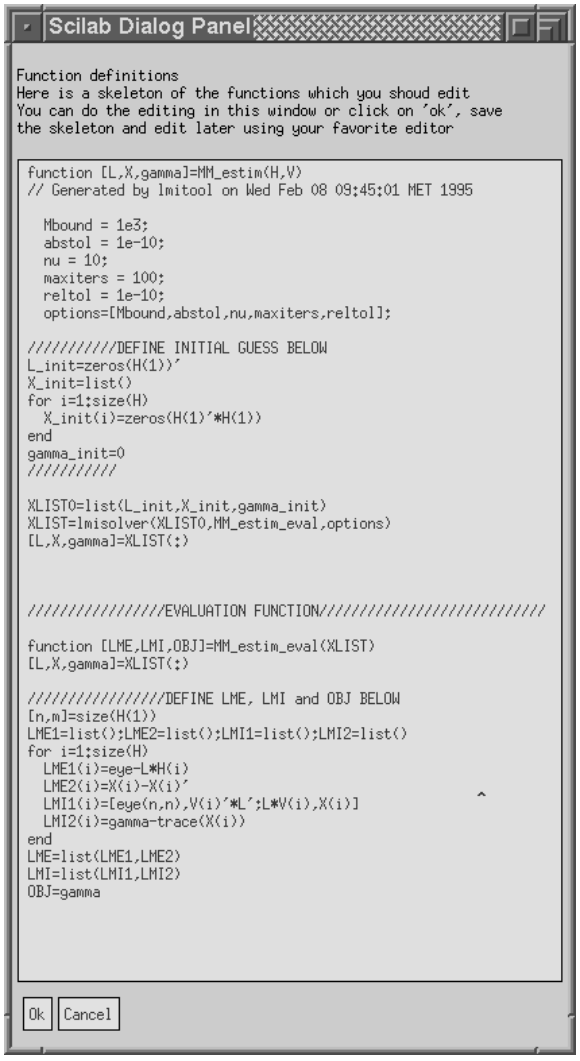

Figure 7.4: After editing, we obtain.

Figure 7.5: A file is proposed in which the solver and evaluation functions are to be saved. You can modify it if you want.

Chapter 8

Optimization data files

This section presents the optimization data files which can be used to configure a specific optimization problem in Scilab. The following is a (non-exhaustive) list of ASCII file formats often used in optimization softwares :

- SIF : Standard Input Format [1, 30],
- GAMS : General Algebraic Modeling System [40, 16]
- AMPL : A Mathematical Programming Language [10, 39]
- MPS : Mathematical Programming System [27, 41]

but other file formats appeared in recent years, such as the XML-based file format OSiL [35, 8, 36].

The following sections describe Scilab tools to manage optimization data files.

8.1 MPS files and the Quapro toolbox

The Quapro toolbox implements the `readmps` function, which reads a file containing description of an LP problem given in MPS format and returns a tlist describing the optimization problem. It is an interface with the program rdmps1.f of hopdm (J. Gondzio). For a description of the variables, see the file rdmps1.f. MPS format is a standard ASCII medium for LP codes. MPS format is described in more detail in Murtagh's book [30].

8.2 SIF files and the CUTEr toolbox

The SIF file format can be processed with the CUTEr Scilab toolbox. Given a SIF [1] file the function `sifdecode` generates associated Fortran routines RANGE.f, EXTER.f, ELFUN.f, GROUP.f and if automatic differentiation is required ELFUND.f, GROUPD.f, EXTERA.f. An associated data file named OUTSDIF.d and an Output messages file OUTMESS are also generated. All these files are created in the directory whose path is given in Pathout. The `sifdecode` function is based on the Sifdec code [20]. More precisely it results of an interface of SDLANC Fortran procedure.

Chapter 9

Scilab Optimization Toolboxes

Some Scilab toolboxes are designed to solve optimization problems. In this chapter, we begin by presenting the Quapro toolbox, which allows to solve linear and quadratic problems. Then we outline other main optimization toolboxes.

9.1 Quapro

The Quapro toolbox was formely a Scilab built-in optimization tool. It has been transformed into a toolbox for license reasons.

9.1.1 Linear optimization

Mathematical point of view

This kind of optimization is the minimization of function $f(x)$ with

$$f(x) = p^T x$$

under:

- no constraints
- inequality constraints (9.1)
- *or* inequality constraints and bound constraints ((9.1) & (9.2))
- *or* inequality constraints, bound constraints and equality constraints ((9.1) & (9.2) & (9.3)).

$$C * x \leq b \tag{9.1}$$

$$ci \leq x \leq cs \tag{9.2}$$

$$C_e * x = b_e \tag{9.3}$$

Scilab function

Scilab function called *linpro* is designed for linear optimization programming. For more details about this function, please refer to Scilab online help This function and associated routines have been written by Cecilia Pola Mendez and Eduardo Casas Renteria from the University of Cantabria. Please note that this function can not solve problems based on sparse matrices. For this kind of problem, you can use a Scilab toolbox called LIPSOL that gives an equivalent of *linpro* for sparse matrices. LIPSOL is available on Scilab web site

Optimization routines

Scilab *linpro* function is based on:

- some Fortran routines written by the authors of *linpro*
- some Fortran BLAS routines
- some Fortran Scilab routines
- some Fortran LAPACK routines

9.1.2 Linear quadratic optimization

Mathematical point of view

This kind of optimization is the minimization of function $f(x)$ with

$$f(x) = \frac{1}{2}x^T Q x + p^T x$$

under:

- no constraints
- inequality constraints (9.1)
- *or* inequality constraints and bound constraints ((9.1) & (9.2))
- *or* inequality constraints, bound constraints and equality constraints ((9.1) & (9.2) & (9.3)).

Scilab function

Scilab functions called *quapro* (whatever Q is) and *qld* (when Q is positive definite) are designed for linear optimization programming. For more details about these functions, please refer to Scilab online help for *quapro* and Scilab online help for *qld* *qld* function and associated routine have been written by K. Schittkowski from the University of Bayreuth, A.L. Tits and J.L. Zhou from the University of Maryland. *quapro* function and associated routines have been written by Cecilia Pola Mendez and Eduardo Casas Renteria from the University of Cantabria. Both functions can not solve problems based on sparse matrices.

Optimization routines

Scilab *quapro* function is based on:

- some Fortran routines written by the authors of *linpro*
- some Fortran BLAS routines
- some Fortran Scilab routines
- some Fortran LAPACK routines

9.2 CUTEr

CUTEr is a versatile testing environment for optimization and linear algebra solvers [31]. This toolbox is a scilab port by Serge Steer and Bruno Durand of the original Matlab toolbox.

A typical use start from problem selection using the scilab function `sifselect`. This gives a vector of problem names corresponding to selection criteria [32]. The available problems are located in the sif directory.

The `sifbuild` function can then be used to generate the fortran codes associated to a given problem, to compile them and dynamically link it to Scilab. This will create a set of problem relative functions, for example, ufn or ugr. This functions can be called to compute the objective function or its gradient at a given point.

The `sifoptim` function automatically applies the optim function to a selected problem.

A Fortran compiler is mandatory to build problems.

This toolbox contains the following parts.

- Problem database

  A set of testing problems coded in "Standard Input Format" (SIF) is included in the sif/ sub-directory. This set comes from www.numerical.rl.ac.uk/cute/mastsif.html. The Scilab function `sifselect` can be used to select some of this problems according to objective function properties, contraints properties and regularity properties

- SIF format decoder

  The Scilab function `sifdecode` can be used to generate the Fortran codes associated to a given problem, while the Scilab function `buildprob` compiles and dynamically links these fortran code with Scilab

- problem relative functions

  The execution of the function buildprob adds a set of functions to Scilab. The first one is `usetup` for unconstrained or bounded problems or `csetup` for problems with general contraints. These functions are to be called before any of the following to initialize the problem relative data (only one problem can be run at a time). The other functions allow to compute the objective, the gradient, the hessian values, ... of the problem at a given point (see `ufn`, `ugr`, `udh`, ... for unconstrained or bounded problems or `cfn`, `cgr`, `cdh`, ... for problems with general contraints)

- CUTEr and `optim` The Scilab function `optim` can be used together with CUTEr using either the external function `ucost` or the driver function `sifoptim`.

The following is a list of references for the CUTEr toolbox :

- CUTEr toolbox on Scilab Toolbox center
- CUTEr website

9.3 The Unconstrained Optimization Problem Toolbox

The Unconstrained Optimization Problem Toolbox provides 35 unconstrained optimization problems.

The goal of this toolbox is to provide unconstrained optimization problems in order to test optimization algorithms.

The More, Garbow and Hillstrom collection of test functions [29] is widely used in testing unconstrained optimization software. The code for these problems is available in Fortran from the netlib software archives.

It provides the function value, the gradient, the function vector, the Jacobian and provides the Hessian matrix for 18 problems. It provides the starting point for each problem, the optimum function value and the optimum point x for many problems. Additionnally, it provides finite difference routines for the gradient, the Jacobian and the Hessian matrix. The functions are based on macros based functions : no compiler is required, which is an advantage over the CUTEr toolbox. Finally, all function values, gradients, Jacobians and Hessians are tested.

This toolbox is available in ATOMS :

`http://atoms.scilab.org/toolboxes/uncprb`

and is manage under Scilab's Forge :

`http://forge.scilab.org/index.php/p/uncprb`

To install it, type the following statement in Scilab v5.2 (or better).

```
1 atomsInstall('uncprb')
```

9.4 Other toolboxes

- Interface to **CONMIN**: An interface to the NASTRAN / NASA CONMIN optimization program by Yann Collette. CONMIN can solve a nonlinear objective problem with nonlinear constraints. CONMIN uses a two-step limited memory quasi-Newton-like Conjugate Gradient. The CONMIN optimization method is currently used in NASTRAN (a professionnal finite element tool) and the optimization part of NASTRAN (the CONMIN tool). The CONMIN fortran program has been written by G. Vanderplaats (1973).
  - CONMIN on Scilab Toolbox center
- **Differential Evolution**: random search of a global minimum by Helmut Jarausch. This toolbox is based on a Rainer-Storn algorithm.

  - Differential Evolution on Scilab Toolbox center

- **FSQP Interface**: interface for the Feasible Sequential Quadratic Programming library. This toolbox is designed for non-linear optimization with equality and inequality constraints. FSQP is a commercial product.
  - FSQP on Scilab Toolbox center
  - FSQP website

- **IPOPT interface**: interface for IPOPT, which is based on an interior point method which can handle equality and inequality nonlinear constraints. This solver can handle large scale optimization problems. As open source software, the source code for Ipopt is provided without charge. You are free to use it, also for commercial purposes. This Scilab-Ipopt interface was based on the Matlab Mex Interface developed by Claas Michalik and Steinar Hauan. This version only works on linux, scons and Scilab >=4.0. Tested with gcc 4.0.3. Modifications to Scilab Interface made by Edson Cordeiro do Valle.
  - Ipopt on Scilab Toolbox center
  - Ipopt website

- Interface to **LIPSOL**: sparse linear problems with interior points method by H. Rubio Scola. LIPSOL can minimize a linear objective with linear constraints and bound constraints. It is based on a primal-dual interior point method, which uses sparse-matrix data-structure to solve large, sparse, symmetric positive definite linear systems. LIPSOL is written by Yin Zhang . The original Matlab-based code has been adapted to Scilab by H. Rubio Scola (University of Rosario, Argentina). It is distributed freely under the terms of the GPL. LIPSOL also uses the ORNL sparse Cholesky solver version 0.3 written by Esmond Ng and Barry Peyton by H. Rubio Scola.
  - LIPSOL on Scilab Toolbox center
  - LIPSOL website
  - LIPSOL User's Guide

- **LPSOLVE**: an interface to lp_solve. lp_solve is a free mixed integer/binary linear programming solver with full source, examples and manuals. lp_solve is under LGPL, the GNU lesser general public license. lp_solve uses the 'Simplex' algorithm and sparse matrix methods for pure LP problems.
  - LPSOLVE toolbox on Scilab Toolbox center
  - lp_solve solver on Sourceforge
  - lp_solve on Geocities
  - lp_solve Yahoo Group

- **NEWUOA**: NEWUOA is a software developped by M.J.D. Powell for unconstrained optimization without derivatives. The NEWUOA seeks the least value of a function F(x) (x is a vector of dimension n) when F(x) can be calculated for any vector of variables x

. The algorithm is iterative, a quadratic model being required at the beginning of each iteration, which is used in a trust region procedure for adjusting the variables. When the quadratic model is revised, the new model interpolates F at m points, the value m=2n+1 being recommended.

- NEWUOA toolbox on Scilab Toolbox center
- NEWUOA at INRIA Alpes

Chapter 10

Missing optimization features in Scilab

Several optimization features are missing in Scilab. Two classes of missing features are to analyse :

- features which are not available in Scilab, but which are available as toolboxes (see previous section),
- features which are not available neither in Scilab, nor in toolboxes.

Here is a list of features which are not available in Scilab, but are available in toolboxes. These features would be to include in Scilab.

- integer parameter with linear objective solver and sparse matrices : currently available in LPSOLVE toolbox, based on the simplex method,
- linear objective with sparse matrices : currently available in LIPSOL, based on interior points method,
- nonlinear objective and non linear constraints : currently available in interface to IPOPT toolbox, based on interior point methods,
- nonlinear objective and non linear constraints : currently available in interface to CONMIN toolbox, based on method of feasible directions,

Notice that IPOPT is a commercial product and CONMIN is a domain-public library. Therefore the only open-source, free, nonlinear solver with non linear constraints tool available with Scilab is the interface to CONMIN.

Here is a list of features which are not available neither in Scilab, nor in toolboxes.

- quadratic objective solver with sparse objective matrix,
- simplex programming method (*),
- non-linear objective with nonlinear constraints (*),
- non-linear objective with nonlinear constraints problems based on sparse linear algebra,
- enabling/disabling of unknowns or constraints,
- customization of errors for constraints.

Functionalities marked with a (*) would be available in Scilab if the MODULOPT library embedded in Scilab was updated.

Conclusion

Even if Scilab itself has lacks in optimization functionalities, all embedded functions are very useful to begin with. After that, by downloading and installing some toolboxes, you can easily improve your Scilab capabilities.

One of the questions we can ask is: "Why are these toolboxes not integrated in Scilab distribution?". The answer is often a problem of license. All GPL libraries can not be included in Scilab since Scilab is not designed to become GPL.

Bibliography

[1] Nicholas I. M. Gould Andrew R. Conn and Philippe L. Toint. The sif reference document. http://www.numerical.rl.ac.uk/lancelot/sif/sifhtml.html.

[2] Michael Baudin. Nelder mead user's manual. http://wiki.scilab.org/The_Nelder-Mead_Component, 2009.

[3] Michael Baudin and Serge Steer. Optimization with scilab, present and future. To appear in Proceedings Of 2009 International Workshop On Open-Source Software For Scientific Computation (Ossc-2009).

[4] Frédéric Bonnans. A variant of a projected variable metric method for bound constrained optimization problems. Technical Report RR-0242, INRIA - Rocquencourt, Octobre 1983.

[5] Joseph Frédéric Bonnans, Jean-Charles Gilbert, Claude Lemaréchal, and Claudia A. Sagastizábal. *Numerical Optimization. Theoretical and Practical Aspects*. Universitext. Springer Verlag, November 2006. Nouvelle édition, revue et augmentéee. 490 pages.

[6] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*, volume 15 of *Studies in Applied Mathematics*. SIAM, Philadelphia, PA, June 1994.

[7] Carlos A. Coello Coello. List of references on evolutionary multiobjective optimization. http://www.lania.mx/~ccoello/EMOObib.html.

[8] coin or.org. Optimization services instance language (osil). https://www.coin-or.org/OS/OSiL.html.

[9] Yann Collette. Personnal website. http://ycollette.free.fr.

[10] AMPL company. A modeling language for mathematical programming. http://en.wikipedia.org/wiki/AMPL_programming_language.

[11] Goldfarb D. and Idnani A. Dual and primal-dual methods for solving strictly convex quadratic programs. *Lecture Notes in Mathematics, Springer-Verlag*, 909:226–239, 1982.

[12] Goldfarb D. and Idnani A. A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical Programming*, 27:1–33, 1982.

[13] Lawrence Davis. *Genetic Algorithms and Simulated Annealing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[14] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. pages 849–858. Springer, 2000. http://www.lania.mx/%7Eccoello/deb00.ps.gz.

[15] Carlos M. Fonseca and Peter J. Fleming. Genetic algorithms for multiobjective optimization: Formulationdiscussion and generalization. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 416–423, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc. http://www.lania.mx/%7Eccoello/fonseca93.ps.gz.

[16] gams.com. The general algebraic modeling system. http://www.gams.com/.

[17] David E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley, 1989.

[18] Jean-Baptiste Hiriart-Urruty and Claude Lemaréchal. *Convex Analysis and Minimization Algorithms I: Fundamentals*. Springer, October 1993.

[19] Jean-Baptiste Hiriart-Urruty and Claude Lemaréchal. *Convex Analysis and Minimization Algorithms II: Advanced Theory and Bundle Methods*. Springer, October 1993.

[20] hsl.rl.ac.uk. A lonesome sif decoder. http://hsl.rl.ac.uk/cuter-www/sifdec/doc.html.

[21] Stephen J. Wright Jorge Nocedal. *Numerical Optimization*. Springer, 1999.

[22] P.P. Khargonekar and M.A. Rotea. Mixed h2/h? control: a convex optimization approach. *Automatic Control, IEEE Transactions on*, 36(7):824–837, Jul 1991.

[23] Vandenberghe L. and S. Boyd. Semidefinite programming. Internal Report, Stanford University, 1994 (submitted to SIAM Review).

[24] P. J. M. Laarhoven and E. H. L. Aarts, editors. *Simulated annealing: theory and applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1987.

[25] P. J. M. van Laarhoven. *Theoretical and Computational Aspects of Simulated Annealing*. Amsterdam, Netherlands : Centrum voor Wiskunde en Informatica, 1988.

[26] Claude Lemaréchal. A view of line-searches. *Lectures Notes in Control and Information Sciences*, 30:59–78, 1981.

[27] lpsolve. Mps file format. http://lpsolve.sourceforge.net/5.5/mps-format.htm.

[28] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1996.

[29] J. J. Moré, Burton S. Garbow, and Kenneth E. Hillstrom. Algorithm 566: Fortran subroutines for testing unconstrained optimization software [c5], [e4]. *ACM Trans. Math. Softw.*, 7(1):136–140, 1981.

[30] B. Murtagh. *Advanced Linear Programming: Computation and Practice*. McGraw-Hill, 1981.

[31] Philippe L. Toint Nicholas I.M. Gould, Dominique Orban. A constrained and unconstrained testing environment, revisited. http://hsl.rl.ac.uk/cuter-www/.

[32] Philippe L. Toint Nicholas I.M. Gould, Dominique Orban. The cuter test problem set. http://www.numerical.rl.ac.uk/cute/mastsif.html.

[33] R. Nikoukhah, A.S. Willsky, and B.C. Levy. Kalman filtering and riccati equations for descriptor systems. *Automatic Control, IEEE Transactions on*, 37(9):1325–1342, Sep 1992.

[34] Jorge Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782, 1980.

[35] optimizationservices.org. Optimization services. http://www.optimizationservices.org/.

[36] Jun Ma Robert Fourer and Kipp Martin. Osil: An instance language for optimization. *Computational Optimization and Applications*, 2008.

[37] N. Srinivas and Kalyanmoy Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2:221–248, 1994. http://www.lania.mx/%7Eccoello/deb95.ps.gz.

[38] Berwin A Turlach. Quadprog, (quadratic programming routines). http://www.maths.uwa.edu.au/~berwin/software/quadprog.html.

[39] Wikipedia. Ampl (programming language). http://en.wikipedia.org/wiki/AMPL_programming_language.

[40] Wikipedia. General algebraic modeling system. http://en.wikipedia.org/wiki/General_Algebraic_Modeling_System.

[41] Wikipedia. Mps (format). http://en.wikipedia.org/wiki/MPS_(format).

[42] Wikipedia. Simulated annealing. http://en.wikipedia.org/wiki/Simulated_annealing.

[43] Patrick Siarry Yann Collette. *Optimisation Multiobjectif*. Eyrolles, Collection Algorithmes, 1999.