Partial Spectral Projected Gradient Method with Active-Set Strategy for Linearly Constrained Optimization [*]

Marina Andretta [†] Ernesto G. Birgin [†] J. M. Martínez [‡]

March 9, 2009

Abstract

A method for linearly constrained optimization which modifies and generalizes recent box-constraint optimization algorithms is introduced. The new algorithm is based on a relaxed form of Spectral Projected Gradient iterations. Intercalated with these projected steps, internal iterations restricted to faces of the polytope are performed, which enhance the efficiency of the algorithms. Convergence proofs are given and numerical experiments are included and commented. Software supporting this paper is available through the TANGO Project web page: *http://www.ime.usp.br/∼egbirgin/tango/*.

Key words: Linearly constrained optimization, spectral projected gradient method, active set methods.

1 Introduction

The seminal 1988 paper of Barzilai and Borwein [8] on alternative steplengths for the steepest descent method gave rise to a lot of computational optimization research.

The Spectral Projected Gradient (SPG) method was introduced, analyzed and implemented in [17, 18, 19]. This method combines the basic spectral-step ideas [8, 53, 54] with projected gradient strategies [10, 37, 45]. The extension of the spectral gradient method to smooth convex programming problems [5, 15, 14, 17, 18, 19] was motivated by the surprisingly good performance of the spectral gradient for large-scale unconstrained minimization [54]. Nonmonotone strategies, like the one proposed by Grippo, Lampariello and Lucidi [40], turned out to be an important ingredient for the success of the spectral idea for unconstrained minimization and other extensions.

The SPG method is applicable to convex constrained problems in which the projection onto the feasible set is easy to compute. It has been intensively used in many different applied problems [9, 11, 13, 22, 29, 43, 44, 52, 58, 60, 65]. Several interesting parameter modifications were proposed in the papers [24, 23, 26, 27, 33, 41, 55, 64]. Alternative nonmonotone strategies

[*]This work was supported by PRONEX-Optimization (PRONEX - CNPq / FAPERJ E-26 / 171.510/2006 - APQ1), FAPESP (Grant 2006/53768-0) and CNPq.

[†]Department of Computer Science, Institute of Mathematics and Statistics, University of São Paulo, Rua do Matão 1010 Cidade Universitária, 05508-090 São Paulo SP, Brazil. ({andretta|egbirgin}@ime.usp.br)

[‡]Department of Applied Mathematics, IMECC-UNICAMP, CP 6065, 13081-970 Campinas SP, Brazil. (martinez@ime.unicamp.br)

have been suggested in [24, 27, 28, 59, 66], whereas convergence and stability properties were studied in [25, 32, 62, 63]. In addition, SPG has been combined with other algorithms for particular optimization problems. Bound constrained minimization was addressed in [6, 14, 15]. Linearly constrained optimization and nonlinear systems were considered in [5, 48] and [42, 67], respectively. In [48] a penalty approach was used and in [5] problems were solved by means of sequential quadratic programming. For general nonlinear programming, a method combining SPG with the augmented Lagrangian was proposed in [30], and augmented Lagrangian methods that use the methods introduced in [6, 15] to solve bound constrained subproblems were developed in [3, 4, 12]. More recently, Gomes-Ruggiero, Martínez and Santos [38] showed that the Inexact Restoration framework [39, 46, 47] motivates a natural generalization of SPG to general constrained optimization.

The main drawback for the application of SPG to arbitrary convex (in particular, linearly constrained) problems is the expensiveness of computing projections in general cases. In the present research, in order to apply SPG ideas to general linearly constrained optimization, we used two different strategies:

1. Enlarging the set onto which projections must be computed.
2. Combining the SPG-like iterations with easy-to-compute "essentially unconstrained" iterations which provide practical efficiency to the method.

The enlargement of the projection set involves elimination of constraints. So, by means of this procedure, projections become less expensive. Radical elimination would involve preserving only active constraints in the projection set, but this elimination would be inefficient and even convergence proofs would be impossible. A careful management of the constraints that must be preserved at each state of the calculation gives rise to the Partial Spectral Projected Gradient (PSPG) strategy introduced in the present paper.

The strategy of combining SPG-like iterations with iterations restricted to proper faces was already used in [6, 14, 15] in connection to box-constrained optimization. The main ideas of the present research are inherited from the box-constrained ideas of [6, 15] but the consideration of general linear constraints, instead of merely bounds, imposes different algorithmic decisions. In the case of boxes, projections are trivially computed, therefore it is inexpensive to test, at every iteration, if the face where the current approximation lies must be preserved or not, using the comparison of internal and external components of the constrained steepest descent direction [15]. In the case of linear constraints we prefer to preserve the same active set until the boundary is reached or an approximate active-face constrained stationary point occurs. In this way we reduce, as much as possible, projection steps. The idea of combining PSPG steps with internal iterations can also be understood from the point of view of active-set methods. In Rosen's gradient projection method [56] (one of the first algorithms to which the active-set paradigm applies), and many variations reported in text books, the procedure for giving up active constraints involves removing one constraint per iteration (as in the Simplex method) and, usually, no more than one constraint is added to the working set when the current point hits the boundary. Removing many constraints per iteration is guaranteed in the algorithm presented in the current paper by the PSPG procedure. The possibility of adding constraints very slowly (possibly because of primal degeneracy) is monitored and also corrected by means of PSPG.

This paper is organized as follows. In Section 2 we define the problem, faces and projections that will be used in the algorithms. In Section 3 we present the main algorithm, the PSPG iteration and the Internal Algorithm. The assumptions that must be fulfilled by the internal procedures and a practical scheme to define an Internal Algorithm starting from an unconstrained method are also described. Convergence of the main algorithm is proved in Section 4. In Section 5 we discuss the practical implementation of the algorithms. Numerical experiments are reported in Section 6. In the last section we state some conclusions.

Notation

We denote $I\!N = \{0, 1, 2, \ldots\}$.

If $v, w \in I\!R^n$, we denote

$$[v, w] = \{u \in I\!R^n \mid u = tv + (1-t)w \text{ for some } t \in [0, 1]\}$$

and

$$v_+ = (\max\{v_1, 0\}, \ldots, \max\{v_n, 0\})^T.$$

If $v, w \in I\!R^n$, the statement $v \le w$ means that $v_i \le w_i$ for all $i = 1, \ldots, n$. We denote $P_A(v)$ the Euclidian projection of v onto the set A. The symbol $\|\cdot\|$ denotes the Euclidian norm, although many times may be replaced by an arbitrary norm. The closure of the set A will be denoted, as usually, by $\bar{A}$.

2 Statement of the problem

Let $f : I\!R^n \to I\!R$ be continuously differentiable. Let $\Omega \subset I\!R^n$ be defined by:

$$\Omega = \{x \in I\!R^n \mid C_i(x) \le 0, i = 1, \ldots, p\}, \tag{1}$$

where, for all $i = 1, \ldots, p$,

$$C_i(x) = (a^i)^T x - b_i$$

and

$$a^i = (a^i_1, ..., a^i_n)^T.$$

The problem addressed here is

$$\text{Minimize } f(x) \text{ subject to } C_i(x) \le 0, i = 1, \ldots, p. \tag{2}$$

Given $I \subset \{1, \ldots p\}$ we denote

$$F_I = \{x \in \Omega \mid C_i(x) = 0 \text{ if and only if } i \in I\}.$$

Clearly, Ω is the union of the sets F_I, for $I \subset \{1, \ldots, p\}$. Moreover, $F_I \ne F_J$ implies that $F_I \cap F_J = \emptyset$. The smaller affine subspace that contains a nonempty face F_I will be denoted by V_I and the parallel linear subspace to V_I will be S_I.

Given $z \in \Omega$ and $\delta \in [0, \infty]^p$, we define

$$\Omega(z, \delta) = \{x \in I\!R^n \mid C_i(x) \le 0 \text{ for all } i \in I(z, \delta)\},$$

where

$$I(z, \delta) = \{i \mid C_i(z) \geq -\delta_i\}.$$

Clearly, $\Omega \subset \Omega(z, \delta)$ and $\Omega(z, \infty) = \Omega$. Observe that, if $z \in \Omega$ satisfies the KKT conditions of (2) then it also satisfies the KKT conditions of

$$\text{Minimize } f(x) \text{ subject to } x \in \Omega(z, \delta) \tag{3}$$

for all $\delta \geq 0$. Reciprocally, if z satisfies the KKT conditions of (3) for some $\delta \geq 0$, then z fulfills the KKT conditions of (2). This results from the fact that $z \in \Omega \subset \Omega(z, \delta)$ since the active constraints of $\Omega(z, \delta)$ at z are the same as the active constraints of Ω at z.

Given $z \in \Omega$, we define

$$g_P(z) = P_\Omega(z - \nabla f(z)) - z.$$

We also define

$$g_P(z, \delta) = P_{\Omega(z,\delta)}(z - \nabla f(z)) - z$$

and, for all $\sigma > 0$,

$$g_P(z, \delta, \sigma) = P_{\Omega(z,\delta)}(z - \sigma \nabla f(z)) - z.$$

The following identities hold trivially:

$$g_P(z) = g_P(z, \infty) = g_P(z, \infty, 1)$$

and

$$g_P(z, \delta) = g_P(z, \delta, 1).$$

Note that, given $\delta \in [0, \infty]^p, \sigma > 0$, the fulfillment of the KKT conditions at $z \in \Omega$ is equivalent to $g_P(z, \delta, \sigma) = 0$. Moreover, if $\delta_i^k \geq \delta_{\min} > 0$ for all $i = 1, \ldots, p$, $k \in I\!N$, $z^k \to z$ and z is a KKT point, we have that $\|g_P(z^k, \delta^k)\| \to 0$. This justifies the use of $\|g_P(z^k, \delta^k)\|$ as stopping criterion for numerical algorithms.

Finally, if $z \in F_I \subset \Omega$, we define

$$g_S(z) = P_{S_I}(-\nabla f(z)).$$

3 Main Algorithm

Algorithm 3.1 describes the main method presented in this paper to solve linearly constrained optimization problems. All the iterates x^k will be feasible if $k \geq 1$. If x^0 is infeasible, we project this point onto the feasible region for obtaining x^1. If this projection does not exist, the original problem is infeasible.

At every iteration, x^k belongs to a face F_I. If the norm of the inner gradient $g_S(x^k)$ is larger than a tolerance ε, the algorithm judges that it is still worthwhile to stay in the same face and, so, an internal iteration is performed using, essentially, an unconstrained algorithm. In this process, the new iterate may hit the boundary of F_I. If this happens 20 consecutive times, the algorithm imposes that the next iteration must be of PSPG type. In this way we aim to incorporate as many new constraits as possible.

A PSPG iteration is also performed when $\|g_S(x^k)\|_\infty \leq \varepsilon$. By means of PSPG steps we aim to add or remove many constraints at a single iteration. The tolerance vector δ^k for PSPG is chosen adaptively with the only requirement that all the components must be not smaller than a fixed parameter $\delta_{\min} > 0$. PSPG iterations are more expensive the bigger is δ^k but, on the other hand, very small values of δ^k provide poor information on the geometry of the feasible set.

Algorithm 3.1. (Main Algorithm)
Assume that $x^0 \in I\!R^n$, $0 < \sigma_{\min} < \sigma_{\max} < \infty$, $\alpha \in (0, 1/2]$, $\delta_{\min} > 0$, $\varepsilon > 0$.

Step 1. Compute, if possible, $x^1 = P_\Omega(x^0)$. If this projection does not exist, stop declaring that the original problem is infeasible. Else, set $k \leftarrow 1$, $k_1 \leftarrow 0$.

Step 2. If $k_1 = 20$, perform Step 6.

Step 3. Let $I \subset \{1, \ldots, p\}$ be such that $x^k \in F_I$. If $\|g_S(x^k)\|_\infty > \varepsilon$, perform Steps 4 and 5 , else perform Step 6.

Step 4. Compute $x^{k+1} \in \bar{F}_I$ using the Internal Algorithm, whose main characteristics are given in Assumption A1 below.

Step 5. Let J be such that $x^{k+1} \in F_J$. If I is strictly contained in J, set $k_1 \leftarrow k_1 + 1$, else set $k_1 \leftarrow 0$. Set $k \leftarrow k + 1$ and go to Step 2.

Step 6. Choose $\delta^k \in I\!R^p$ such that $\delta_i^k \geq \delta_{\min}$ for all $i = 1, \ldots, p$. If $\|g_P(x^k, \delta^k)\|_\infty \leq \varepsilon$, stop. Else, compute x^{k+1} using a Partial SPG (PSPG) iteration (Algorithm 3.2 below), set $k \leftarrow k + 1, k_1 \leftarrow 0$ and go to Step 3.

The PSPG iteration used at Step 6 of Algorithm 3.1 is described below.

Algorithm 3.2 (Partial SPG Iteration)

Step 1. If $k = 0$ or $(x^k - x^{k-1})^T(\nabla f(x^k) - \nabla f(x^{k-1})) \leq 0$, set $\sigma_k = 1$. Else, define σ_k as the safeguarded spectral coefficient:

$$\sigma_k = \max\left\{\sigma_{\min}, \min\left\{\sigma_{\max}, \frac{\|x^k - x^{k-1}\|^2}{(x^k - x^{k-1})^T(\nabla f(x^k) - \nabla f(x^{k-1}))}\right\}\right\}.$$

Step 2. Define $d^k = g_P(x^k, \delta^k, \sigma_k)$.

Step 3. Compute

$$t_{\text{break}} = \max\{t \in [0, 1] \mid [x^k, x^k + td^k] \subset \Omega\}.$$

Set $t \leftarrow t_{\text{break}}$.

Step 4. Test the Armijo condition

$$f(x^k + td^k) \leq f(x^k) + \alpha t (d^k)^T \nabla f(x^k). \tag{4}$$

Step 5. If (4) holds, set $t_k = t$, define $x^{k+1} = x^k + t_k d$ and return. Else, choose

$$t_{\text{new}} \in [0.1t, 0.9t], \tag{5}$$

set $t \leftarrow t_{\text{new}}$ and go to Step 4.

The internal steps used at Step 4 of Algorithm 3.1 admit many possible implementations. Essentially, they correspond to the iterations of some convergent algorithm for unconstrained minimization with some modification that corresponds to the case in which the algorithm generates nonfeasible points. Before giving more detailed descriptions we will state here the essential assumption that the Internal Algorithm must satisfy.

Assumption A1

- *If x^{k+1} is computed by the Internal Algorithm and $x^k \in F_I$, then $x^{k+1} \in \bar{F}_I$ and $f(x^{k+1}) \leq f(x^k)$.*
- *If $x^{k+j} \in F_I$ is computed by the Internal Algorithm for all $j \geq 1$, there exists j such that $\|g_S(x^{k+j})\|_\infty \leq \varepsilon$.*

Taking a basis of the subspace S_I and using the coordinates that corresponds to this basis, the auxiliary problem that consists of minimizing f subject to $\bar{F}_I$ may be transformed into a minimization problem on a closed convex set A. Except in the trivial case in which F_I is a single point, A is open and corresponds to the transformation of F_I. With some abuse of notation, let us express the new equivalent auxiliary problem as:

$$\text{Minimize } f(x) \text{ subject to } x \in \bar{A}.$$

Assume that Algorithm U is a monotone method for unconstrained optimization with the property that every limit point of a sequence generated by this method is stationary. Then, a typical iteration of a general internal algorithm that satisfies Assumption A1 may be described as follows.

Algorithm 3.3. (Internal Algorithm Iteration)

Step 1. Compute x^{k+1} using Algorithm U. If $x^{k+1} \in A$, return. Else, define $d^k = x^{k+1} - x^k$ and continue.

Step 2. Compute
$$t_{\text{break}} = \max\{t > 0 \mid [x^k, x^k + td^k] \subset \bar{A}\}. \tag{6}$$
If $f(x^k + t_{\text{break}} d^k) < f(x^k)$, re-define $x^{k+1} = x^k + t_{\text{break}} d^k$ and return.

Step 3. If
$$(d^k)^T \nabla f(x^k) \geq -10^{-6} \|d^k\| \|\nabla f(x^k)\|$$
re-define $d^k \leftarrow -\nabla f(x^k)$ and, consequently, re-define t_{break} as in (6).

Step 4. Set $t \leftarrow t_{\text{break}}$.

Step 5. Test the Armijo inequality
$$f(x^k + td^k) \leq f(x^k) + \alpha t (d^k)^T \nabla f(x^k). \tag{7}$$

Step 6. If (7) is fulfilled, define $t_k = t$, $x^{k+1} = x^k + t_k d^k$ and return. Otherwise, choose $t_{\text{new}} \in [0.1t, 0.9t]$, set $t \leftarrow t_{\text{new}}$ and go to Step 5.

Theorem 3.1. *Algorithm 3.1 satisfies Assumption A1.*

Proof. By construction, the first requirement of Assumption A1 trivially holds. Let us go to the second requirement. If, for all k large enough, the iterations are computed by Algorithm U, the desired property comes from the basic assumption on this algorithm. It remains to consider the case in which, for infinitely many iterations, one has that x^{k+1} is computed at Step 6. In this case, using standard arguments on minimization algorithms based on sufficient descent directions, we get that any limit point of the corresponding subsequence is, in fact, stationary. Therefore, the theorem is proved. □

In our implementations, we used two differents algorithms with the assumption needed by Algorithm U. One is based on the unconstrained procedure implicit in Gencan [15] and the other on the unconstrained trust-region procedure that underlies Betra [6]. These methods will be called Genlin and Betralin, respectively.

4 Convergence

In this section we prove that Algorithm 3.1 necessarily terminates at a point that satisfies $\|g_P(x^k, \delta^k)\|_\infty \leq \varepsilon$.

The following auxiliary algorithm is a monotone version of the SPG method [17, 18].

Algorithm 4.1. (Monotone SPG)

Let $\widehat{\Omega} \subset I\!R^n$ be a closed and convex set. Assume $\alpha \in (0,1)$, $0 < \sigma_{\min} < \sigma_{\max} < \infty$, $t_{\min} > 0$. Let $x^0 \in \widehat{\Omega}$ be an arbitrary initial point. Given $x^k \in \widehat{\Omega}$, $\sigma_k \in [\sigma_{\min}, \sigma_{\max}]$, the steps of the $k-$th iteration of the algorithm are:

Step 1. *Compute the search direction*

Compute

$$d^k = P_{\widehat{\Omega}}\Big(x^k - \sigma_k \nabla f(x^k)\Big) - x^k.$$

If $d^k = 0$, stop the execution of the algorithm declaring that x_k is a stationary point.

Step 2. *Compute the steplength*

Set $t_{\text{ini}} \geq t_{\min}$ and $t = t_{\text{ini}}$. If

$$f(x^k + td^k) \leq f(x^k) + \alpha t \nabla f(x^k)^T d^k, \tag{8}$$

set x^{k+1} such that $f(x^{k+1}) \leq f(x^k + td^k)$ and finish the iteration. Otherwise, choose $t_{\text{new}} \in [0.1t, 0.9t]$, set $t \leftarrow t_{\text{new}}$ and repeat test (8).

Theorem 4.1. *Let $\widehat{\Omega}$ be convex and closed. Assume that the sequence generated by Algorithm 4.1 is bounded. Then:*

1. *If Algorithm 4.1 does not terminate at $x^k \in \widehat{\Omega}$, then x^{k+1} is well defined.*
2. *If Algorithm 4.1 terminates at x^k then x^k is a stationary point of the problem*

$$\text{Minimize } f(x) \text{ subject to } x \in \widehat{\Omega}. \tag{9}$$

3. *If x^* is a limit point of a sequence generated by Algorithm 4.1 then x^* is a KKT point of (9). Moreover,*

$$\lim_{k\to\infty} \|d^k\| = \lim_{k\to\infty} \|P_{\widehat{\Omega}}(x^k - \sigma_k \nabla f(x^k)) - x^k\| = \lim_{k\to\infty} \|P_{\widehat{\Omega}}(x^k - \nabla f(x^k)) - x^k\| = 0.$$

Proof. See, for example, Theorem 2.1 of [19]. □

Now we are in position of proving the convergence of Algorithm 3.1.

Theorem 4.2. *Assume that Algorithm 3.1 is applied to problem (2). Let Ω be bounded. Then:*

1. *For all $k = 0, 1, 2, \ldots$, if the algorithm does not terminate at x^k, then x^{k+1} is well defined.*
2. *The sequence $\{x^k\}$ generated by the algorithm terminates in a finite number of iterations at a point where $\|g_P(x^k, \delta^k)\|_\infty \le \varepsilon$.*

Proof. The first part of the thesis follows from Assumption A1 and Theorem 4.1. Let us prove the second part. Assume, by contradiction, that the algorithm generates infinitely many iterations. We consider two cases.

1. There are infinitely many iterations of PSPG type.
2. There are only finitely many iterations of PSPG type.

Consider the first case. Let $K \subset I\!N$ be the set of indices k such that x^{k+1} is computed by the Partial SPG method. Since the number of faces F_I is finite, there exists $I \subset \{1, \ldots, p\}$ and K_1 an infinite subset of K such that $x^k \in F_I$ for all $k \in K_1$. Now, the number of different subsets $\Omega(x^k, \delta^k)$ is also finite, therefore, there exist $\widehat{\Omega}$ and K_2, an infinite subset of K_1, such that

$$\Omega(x^k, \delta^k) = \widehat{\Omega} \text{ for all } k \in K_2.$$

Therefore, for all $k \in K_2$,

$$d^k = P_{\widehat{\Omega}}(x^k - \sigma_k \nabla f(x^k)) - x^k.$$

Assume, without loss of generality, that $\widehat{\Omega}$ is defined by the inequalities

$$(a^i)^T x - b_i \le 0 \text{ for all } i = 1, \ldots, q.$$

Let $i \in \{q+1, \ldots, p\}$. Then,

$$(a^i)^T x^k - b_i < -\delta_i^k.$$

Assume that $t \geq 0$ is such that

$$(a^i)^T(x^k + td^k) - b_i = 0.$$

Thus,

$$t = \frac{b_i - (a^i)^T x^k}{(a^i)^T d^k}.$$

Therefore,

$$t > \frac{\delta_i^k}{\|a^i\| \|d^k\|}.$$

Since Ω is compact, the sequence $\{x^k\}$ is bounded and, so, the sequence $\{d^k\}$ is bounded too. Say, $\|d^k\| \leq c$ for all k. Therefore,

$$t > \frac{\delta_i^k}{\|a^i\| c} \geq \frac{\min\{\delta_i^k\}}{\max\{\|a^i\|\} c} \equiv t_{\min}.$$

This implies that the first trial point of the form $x^k + t_{\text{break}} d^k$ that is tested in the PSPG algorithm is bounded away from $t_{\min} > 0$. Therefore, since $f(x^{k+1}) < f(x^k)$ for all k, the sequence $\{x^k\}_{k \in K_2}$ may be thought, after relabeling, as being generated by Algorithm 4.1 applied to the minimization of f on $\widehat{\Omega}$ (taking $t_{\text{ini}} = t_{\text{break}}$). Therefore, any limit point x^* of this sequence is a stationary point of the auxiliary problem. Since $\Omega \subset \widehat{\Omega}$ this implies that x^* is a stationary point of (2). So, $\lim_{k \to K_2} \|g_P(x^k, \delta^k)\| = 0$. Therefore, for $k \in K_2$ large enough, we have that $\|g_P(x^k, \delta^k)\|_\infty \leq \varepsilon$ and the algorithm would have stopped at x^k. This contradicts the assumption that $\{x^k\}$ is infinite.

In the case that there are only finitely many iterations of PSPG type, we have that, for k large enough, all the iterations are internal. Moreover, after a finite number of iterations, all the iterates belong to the same face F_I. By Assumption A1, this implies that there exists k such that $\|g_S(x^k)\|_\infty \leq \varepsilon$. Therefore, at this iteration we must have that $\|g_P(x^k, \delta^k)\|_\infty \leq \varepsilon$, otherwise the next iterate would have been obtained by PSPG. □

5 Implementation

For the implementation of Algorithm 3.1 several decisions are necessary. The practical efficiency of the algorithm strongly depends on the correctness of them. We will sketch the main algorithmic decisions in this section.

5.1 Projection Algorithm

Algorithm 3.1 computes projections on the polytopes Ω or $\Omega(x^k, \delta^k)$. We adopted Subroutine QL, the Goldfarb-Idnani method [36] implemented by Powell [50] and modified by Schittkowski [57] for computing these projections. Subroutine QL can be downloaded from the web site (www.scilab.org) of the open source software Scilab (scientific software package for numerical computations). The Goldfarb-Idnani algorithm is a dual method for positive definite quadratic programming that uses the unconstrained minimizer as initial point. It uses Cholesky and QR decompositions as well as low-rank updating of factorizations.

Subroutine QL has two parameters related to stopping criteria: a small tolerance $\varepsilon_{\text{QL}} > 0$ and a maximum number of iterations MAXIT_{QL}. On return, there is an output parameter INFO_{QL} whose meaning is:

info$_{\text{QL}}$=0: Success. Maximal violation of the normalized constraints (using $1/\|a^i\|_\infty$ for constraint $C_i(x) = (a^i)^T x - b_i \leq 0$, $i = 1, \dots, p$) smaller than or equal to ε_{QL}.

info$_{\text{QL}}$=1: Maximum number of iterations MAXIT$_{\text{QL}}$ reached.

info$_{\text{QL}}$=2: Accuracy is insufficient to mantain increasing function values.

info$_{\text{QL}}$<0: Problem is infeasible.

Note that our Algorithm 3.1 is not necesarily dealing with normalized constraints. Therefore, independently of the stopping criteria satisfied by Subroutine QL, we check feasibility at its final iterate.

Our projections aim to obtain feasible points with tolerance $\varepsilon_{\text{feas}}$. In the terms of (2) this means that

$$C_i(x) \leq \varepsilon_{\text{feas}} \text{ for all } i = 1, \dots, p. \tag{10}$$

In order achieve (10), we proceed as follows:

Step 1. *First trial.*
Call Subroutine QL with $\varepsilon_{\text{QL}} = (\varepsilon_{\text{feas}})^{1.25}$. Let $\bar{x}$ be the projected point and INFO$_{\text{QL}}$ be the output flag. Let $R = \{\bar{x}\}$. If (10) holds for $\bar{x}$ then return $\bar{x}$ declaring "Projection successfully computed". Otherwise, if INFO$_{\text{QL}}$=0 then go to Step 2, else go to Step 3.

Step 2. *Try tighter tolerances aiming to satisfy (10).*
Step 2.1. Set $\varepsilon_{\text{QL}} \leftarrow \varepsilon_{\text{QL}}/100$.

Step 2.2. Call Subroutine QL with ε_{QL}. Let $\bar{x}$ be the projected point and INFO$_{\text{QL}}$ be the output flag. Let $R = R \cup \{\bar{x}\}$. If (10) holds for $\bar{x}$ then return $\bar{x}$ declaring "Projection successfully computed".

Step 2.3. If INFO$_{\text{QL}}$=0 and $\varepsilon_{\text{QL}} > 10^{-16}$ go to Step 2. Otherwise, go to Step 4.

Step 3. *Try a looser tolerance.*
Call Subroutine QL with $\varepsilon_{\text{QL}} = \sqrt{(\varepsilon_{\text{feas}})^{1.25}}$. Let $\bar{x}$ be the projected point and INFO$_{\text{QL}}$ be the output flag. Let $R = R \cup \{\bar{x}\}$. If (10) holds for $\bar{x}$ then return $\bar{x}$ declaring "Projection successfully computed".

Step 4. *Treatment for loss of feasibility.*
Let $\hat{x} = \text{argmin}_{x \in R}\{\max_{i=1,\dots,p}\{C_i(x)\}\}$. If $C_i(\hat{x}) \leq \sqrt{\varepsilon_{\text{feas}}}$ for all $i = 1, \dots, p$, then return $\hat{x}$ declaring "Partial loss of feasibility" and continue the execution of the main algorithm. Otherwise, stop the main algorithm declaring "Projection failure". (A second algorithmic option, not used in the experiments reported in this paper, is not to tolerate any loss of feasibility and to stop the main algorithm declaring "Projection failure" when arriving to this step.)

5.2 Internal Algorithms

As mentioned in Section 3, we employed two internal methods for computing iterations that preserve the current face F_I. The first (GENLIN) is a variation of GENCAN [15] and the second

(Betralin) is the adaptation of Betra [6] to the case in which the internal face is a polytope, instead of a box (as in [6] and [15]). The "unconstrained directions" of Genlin are, therefore, inexact-Newton directions preconditioned with the technique presented in [16]. On the other hand, Betralin, as Betra, uses a trust-region strategy. The differences between Genlin and Betralin in terms of robustness and efficiency were not meaningful in our experiments. Therefore, we are going to report here only experiments related to Genlin.

5.3 Linear Algebra

For computing internal iterations one needs to transform the face F_I into a convex polytope with nonempty interior in a finite-dimensional Euclidian space. The tool for doing this is to compute a basis of the parallel subspace S_I. In this way, every point of F_I may be expressed as the addition of x^k and a linear combination of the coefficients of the basis. The constraints are defined in terms of these coefficients in an obvious way. The basis of S_I is computed using the LQ factorization of the matrix formed by the rows $a^i, i \in I$. When the face F_I changes and new constraints need to be added to form the new current face, the corresponding LQ factorization is updated using the technique described in [35].

5.4 Feasibility and Rounding

In the absence of rounding errors, all the iterates of Algorithm 3.1 are feasible. As a consequence, the algorithm may stop only when the criterion $\|g_P(x^k, \delta^k)\|_\infty \le \varepsilon$ is satisfied, as established by the convergence theory. In floating point computations, however, small losses of feasibility may occur at different stages of the algorithm. The default option in the presence of feasibility loss is to tolerate infeasibilities up to the level $\sqrt{\varepsilon_{\text{feas}}}$ while still requiring projections to satisfy (10). Projections and Partial SPG iterations are used to correct higher levels of infeasibility. As a consequence, at a final successful point x^*, it may happen that $\varepsilon_{\text{feas}} \le C_i(x^*) \le \sqrt{\varepsilon_{\text{feas}}}$. In our experiments, although feasibility was partially lost at a few iterations in a few problems, we always recovered the inequalities (10) at the final point.

If, due to rounding errors, the Internal Algorithm fails to satisfy the monotonicity assumption, a Partial SPG iteration is performed to (presumably) correct this anomaly.

5.5 Parameters and Stopping Criteria

In Algorithm 3.1 we use $\varepsilon = 10^{-8}$ for the stopping criterion related to the sup-norm of $g_P(x^k, \delta^k)$. We used $\varepsilon_{\text{feas}} = \varepsilon$ in (10). The choice of δ^k will be explained below. We set $\sigma_{\min} = 10^{-10}$, $\sigma_{\max} = 10^{10}$ (for the safeguarded spectral coefficient of Algorithm 3.2). We use $\alpha = 10^{-4}$ for the Armijo condition of Algorithm 3.2. In the implementation of Algorithm 3.3, we use the default parameters of Gencan and Betra (see [6] and [15] for details.)

6 Numerical Experiments

The objective of the present section is to test the reliability of Algorithm 3.1, with the implementation features described in Section 5. We chose, as test problems, all the linearly constrained examples included in the CUTEr collection [20], with a limited size. Above this size, sparse matrix techniques need to be used for factorizations, which are not employed in our present implementation. The CUTEr collection has been chosen for testing due to its popularity among algorithmic developers.

We compared GENLIN with the Harwell subroutine VE11 [51], with the popular active set method for large-scale optimization MINOS [49] and with the interior-point algorithm for large nonlinear programming IPOPT [61]. MINOS and IPOPT use sparse matrix technology, which has a benefical effect in the larger problems chosen in this study, therefore, time comparisons with respect to these two algorithms are not meaningful. However, it is interesting to compare the results with respect to the capacity of finding probably global minimizers and the number of function evaluations performed.

All the experiments were run using a 2.2GHz AMD Athlon 64 Processor 3200+ with 1.0GB of RAM memory and a (32 bits) Linux Operating System (Debian 3.4.6-6). The algorithms were coded in double precision Fortran 77 and compiled with g77 (GNU Fortran (GCC) 3.4.6). The compiler optimization option -O3 was adopted.

6.1 Linearly Constrained Test Problems

Linearly constrained optimization problems can always be formulated in the form (2). However, in practice, one uses to say that this type of problems possess three types of constraints: bounds on the variables, equality constraints and proper (not bound) inequality constraints. Of course, each equality constraint can be conceptually reduced to two inequality constraints and, for the basic description of our Algorithm 3.1, this distinction is quite irrelevant.

From now on, n_{bounds} will be the number of bound constraints of a problem. We consider that a variable bounded both above and below contributes with two bounds. We denote by n_{eq} the number of equality constraints and by n_{ineq} the number of inequality constraints. The number of independent variables will be denoted by n. We selected, for our tests, all the problems of the CUTEr collection with at most 500 variables and with $1 \leq n_{\text{eq}} + n_{\text{ineq}} \leq 2000$. This corresponds to 133 problems. The main characteristics of the problems are in Table 1. In this table, the problems are roughly ordered from the smallest to the largest one. Some problems from CUTEr have variables where the lower bound coincides with the upper bound. In Table 1, $n = n_1(n_2)$ means that problem has n_1 genuine variables plus n_2 fixed variables. In this case, n_{bounds} is the number of bounds imposed to the genuine variables.

Problems NASH, MODEL, ARGLALE, ARGLBLE, ARGLCLE and LINCONT are infeasible and, thus, Algorithm 3.1 stops at Step 1 saying that it failed to compute x^1 (the projection of the initial guess x^0 onto the feasible set). These 6 problems were removed from the experiments below. So, from now on we consider the remaining 127 problems.

6.2 Supporting Internal Decisions

6.2.1 Choice of δ^k

We tried two choices for the Partial SPG parameters δ^k:

Choice 1: $\delta_i^k = \bar{\delta}^k \max\{1, |b_i|\}$, where $\bar{\delta}^1 = 0.1$ and $\bar{\delta}^{k+1} = 100\ \bar{\delta}^k$ whenever the iteration k is PSPG and $t_{\text{break}} < 0.1$. Else $\bar{\delta}^{k+1} = \bar{\delta}^k$.

Choice 2: $\delta_i^k \equiv \infty\ \forall\ i$ and $\forall\ k$.

The second choice corresponds to use the full SPG iteration in the cases in which PSPG is invoked.

Problem	n	n_{bounds}	n_{eq}	n_{ineq}	Problem	n	n_{bounds}	n_{eq}	n_{ineq}
EXTRASIM	2	1	1	0	AVGASB	8	16	0	10
HS9	2	0	1	0	DUALC5	8	16	1	277
TAME	2	2	1	0	DUALC8	8	16	1	502
HS21	2	4	0	1	DUALC1	9	18	1	214
HS35MOD	2(1)	2	0	1	HS112	10	10	3	0
HUBFIT	2	1	0	1	ODFITS	10	10	6	0
LSQFIT	2	1	0	1	GENHS28	10	0	8	0
BOOTH	2	0	2	0	PORTFL1	12	24	1	0
HIMMELBA	2	0	2	0	PORTFL2	12	24	1	0
SUPERSIM	2	1	2	0	PORTFL3	12	24	1	0
SIMPLLPA	2	2	0	2	PORTFL4	12	24	1	0
ZECEVIC2	2	4	0	2	PORTFL6	12	24	1	0
HS24	2	2	0	3	LOTSCHD	12	12	7	0
SIMPLLPB	2	2	0	3	HS118	15	30	0	29
PT	2	0	0	501	HS119	16	32	8	0
SIPOW1M	2	0	0	2000	NASH	18(54)	6	24	0
SIPOW1	2	0	0	2000	FCCU	19	19	8	0
SIPOW2M	2	0	0	2000	RES	20	40	12	2
SIPOW2	2	0	0	2000	DEGENLPA	20	40	15	0
HS28	3	0	1	0	DEGENLPB	20	40	15	0
HS62	3	6	1	0	KSIP	20	0	0	1001
HS35I	3	6	0	1	MAKELA4	21	0	0	40
HS35	3	3	0	1	WATER	31	62	10	0
HS36	3	6	0	1	LOADBAL	31	42	11	20
HS37	3	6	0	2	MODEL	42(1500)	84	23	15
STANCMIN	3	3	0	2	HIMMELBJ	43(2)	43	14	0
ZANGWIL3	3	0	3	0	DALLASS	46	92	31	0
TFI2	3	0	0	101	AVION2	49	98	15	0
TFI3	3	0	0	101	GOFFIN	51	0	0	50
OET1	3	0	0	1002	DUAL4	75	150	1	0
HONG	4	8	1	0	LINSPANH	81(16)	162	33	0
HS41	4	8	1	0	SPANHYD	81(16)	162	33	0
LIN	4	8	2	0	QPCBLEND	83	83	43	31
HS76I	4	8	0	3	QPNBLEND	83	83	43	31
HS76	4	4	0	3	DUAL1	85	170	1	0
S277-280	4	4	0	4	DUAL2	96	192	1	0
HS44NEW	4	4	0	6	HIMMELBI	100	200	0	12
HS44	4	4	0	6	DUAL3	111	222	1	0
BIGGSC4	4	8	0	13	SMBANK	117	234	64	0
HATFLDH	4	8	0	13	QPCBOEI2	143	197	4	181
OET3	4	0	0	1002	QPNBOEI2	143	197	4	181
SIPOW3	4	0	0	2000	AGG	163	163	36	452
SIPOW4	4	0	0	2000	HYDROELS	167(2)	334	0	336
HS48	5	0	2	0	GMNCASE1	175	0	0	300
HS49	5	0	2	0	GMNCASE4	175	0	0	350
BT3	5	0	3	0	GMNCASE2	175	0	0	1050
HS50	5	0	3	0	GMNCASE3	175	0	0	1050
HS51	5	0	3	0	SSEBLIN	192(2)	360	48	24
HS52	5	0	3	0	DALLASM	196	392	151	0
HS53	5	10	3	0	ARGLCLE	200	0	399	0
LSNNODOC	5	6	4	0	ARGLALE	200	0	400	0
HS268	5	0	0	5	ARGLBLE	200	0	400	0
S268	5	0	0	5	PRIMALC1	230	215	0	9
HS86	5	5	0	10	PRIMALC2	231	229	0	7
EXPFITA	5	0	0	22	LINCONT	249(1008)	0	419	0
EXPFITB	5	0	0	102	PRIMALC5	287	278	0	8
EXPFITC	5	0	0	502	PRIMAL1	325	1	0	85
HS54	6	12	1	0	QPCBOEI1	384	540	9	431
HS55	6	8	6	0	QPNBOEI1	384	540	9	431
PENTAGON	6	0	0	15	QPCSTAIR	385(82)	385	209	147
HS21MOD	7	8	0	1	QPNSTAIR	385(82)	385	209	147
EQC	7(2)	14	0	3	STEENBRA	432	432	108	0
QCNEW	7(2)	14	0	3	STATIC3	434	144	96	0
QC	7(2)	14	0	4	STEENBRB	468	468	108	0
DUALC2	7	14	1	228	STEENBRD	468	468	108	0
HS105	8	16	0	1	STEENBRF	468	468	108	0
AVGASA	8	16	0	10					

Table 1: Linearly constrained selected problems from CUTEr.

There are 10 problems (BOOTH, HIMMELBA, SUPERSIM, TAME, STANCMIN, ZANGWIL3, HS55, RES, LINSPANH and GMNCASE4) for which the solution is given by x^1, the projection of the initial point x^0 onto the feasible set. Since the versions with choices 1 and 2 for δ^k coincide in those cases, we eliminate them from the experiments of the present subsection. So, we have at hand the remaining 117 problems.

Both versions found equivalent functional values at their final iterate in all the 117 problems. (We say that f_1 and f_2 are *equivalent* if

$$[|f_1 - f_2| \le \max\{10^{-10}, 10^{-6}\min\{|f_1|, |f_2|\}\}] \text{ or } [f_1 \le -10^{20} \text{ and } f_2 \le -10^{20}].) \qquad (11)$$

For comparing these two versions of the method we use the CPU time as a performance measure. (Note that the cost of the projection may depend on δ^k.)

Figure 1 shows the performance profile [31]. The figure shows that there is no difference in the robustness of both versions. On the other hand, with Choice 1 the method is more efficient. Choice 1 is faster than Choice 2 in 70.08% of the problems; whereas the percentage of cases in which Choice 2 is faster is 36.75%.

We arrived to the dynamic Choice 1 after tests that showed us that using a fixed small δ^k could be harmful in critical cases. When, in a neighborhood of x^k one has many active constraints that are not active at x^k, small δ^k–PSPG iterations do not take most of these constraints into account and, so, for obtaining a feasible x^{k+1} one needs to use a small fraction of the natural PSPG step d^k. As a consequence, few constraints are added at each iteration and convergence is very slow. When this phenomenon is detected, Choice 1 increases δ^k, in such a way that the (previously not considered) nearly active constraints are now taken into account for the projection. Similar considerations lead to the tolerant quadratic programming strategy of VE11 [51].

6.2.2 Choice of σ_k

According to the experiments in the previous subsection, we adopted Choice 1 for δ^k. We will now corroborate that, in Algorithm 3.2, using σ_k as the safeguarded spectral step σ_{spec}, where

$$\sigma_{\text{spec}} = \begin{cases} \frac{\|x^k - x^{k-1}\|^2}{(x^k - x^{k-1})^T(\nabla f(x^k) - \nabla f(x^{k-1}))}, & \text{if } (x^k - x^{k-1})^T(\nabla f(x^k) - \nabla f(x^{k-1})) > 0, \\ 1, & \text{otherwise,} \end{cases}$$

is better than using $\sigma_k \equiv 1$.

Since both choices coincide for problems with linear or constant objective function, we will exclude the 26 problems with this characteristic. Therefore, we have at hand the remaining 101 problems. Both choices also coincide if Algorithm 3.1 never executes Step 6, as is the case for other 40 problems (including the 10 problems for which x^1 is the solution). Eliminating these 40 problems too, we arrive to a set of 61 problems to be considered for the numerical experiments of the current subsection.

We will use the number of function evaluations to evaluate the efficiency of these methods. Figure 2 shows the corresponding performance profile. With both choices we found equivalent functional values for all the problems but the figure shows that using the spectral step is considerably more efficient.

The detailed performance of GENLIN is given in Tables 2 and 3. In these tables, ITER is the number of iterations, PSPGit is the number of Partial SPG iterations, FCNT is the

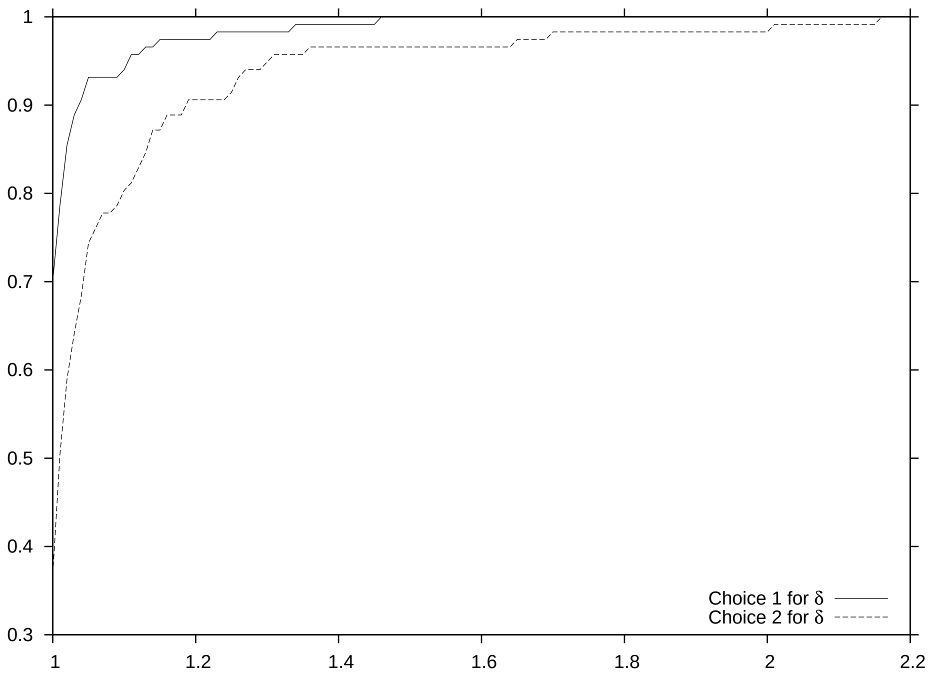

Figure 1: Performance profile comparing choices 1 and 2 of the Partial SPG parameter δ^k.

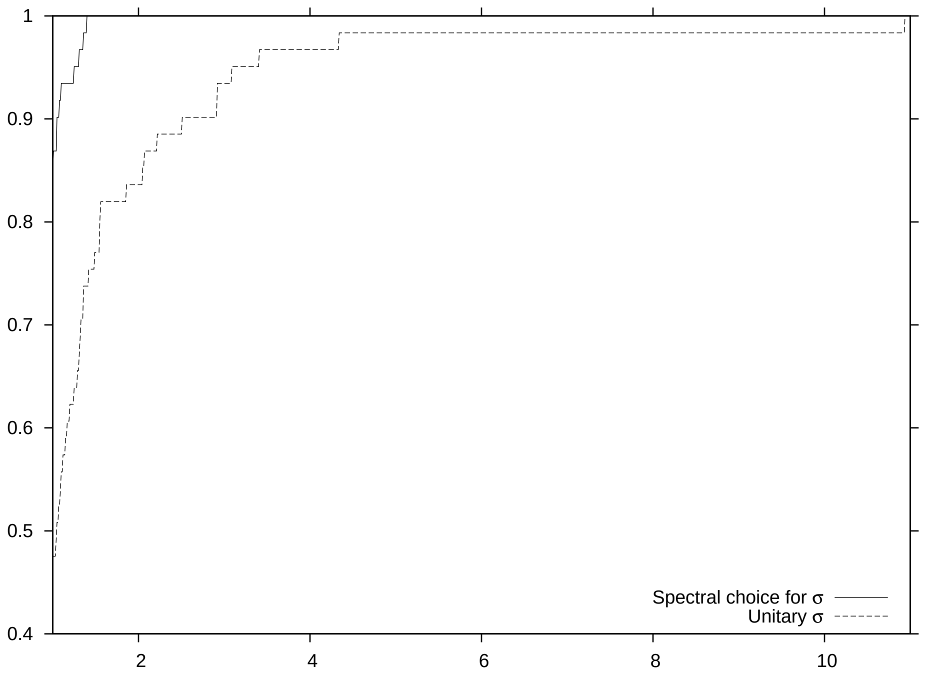

Figure 2: Performance profile comparing the choice of the safeguarded spectral step for σ_k and $\sigma_k \equiv 1$.

Problem	ITER	PSPGit	FCNT	$f(x^*)$	$\|C(x^*)_+\|_\infty$	Time	ProjTime	
EXTRASIM	2	0	2	1.00000E+00	4.4E−16	0.00007	20.70%	(3)
HS9	2	0	6	-5.00000E−01	1.7E−13	0.00007	19.30%	(4)
TAME	1	0	1	0.00000E+00	0.0E+00	<0.00001	60.00%	(2)
HS21	2	0	2	-9.99600E+01	0.0E+00	0.00007	27.20%	(4)
HS35MOD	3	0	3	2.50000E−01	0.0E+00	0.00008	8.80%	(3)
HUBFIT	5	1	5	1.68935E−02	0.0E+00	0.00018	11.10%	(5)
LSQFIT	5	1	5	3.37870E−02	0.0E+00	0.00017	14.30%	(5)
BOOTH	1	0	1	0.00000E+00	0.0E+00	<0.00001	59.00%	(2)
HIMMELBA	1	0	1	0.00000E+00	0.0E+00	<0.00001	59.60%	(2)
SUPERSIM	1	0	1	6.66667E−01	0.0E+00	<0.00001	63.30%	(2)
SIMPLLPA	2	1	2	1.00000E+00	0.0E+00	0.00007	37.50%	(4)
ZECEVIC2	3	1	4	-4.12500E+00	0.0E+00	0.00010	17.20%	(5)
HS24	3	0	3	-1.00000E+00	0.0E+00	0.00011	3.80%	(3)
SIMPLLPB	2	0	2	1.10000E+00	0.0E+00	0.00007	22.80%	(3)
PT	4	2	4	1.78394E−01	0.0E+00	0.00095	25.60%	(5)
SIPOW1M	15	9	15	-1.00000E+00	0.0E+00	0.01140	3.30%	(12)
SIPOW1	15	9	15	-1.00000E+00	0.0E+00	0.01150	5.90%	(12)
SIPOW2M	14	8	14	-1.00000E+00	0.0E+00	0.01010	3.10%	(11)
SIPOW2	15	8	15	-1.00000E+00	0.0E+00	0.01080	1.10%	(11)
HS28	2	0	2	3.59918E−30	1.6E−15	0.00004	23.20%	(3)
HS62	10	0	13	-2.62725E+04	0.0E+00	0.00043	6.10%	(4)
HS35I	3	0	3	1.11111E−01	0.0E+00	0.00013	15.80%	(4)
HS35	3	0	3	1.11111E−01	0.0E+00	0.00012	4.50%	(4)
HS36	4	0	4	-3.30000E+03	0.0E+00	0.00015	14.80%	(4)
HS37	6	0	6	-3.45600E+03	0.0E+00	0.00023	14.00%	(4)
STANCMIN	1	0	1	4.25000E+00	0.0E+00	<0.00001	73.30%	(2)
ZANGWIL3	1	0	1	0.00000E+00	0.0E+00	<0.00001	74.40%	(2)
TFI2	22	10	22	6.49031E−01	5.6E−17	0.00187	17.00%	(13)
TFI3	5	2	5	4.30116E+00	0.0E+00	0.00048	33.00%	(6)
OET1	13	6	13	5.38243E−01	2.8E−17	0.00667	7.30%	(9)
HONG	7	0	7	2.25711E+01	2.2E−16	0.00031	20.00%	(4)
HS41	6	1	6	1.92593E+00	4.4E−16	0.00023	28.60%	(5)
LIN	2	1	3	-1.75775E−02	5.0E−13	0.00009	45.30%	(5)
HS76I	6	1	6	-4.68182E+00	8.9E−16	0.00025	16.10%	(5)
HS76	6	1	6	-4.68182E+00	8.9E−16	0.00023	15.10%	(5)
S277-280	4	0	4	5.07619E+00	0.0E+00	0.00017	13.50%	(3)
HS44NEW	6	0	6	-1.50000E+01	0.0E+00	0.00024	12.10%	(3)
HS44	6	2	6	-1.50000E+01	0.0E+00	0.00023	24.20%	(5)
BIGGSC4	4	1	4	-2.43750E+01	0.0E+00	0.00018	17.90%	(4)
HATFLDH	3	1	3	-2.43750E+01	0.0E+00	0.00014	41.90%	(5)
OET3	4	2	4	4.50505E−03	2.8E−17	0.00281	37.90%	(5)
SIPOW3	9	3	9	5.34659E−01	0.0E+00	0.00950	6.20%	(6)
SIPOW4	11	3	11	2.72362E−01	5.6E−17	0.01260	6.70%	(6)
HS48	2	0	2	3.94430E−31	8.9E−16	0.00006	25.30%	(3)
HS49	20	0	20	1.06000E−11	5.3E−15	0.00084	4.10%	(4)
BT3	2	0	2	4.09302E+00	2.8E−15	0.00010	40.60%	(4)
HS50	10	0	10	2.66302E−28	2.2E−14	0.00039	8.50%	(3)
HS51	2	0	2	1.77494E−30	1.3E−15	0.00006	29.20%	(3)
HS52	2	0	2	5.32665E+00	3.2E−16	0.00010	46.80%	(4)
HS53	2	0	2	4.09302E+00	2.2E−16	0.00010	44.70%	(4)
LSNNODOC	7	1	7	1.23112E+02	8.9E−16	0.00030	21.40%	(5)
HS268	7	2	20	3.63798E−12	0.0E+00	0.00033	10.50%	(5)
S268	7	2	20	3.63798E−12	0.0E+00	0.00033	9.30%	(5)
HS86	10	2	10	-3.23487E+01	8.9E−16	0.00053	14.00%	(6)
EXPFITA	19	3	19	1.13661E−03	1.4E−14	0.00134	3.60%	(7)
EXPFITB	24	4	24	5.01937E−03	3.8E−15	0.00426	3.90%	(8)
EXPFITC	46	12	48	2.33026E−02	5.7E−14	0.03330	2.90%	(16)
HS54	851	16	861	-9.03490E−01	1.1E−09	0.03920	2.00%	(20)
HS55	1	0	1	6.66667E+00	8.9E−16	0.00002	78.80%	(2)
PENTAGON	15	0	15	1.36522E−04	0.0E+00	0.00073	9.10%	(4)
HS21MOD	2	0	2	-9.59600E+01	0.0E+00	0.00009	47.60%	(4)
EQC	2	1	2	-8.29548E+02	0.0E+00	0.00011	62.50%	(4)
QCNEW	2	1	2	-8.06522E+02	0.0E+00	0.00012	66.70%	(4)
QC	7	0	7	-9.56538E+02	0.0E+00	0.00041	14.40%	(3)
DUALC2	7	1	7	3.55131E+03	0.0E+00	0.00161	9.90%	(5)
HS105	24	2	30	1.06188E+03	0.0E+00	0.02570	0.60%	(6)
AVGASA	4	0	4	-4.63193E+00	1.1E−16	0.00024	36.70%	(4)

Table 2: Performance of GENLIN (Part I).

Problem	ITER	PSPGit	FCNT	$f(x^*)$	$\|C(x^*)_+\|_\infty$	Time	ProjTime	
AVGASB	4	0	4	-4.48322E+00	1.2E−16	0.00023	32.40%	(4)
DUALC5	5	0	5	4.27233E+02	0.0E+00	0.00166	6.00%	(4)
DUALC8	11	2	12	1.83094E+04	0.0E+00	0.00571	1.90%	(6)
DUALC1	12	2	13	6.15525E+03	0.0E+00	0.00323	2.50%	(6)
HS112	22	2	31	-4.77611E+01	4.4E−16	0.00180	7.70%	(6)
ODFITS	6	0	6	-2.38003E+03	2.3E−13	0.00043	35.90%	(4)
GENHS28	2	0	2	9.27174E−01	1.6E−15	0.00020	62.90%	(4)
PORTFL1	8	0	8	2.04863E−02	3.3E−16	0.00082	12.50%	(4)
PORTFL2	9	0	9	2.96892E−02	2.2E−16	0.00090	10.40%	(4)
PORTFL3	11	1	12	3.27497E−02	2.2E−16	0.00107	8.20%	(5)
PORTFL4	8	0	8	2.63070E−02	2.2E−16	0.00085	11.40%	(4)
PORTFL6	8	0	8	2.57918E−02	1.1E−16	0.00084	9.10%	(4)
LOTSCHD	5	0	5	2.39842E+03	2.5E−14	0.00040	57.50%	(4)
HS118	16	1	16	6.64820E+02	1.4E−14	0.00136	31.30%	(5)
HS119	12	1	12	2.44900E+02	8.9E−16	0.00195	29.90%	(5)
FCCU	6	1	6	1.11491E+01	8.9E−15	0.00069	62.30%	(5)
RES	1	0	1	0.00000E+00	9.2E−15	0.00006	87.60%	(2)
DEGENLPA	3	1	3	3.06039E+00	1.1E−16	0.00096	87.00%	(4)
DEGENLPB	2	0	2	-3.07312E+01	7.5E−16	0.00070	89.00%	(3)
KSIP	23	5	25	5.75798E−01	3.3E−16	0.07820	12.90%	(9)
MAKELA4	16	0	16	2.10942E−15	3.8E−15	0.00149	18.20%	(3)
WATER	21	2	23	1.05494E+04	1.1E−13	0.00306	65.40%	(6)
LOADBAL	15	3	14	4.52851E−01	1.4E−14	0.00343	47.40%	(7)
HIMMELBJ	622	102	1233	-1.91034E+03	2.0E−10	0.22000	31.20%	(129)
DALLASS	49	3	60	-3.23932E+04	8.3E−14	0.02190	33.00%	(7)
AVION2	5	0	5	9.46801E+07	1.8E−12	0.00312	78.20%	(4)
GOFFIN	7	0	7	-8.65974E−15	5.4E−14	0.00666	72.80%	(3)
DUAL4	19	1	19	7.46091E−01	0.0E+00	0.02750	17.00%	(5)
LINSPANH	1	0	1	-7.70000E+01	1.4E−13	0.00134	98.20%	(2)
SPANHYD	6	2	7	2.39738E+02	5.1E−13	0.01820	90.30%	(6)
QPCBLEND	8	2	10	-7.84254E−03	1.2E−17	0.03760	93.20%	(6)
QPNBLEND	20	5	22	-9.13614E−03	1.2E−15	0.06240	91.30%	(9)
DUAL1	49	3	57	3.50130E−02	1.1E−15	0.13600	5.60%	(7)
DUAL2	8	0	8	3.37337E−02	8.9E−16	0.04150	12.40%	(4)
HIMMELBI	65	4	67	-1.73557E+03	2.8E−14	0.03680	73.70%	(8)
DUAL3	30	3	34	1.35756E−01	5.6E−16	0.15500	8.30%	(7)
SMBANK	54	5	69	-7.12929E+06	2.3E−10	0.18100	66.10%	(10)
QPCBOEI2	10	2	9	8.17196E+06	1.8E−12	0.17500	93.90%	(6)
QPNBOEI2	42	4	40	1.37140E+06	1.1E−12	0.28100	87.70%	(8)
AGG	60	5	60	-3.59918E+07	4.1E−09	0.44900	65.30%	(8)
HYDROELS	16	3	16	-3.58227E+06	3.6E−14	0.32900	92.00%	(7)
GMNCASE1	16	0	16	2.66973E−01	4.2E−17	0.25000	42.90%	(4)
GMNCASE4	1	0	1	5.94688E+03	1.8E−15	0.23600	95.90%	(2)
GMNCASE2	28	1	28	-9.94445E−01	3.5E−17	0.64300	26.00%	(5)
GMNCASE3	26	1	27	1.52515E+00	4.2E−17	0.63100	29.20%	(5)
SSEBLIN	100	4	100	1.61706E+07	5.8E−11	0.28600	84.10%	(7)
DALLASM	40	2	43	-4.81982E+04	6.4E−13	0.68900	73.00%	(6)
PRIMALC1	6	1	6	-6.15525E+03	1.5E−11	0.13800	100.00%	(5)
PRIMALC2	5	2	6	-3.55131E+03	1.1E−10	0.17200	100.00%	(6)
PRIMALC5	7	1	7	-4.27233E+02	1.8E−11	0.27600	97.30%	(5)
PRIMAL1	3	1	3	-3.50130E−02	2.5E−16	0.53300	98.40%	(5)
QPCBOEI1	19	2	20	1.15039E+07	1.9E−13	5.40000	94.50%	(6)
QPNBOEI1	114	7	119	6.77711E+06	1.2E−12	14.02000	91.60%	(11)
QPCSTAIR	9	1	9	6.20439E+06	2.8E−14	4.67000	97.50%	(5)
QPNSTAIR	15	2	15	5.14603E+06	2.8E−14	5.26000	96.50%	(6)
STEENBRA	38	4	37	1.69577E+04	2.3E−13	5.28000	98.40%	(8)
STATIC3	8	0	8	-1.05655E+20	2.9E−09	2.11000	98.90%	(7)
STEENBRB	116	9	111	9.07586E+03	2.3E−12	9.96000	97.20%	(13)
STEENBRD	144	14	149	9.14472E+03	2.3E−11	13.83000	97.60%	(18)
STEENBRF	114	10	108	8.99185E+03	1.5E−10	10.76000	97.60%	(14)

Table 3: Performance of Genlin (Part II).

number of functional evaluations, $f(x^*)$ is the functional value at the final iterate, $\|C(x^*)_+\|_\infty$ is the maximal violation of feasibility, Time is the total CPU time in seconds and ProjTime is the percentage of Time used by Subroutine QL to compute projections (the total number of projections is within parentheses). In the cases in which the CPU time is very small, we obtained the correct value running the subroutine many times and taking the average. In all the problems except STATIC3 and HIMMELBJ, GENLIN stopped because the criterion $\|g_P(x^k, \delta^k)\| \leq \varepsilon = 10^{-8}$ was fulfilled. In the case of STATIC3, GENLIN stopped because $f(x^k) < -10^{20}$. In the case of HIMMELBJ, GENLIN stopped at a feasible point with well-defined objetive function, after several trials of evaluating the objective function at iterates where it was not well-defined.

6.3 Comparison with Ve11

VE11 is a Harwell subroutine that implements Powell's tolerant method [51]. It has only one relevant parameter $\varepsilon_{\text{VE11}}$ related to tolerances and stopping criteria, which was defined for these experiments as $\varepsilon_{\text{VE11}} = 10^{-8}$.

We applied VE11 to the 127 feasible problems of Table 1 (details can be found in [7]). The final iterate of VE11 does not satisfy the feasibility requirement (10) with tolerance $\varepsilon_{\text{feas}} = 10^{-8}$ in 16 problems:

- Problem STEENBRA is the unique problem for which VE11 stopped due the CPU time limit of 10 minutes imposed for each pair problem/method in our numerical experiments.
- In problems WATER and STATIC3, VE11 stops saying that the current point is feasible but the line search fails to reduce the objective function value.
- In problems SMBANK, DALLASM, STEENBRB, STEENBRD, STEENBRF, VE11 stops saying that the current point is feasible but rounding errors seem to be preventing higher accuracy.
- In problems HS55, LINSPANH and SPANHYD, VE11 stops saying that the equality constraints and the bounds on the variables are incompatible.
- In problems AGG, GMNCASE4, DALLASS, QPCBOEI2 and QPNBOEI2, VE11 stops saying that it is possible to satisfy all the equality constraints and bounds but the general inequality constraints cannot be satisfied.

In the other 111 problems, the final iterate of VE11 is feasible (in the sense of (10) with tolerance $\varepsilon_{\text{feas}} = 10^{-8}$). In 84 out of those 111 problems, VE11 stops satisfying its stopping criterion related to "success". For the remaining 27 problems, the diagnostics provided by the subroutine were:

- In problems HS62 and PRIMALC1, VE11 stops saying the current point is feasible but rounding errors seem to be preventing higher accuracy.
- In problems HS268, S268, QCNEW, DUALC8, HS105, DUALC1, HS119, LOADBAL, AVION2, QPCBLEND, QPNBLEND, DUAL1, DUAL2, DUAL3, HYDROELS, GMNCASE1, GMNCASE2, SSEBLIN, PRIMAL1 and QPCSTAIR, VE11 stops saying that the current point is feasible but the line search fails to reduce the objective function value.
- In problem HIMMELBJ, VE11 stops saying that the equality constraints and the bounds on the variables are incompatible.

- In problems HIMMELBI, QPCBOEI1, QPNBOEI1 and QPNSTAIR, VE11 stops saying that it is possible to satisfy all the equality constraints and bounds but the general inequality constraints cannot be satisfied.

Considering the 111 problems in which VE11 obtained feasible final points, we observed that, in most of them, the final functional value at the final point is equivalent to that obtained by GENLIN. The exceptions were:

1. In 8 problems (LSNNODOC, QPCBLEND, QPNBLEND, HIMMELBI, QPCBOEI1, QPNBOEI1, QPC-STAIR, QPNSTAIR) GENLIN obtained a smaller final functional value than VE11.
2. In 3 problems (HS54, EQC, HS105) VE11 obtained a smaller final functional value than GENLIN.
3. In 2 problems (QCNEW, HIMMELBJ) the final functional value provided by VE11 was not defined (NaN message).

In the remaining 98 problems both GENLIN and VE11 obtained feasible points with equivalent functional values. Considering these 98 problems:

1. GENLIN performed less function evaluations in 83 cases.
2. VE11 performed less function evaluations in 11 cases.
3. GENLIN and VE11 performed the same number of function evaluations in 4 cases.
4. GENLIN was faster than VE11 (up to a tolerance of 10%) in 15 cases.
5. VE11 was faster than GENLIN (up to a tolerance of 10%) in 79 cases.
6. GENLIN and VE11 spent the same amount of time (up to a tolerance of 10%) in 4 cases.
7. Restricting the three items above to the problems in which one of the methods used more than 0.1 seconds (11 problems) GENLIN was faster than VE11 7 times and VE11 was faster than GENLIN 3 times (both methods spent the same time in one case).

A performance profile using the number of functional evaluations as a performance measurement can be seen in Figure 3.

6.4 Comparison with Minos

We ran the experiments of this subsection using MINOS [49] with feasibility and optimality tolerances $\varepsilon_{\text{feas}} = \varepsilon_{\text{opt}} = 10^{-8}$. All the other parameters were set with their default values.

We applied MINOS to the 127 feasible problems of Table 1 (details can be found in [7]). The final iterate of MINOS in 2 out of the 127 problems (HS54 and AGG) does not satisfy the feasibility requirement (10) with tolerance $\varepsilon_{\text{feas}} = 10^{-8}$. In the other 125 problems, the final iterate of MINOS is feasible with tolerance $\varepsilon_{\text{feas}} = 10^{-8}$. In 119 problems, MINOS stops satisfying its stopping criterion related to "success". For the remaining 6 problems, the diagnostics were:

- In problem STATIC3, MINOS stops declaring that the problem is unbounded (or badly scaled).

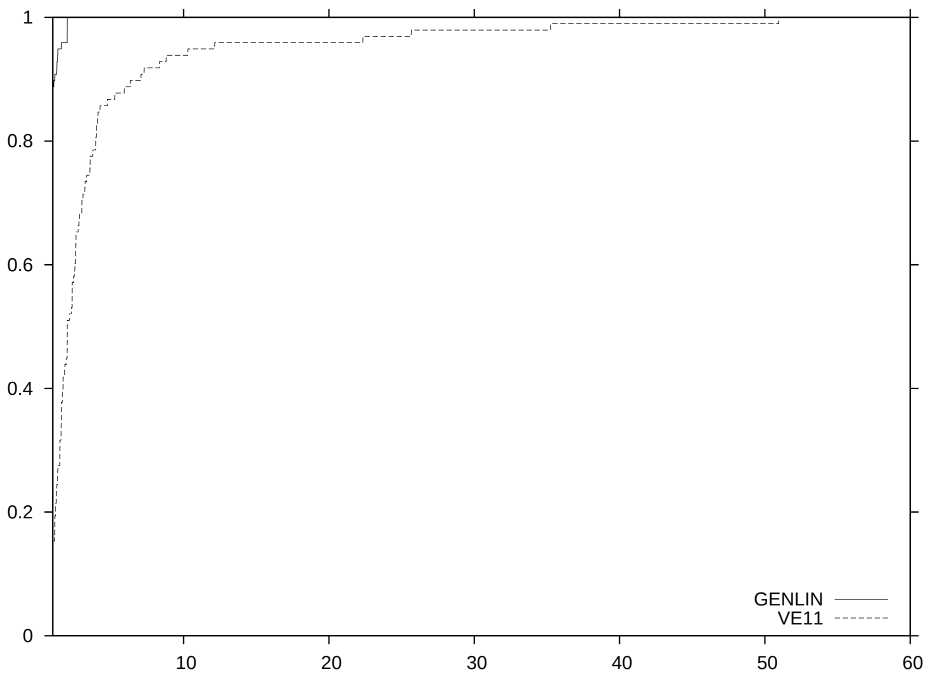

Figure 3: Performance profiles of GENLIN and VE11 in the 83 problems (out of 127) for which both methods obtained feasible points with equivalent functional values. Number of functional evaluations is being used as performance measurement.

- In problem HS55, MINOS stops declaring that the current point cannot be improved.
- In problems HS118, HIMMELBJ, SPANHYD and STEENBRD, MINOS stops declaring that a near-optimal solution was found.

Considering the 125 problems in which MINOS obtained feasible final points, we observed that, in most of them, the functional value at the final point is equivalent to the one obtained by GENLIN. The exceptions are:

1. In 6 problems (HS41, PENTAGON, QC, HS118, QPNBLEND, STEENBRD) GENLIN obtained a smaller functional value than MINOS.
2. In 6 problems (HATFLDH, LIN, EQC, HS105, DEGENLPA, QPNBOEI1, MINOS obtained a smaller functional value than GENLIN.
3. In one problem (HIMMELBJ) the final functional value provided by MINOS was not defined (NaN message).

Considering the 111 problems in which GENLIN and MINOS obtained feasible points with equivalent functional values (excluding here problem STATIC3 for which both methods detected, by different ways, that it seems to be unbounded), we observed that:

1. GENLIN performed less function evaluations in 84 cases.
2. MINOS performed less function evaluations in 26 cases.

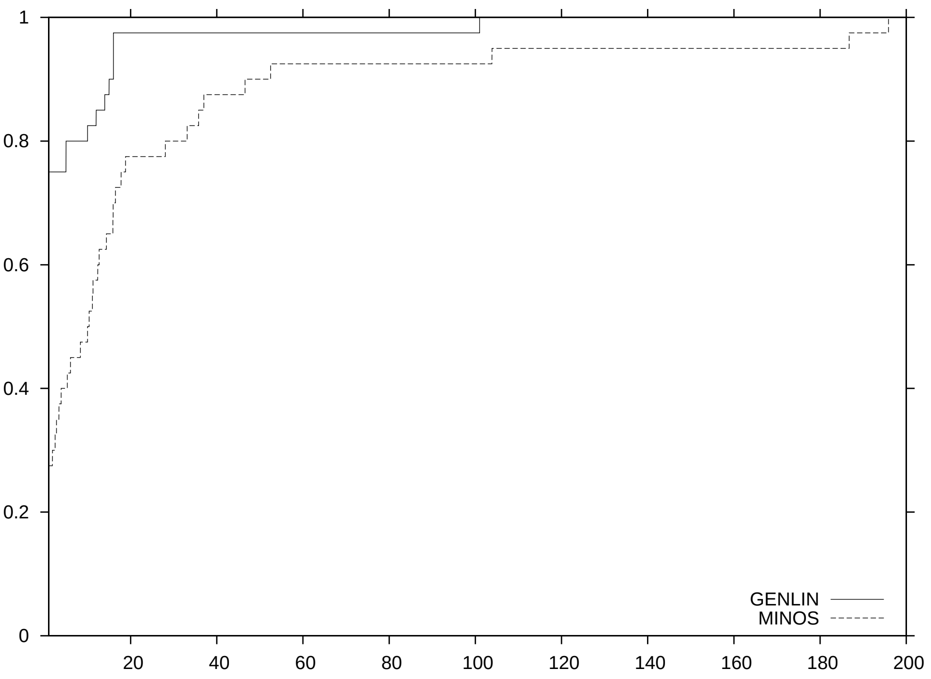

Figure 4: Performance profiles of Genlin and Minos in the 111 problems (out of 127) for which both methods obtained feasible points with equivalent functional values. Number of functional evaluations is being used as performance measurement.

3. Genlin and Minos performed the same number of function evaluations in one case.

4. Considering the problems in which one of the methods used at least 0.01 seconds (40 problems) Genlin was faster than Minos (up to a tolerance of 10%) 17 times and Minos was faster than Genlin (up to a tolerance of 10%) 22 times (both methods spent the same amount of time in one case).

5. Restricting the item above to the problems in which one of the methods used at least 0.1 seconds (27 problems) Genlin was faster than Minos 8 times and Minos was faster than Genlin 18 times (both methods spent the same amount of time in one case).

A performance profile using the number of functional evaluations as a performance measurement can be seen in Figure 4. We recall that, due to the use of sparsity techniques in Minos (and not in Genlin), computer time comparisons have little significance.

6.5 Comparison with Ipopt

Ipopt [61] is one of the best-qualified interior point methods for large-scale constrained optimization. We ran this algorithm using all its default parameters.

We applied Ipopt to the 127 feasible problems of Table 1 (details can be found in [7]). The final iterate of Ipopt in one problem (himmelbj) does not satisfy the feasibility requirement with tolerance $\varepsilon_{\text{feas}} = 10^{-8}$. In this case Ipopt stops declaring that restoration failed.

In the other 126 problems, the final iterate of IPOPT is feasible with tolerance $\varepsilon_{\text{feas}} = 10^{-8}$. In 121 out of those 126 problems, IPOPT stops satisfying its stopping criterion related to "success". For the remaining 5 problems, the diagnostics were:

- In problems EQC and DALLASM, IPOPT stops declaring that the problem was solved to an acceptable level.
- In problem STATIC3, IPOPT stops declaring that the problem might be unbounded.
- In problem QCNEW, IPOPT stops declaring that restoration failed.
- In problem LIN, IPOPT stops declaring that search direction is becoming too small.

Considering the 126 problems in which IPOPT obtained feasible final points, we observed that, in most of them, the final functional value at the final point is equivalent to the one obtained by GENLIN. The exceptions were:

1. In 12 problems (HS44, EXPFITA, EXPFITB, EXPFITC, HS268, S268, HS55, PENTAGON, EQC, MAKELA4, GOFFIN, PRIMAL1) GENLIN obtained a smaller final functional value than IPOPT.
2. In 17 problems (BIGGSC4, HATFLDH, HS54, QCNEW, HS105, DUALC1, DEGENLPA, DEGENLPB, QPCBLEND, QPNBLEND, QPNBOEI2, GMNCASE1, GMNCASE2, GMNCASE3, PRIMALC5, QPNBOEI1, STEENBRD) IPOPT obtained a smaller final functional value than GENLIN.
3. In one problem (LIN) the final functional value provided by IPOPT was not defined (NaN message).

In the remaining 95 problems (excluding here problem STATIC3 for which both methods detected, by different ways, that it seems to be unbounded) GENLIN and IPOPT obtained feasible points with equivalent functional values. Considering these 95 problems:

1. GENLIN performed less function evaluations in 71 cases.
2. IPOPT performed less function evaluations in 15 cases.
3. GENLIN and IPOPT performed the same number of function evaluations in 9 cases.
4. Considering the problems in which one of the methods used at least 0.01 seconds (54 problems) GENLIN was faster than IPOPT (up to a tolerance of 10%) 37 times and IPOPT was faster than GENLIN (up to a tolerance of 10%) 15 times (both methods spent the same amount of time in 2 cases).
5. Restricting the item above to the problems in which one of the methods used at least 0.1 seconds (26 problems) GENLIN was faster than IPOPT 11 times and IPOPT was faster than GENLIN 14 times (both methods spent the same time in 2 cases).

A performance profile using the number of functional evaluations as a performance measurement can be seen in Figure 5. As in the case of MINOS, we recall that time comparisons have little significance in this case since IPOPT uses sparse matrix techniques for linear algebra computations.

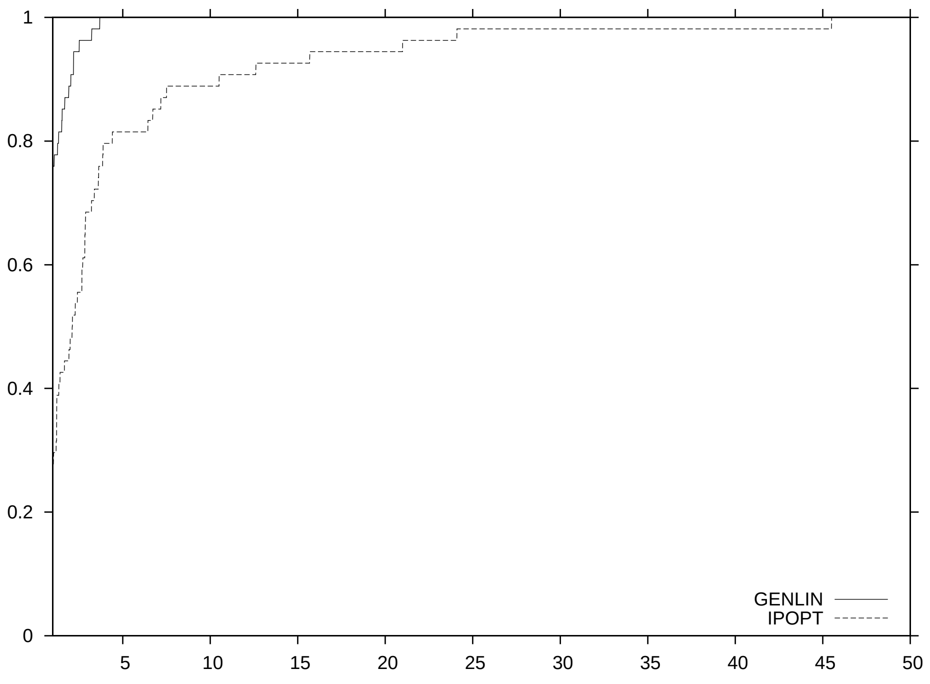

Figure 5: Performance profiles of GENLIN and IPOPT in the 95 problems (out of 127) for which both methods obtained feasible points with equivalent functional values. Number of functional evaluations is being used as performance measurement.

6.6 Conclusions of the Numerical Experiments

One should be very cautious when stating conclusions from very preliminar numerical experience. It must be warned that conclusions are restricted to the particular set of problems addressed and that a thorough evaluation comes only after thousands of tests made by other users. We hope that public availability of our codes may encourage some researchers (especially people involved with applications) to use them for solving their own problems. Nevertheless, the following statements seem to be well supported by the performed numerical experiments.

1. GENLIN and BETRALIN have similar behavior both in terms of robustness and efficiency.

2. GENLIN presents a good adequacy of numerical results to theoretical convergence proofs, in the sense that, except in two cases, it stopped satisfying the predicted convergence criterion $\|g_P(x^k, \delta^k)\|_\infty \leq \varepsilon$. The exceptions were a problem in which the code stopped because the functional values seemed to tend to $-\infty$ and a problem with an undefined objective function at some trial points.

3. We think that the most interesting positive observation coming from comparative experiments is the frequency in which GENLIN obtains solutions using less function evaluations than the competitors. In the case of VE11 and MINOS this can be partially attributed to the fact that these codes use quasi-Newton directions, whereas GENLIN employs truncated Newton steps. However, this not the case of IPOPT, which uses Newtonian search directions. Therefore, the efficiency in terms of functional evaluations seems to rely on the strategy for dealing with linear constraints.

7 Final Remarks

As so happens to occur with unconstrained and box-constrained optimization, improvements in the techniques for minimizing smooth functions with linear constraints may cause almost immediate benefical effects in other practical optimization areas. Penalty and Augmented Lagrangian methods may solve problems with linear and nonlinear constraints placing the linear ones in a lower level and reserving the penalty-dual approach only for the nonlinear constraints [3, 21]. Inexact-Restoration techniques [39, 46, 47] and the traditional methods of GRG type [1, 2, 34] use to employ linearly constrained minimization subproblems for their general nonlinearly constrained calculations, and many other examples can be found in the literature involving nontraditional optimization-like problems. The potentially multiplicative aspect of this research is, therefore, one of the reasons for pursuing increasingly efficient algorithms for linearly constrained optimization. The other reason is, of course, that many Physical, Engineering and Economic problems admit linearly constrained optimization as natural mathematical models.

In this paper we made the first practical and computational attempt to incorporate the well established SPG ideas to an efficient software dedicated to optimization on polytopes without special structure. Although our final goal is to address efficiently large-scale problems, we felt the necessity of producing an acceptable piece of software for small to moderate size problems. Many algorithmic decisions were taken in the process of developing this software and some crucial features of the algorithm were introduced during the testing period. As a whole, we think that the results were pleasantly good although, as mentioned in Section 6, only thousands of tests performed by other users can confirm this initial feeling. With this purpose, we made our software available through the TANGO Project web page *http://www.ime.usp.br/∼egbirgin/tango/* (alternatively, follow the direct link *http://www.ime.usp.br/∼egbirgin/sources/genlin/*) and we hope that it is clear and friendly enough to be used by people having real-life problems. Software for large-scale problems, with sparse technology, will be released soon, although we expect that new algorithmic decisions will be necessary for achieving robustness and efficiency in this case.

References

[1] J. Abadie, Modification of the GRG method, *RAIRO Operations Research* 3, pp. 323–326, 1979.

[2] J. Abadie and G. Guerrero, The General Reduced Gradient Method (GRG), the global Newton Method and their application to Mathematical Programming, *RAIRO Operations Research* 18, pp. 319–351, 1984.

[3] R. Andreani, E. G. Birgin, J. M. Martínez and M. L. Schuverdt, On Augmented Lagrangian methods with general lower-level constraints, *SIAM Journal on Optimization* 18, pp. 1286–1309, 2007.

[4] R. Andreani, E. G. Birgin, J. M. Martínez, and M. L. Schuverdt, Augmented Lagrangian methods under the Constant Positive Linear Dependence constraint qualification, *Mathematical Programming* 111, pp. 5–32, 2008.

[5] R. Andreani, E. G. Birgin, J. M. Martínez, and J-Y. Yuan, Spectral projected gradient and variable metric methods for optimization with linear inequalities, *IMA Journal of Numerical Analysis* 25 pp. 221–252, 2005.

[6] M. Andretta, E. G. Birgin and J. M. Martínez, Practical active-set Euclidian trust-region method with spectral projected gradients for bound-constrained minimization, *Optimization* 54, pp. 305–325, 2005.

[7] M. Andretta, E. G. Birgin and J. M. Martínez, Partial Spectral Projected Gradient Method with Active-Set Strategy for Linearly Constrained Optimization, Technical Report MCDO090309 (see http://www.ime.usp.br/∼egbirgin/), Department of Applied Mathematics, UNICAMP, Brazil, 2009.

[8] J. Barzilai and J. M. Borwein, Two point step size gradient method, *IMA Journal of Numerical Analysis* 8, pp. 141–148, 1988.

[9] L. Bello and M. Raydan, Convex constrained optimization for the seismic reflection tomography problem, *Journal of Applied Geophysics* 62, pp. 158–166, 2007.

[10] D. P. Bertsekas, On the Goldstein-Levitin-Polyak gradient projection method, *IEEE Transactions on Automatic Control* 21, pp. 174–184, 1976.

[11] E. G. Birgin, R. Biloti, M. Tygel, and L. T. Santos, Restricted optimization: a clue to a fast and accurate implementation of the Common Reflection Surface method, *Journal of Applied Geophysics* 42, pp. 143–155, 1999.

[12] E. G. Birgin, R. Castillo, and J. M. Martínez, Numerical comparison of Augmented Lagrangian algorithms for nonconvex problems, *Computational Optimization and Applications* 31, pp. 31–56, 2005.

[13] E. G. Birgin and Y. G. Evtushenko, Automatic differentiation and spectral projected gradient methods for optimal control problems, *Optimization Methods and Software* 10, pp. 125–146, 1998.

[14] E. G. Birgin and J. M. Martínez, A box-constrained optimization algorithm with negative curvature directions and spectral projected gradients, *Computing [Suppl]* 15, pp. 49–60, 2001.

[15] E. G. Birgin and J. M. Martínez, Large-scale active-set box-constrained optimization method with spectral projected gradients, *Computational Optimization and Applications* 23, pp. 101–125, 2002.

[16] E. G. Birgin and J. M. Martínez, Structured minimal-memory inexact quasi-Newton method and secant preconditioners for Augmented Lagrangian Optimization, *Computational Optimization and Applications* 39, pp. 1–16, 2008.

[17] E. G. Birgin, J. M. Martínez, and M. Raydan, Nonmonotone spectral projected gradient methods on convex sets, *SIAM Journal on Optimization* 10, pp. 1196–1211, 2000.

[18] E. G. Birgin, J. M. Martínez, and M. Raydan, Algorithm 813: SPG - Software for convex-constrained optimization, *ACM Transactions on Mathematical Software* 27, pp. 340–349, 2001.

[19] E. G. Birgin, J. M. Martínez, and M. Raydan, Inexact Spectral Projected Gradient methods on convex sets, *IMA Journal on Numerical Analysis* 23, pp. 539–559, 2003.

[20] I. Bongartz, A. R. Conn, N. I. M. Gould, and Ph. L. Toint, CUTE: constrained and unconstrained testing environment, *ACM Transactions on Mathematical Software* 21, pp. 123–160, 1995.

[21] A. R. Conn, N. I. M. Gould, A. Sartenaer, and Ph. L. Toint, Convergence properties of an Augmented Lagrangian algorithm for optimization with a combination of general equality and linear constraints, *SIAM Journal on Optimization* 6, pp. 674–703, 1996.

[22] D. Cores and M. Loreto, A generalized two-point ellipsoidal anisotropic ray tracing for converted waves, *Optimization and Engineering* 8, pp. 373–396, 2007.

[23] Y. H. Dai, Alternate step gradient method, *Optimization* 52, pp. 395–415, 2003.

[24] Y. H. Dai and R. Fletcher, Projected Barzilai-Borwein methods for large-scale box-constrained quadratic programming, *Numerische Mathematik* 100, pp. 21–47, 2005.

[25] Y. H. Dai and R. Fletcher, On the asymptotic behaviour of some new gradient methods, *Mathematical Proramming* 103 pp. 541–559, 2005.

[26] Y. H. Dai and R. Fletcher, New algorithms for single linearly constrained quadratic programs subject to lower and upper bounds, *Mathematical Proramming* 106 pp. 403–421, 2005.

[27] Y. H. Dai, W. W. Hager, K. Schittkowski, and H. C. Zhang, The cyclic Barzilai-Borwein method for unconstrained optimization, *IMA Journal of Numerical Analysis* 26, pp. 604–627, 2006.

[28] Y. H. Dai and H. C. Zhang, Adaptive two-point stepsize gradient algorithm, *Numerical Algorithms* 27, pp. 377–385, 2001.

[29] G. P. Deidda, E. Bonomi, and C. Manzi, Inversion of electrical conductivity data with Tikhonov regularization approach: some considerations, *Annals of Geophysics* 46, pp. 549–558, 2003.

[30] M. A. Diniz-Ehrhardt, M. A. Gomes-Ruggiero, J. M. Martínez, and S. A. Santos, Augmented Lagrangian algorithms based on the spectral projected gradient for solving nonlinear programming problems. *Journal of Optimization Theory and Applications* 123, pp. 497–517, 2004.

[31] E. D. Dolan and J. J. Moré, Benchmarking optimization software with performance profiles, *Mathematical Programming* 91, pp. 101–213, 2002.

[32] R. Fletcher, On the Barzilai-Borwein method, Department of Mathematics, University of Dundee, NA/207, Dundee, Scotland, 2001.

[33] A. Friedlander, J. M. Martínez, B. Molina, and M. Raydan, Gradient method with retards and generalizations, *SIAM Journal on Numerical Analysis* 36, pp. 275–289, 1998.

[34] D. Gabay and D. G. Luenberger, Efficiently converging minimization methods based on reduced gradient, *SIAM Journal on Control* 14, pp. 42–61, 1976.

[35] P. E. Gill, G. H. Golub, W. Murray, and M. A. Saunders, Methods for modifying matrix factorizations, *Mathematics of Computation* 28, pp. 505–535, 1974.

[36] D. Goldfarb and A. Idnani, A numerically stable dual method for solving strictly convex quadratic programs, *Mathematical Programming* 27, pp. 1–33, 1983.

[37] A. A. Goldstein, Convex Programming in Hilbert Space, *Bulletin of the American Mathematical Society* 70, pp. 709–710, 1964.

[38] M. A. Gomes-Ruggiero, J. M. Martínez, and S. A. Santos, Spectral Projected Gradient Method with Inexact Restoration for Minimization with Nonconvex Constraints, *SIAM Journal on Scientific Computing* 31, pp. 1628–1652, 2009.

[39] C. C. Gonzaga, E. Karas, and M. Vanti, A globally convergent filter method for nonlinear programming, *SIAM Journal on Optimization* 14, pp. 646–669, 2003.

[40] L. Grippo, F. Lampariello, and S. Lucidi, A nonmonotone line search technique for Newton's method, *SIAM Journal on Numerical Analysis* 23, pp. 707–716, 1986.

[41] L. Grippo and M. Sciandrone, Nonmonotone globalization techniques for the Barzilai-Borwein gradient method, *Computational Optimization and Applications* 23, pp. 143–169, 2002.

[42] L. Grippo and M. Sciandrone, Nonmonotone derivative free methods for nonlinear equations, *Computational Optimization and Applications* 37, pp. 297–328, 2007.

[43] Z. Jiang, Applications of conditional nonlinear optimal perturbation to the study of the stability and sensitivity of the Jovian atmosphere, *Advances in Atmospheric Sciences* 23, pp. 775–783, 2006.

[44] J. J. Júdice, M. Raydan, S. S. Rosa, and S. A. Santos, On the solution of the symmetric eigenvalue complementarity problem by the spectral projected gradient algorithm, *Numerical Algorithms* 47, pp. 391–407, 2008.

[45] E. S. Levitin and B. T. Polyak, Constrained minimization problems, *USSR Computational Mathematics and Mathematical Physics* 6, pp. 1–50, 1966.

[46] J. M. Martínez, Inexact-restoration method with Lagrangian tangent decrease and new merit function for nonlinear programming, *Journal on Optimization Theory and Applications* 111, pp. 39–58, 2001.

[47] J. M. Martínez and E. A. Pilotta, Inexact-restoration algorithm for constrained optimization, *Journal on Optimization Theory and Applications* 104, pp. 135–163, 2000.

[48] J. M. Martínez, E. A. Pilotta, and M. Raydan, Spectral gradient methods for linearly constrained optimization, *Journal of Optimization theory and Application* 125, pp. 629–651, 2005.

[49] B. A. Murtagh and M. A. Saunders, Large-scale linearly constrained optimization, *Mathematical Programming* 14, pp. 41–72, 1978.

[50] M. J. D. Powell, On the quadratic programming algorithm of Goldfarb and Idnani, *Mathematical Programming Study* 25, pp. 46–61, 1985.

[51] M. J. D. Powell, A tolerant algorithm for linearly constrained optimization calculations, *Mathematical Programming* 45, pp. 547–566, 1989.

[52] A. Ramirez-Porras and W. E. Vargas-Castro, Transmission of visible light through oxidized copper films: feasibility of using a spectral projected gradient method, *Applied Optics* 43, pp. 1508–1514, 2004.

[53] M. Raydan, On the Barzilai and Borwein choice of steplength for the gradient method, *IMA Journal of Numerical Analysis* 13, pp. 321–326, 1993.

[54] M. Raydan, The Barzilai and Borwein gradient method for the large scale unconstrained minimization problem, *SIAM Journal on Optimization* 7, pp. 26–33, 1997.

[55] M. Raydan and B. F. Svaiter, Relaxed steepest descent and Cauchy-Barzilai-Borwein method, *Computational Optimization and Applications* 21, pp. 155–167, 2002.

[56] J. B. Rosen, The Gradient Projection method for Nonlinear Programming 2. Linear constraints, *Journal of the Society for Industrial and Applied Mathematics* 8, pp. 181–217, 1960.

[57] K. Schittkowski, *QL: a Fortran code for convex quadratic programming*, User's Guide, Version 2.1, 2004.

[58] T. Serafini, G. Zanghirati, and L. Zanni, Gradient projection methods for quadratic programs and applications in training support vector machines, *Optimization Methods and Software* 20, pp. 347–372, 2005.

[59] Ph. L. Toint, An assessment of non-monotone linesearch techniques for unconstrained optimization, *SIAM Journal on Scientific Computing* 17, pp. 725–739, 1996.

[60] W. E. Vargas, D. E. Azofeifa, and N. Clark, Retrieved optical properties of thin films on absorbing substrates from transmittance measurements by application of a spectral projected gradient method, *Thin Solid Films* 425, pp. 1–8, 2003.

[61] A. Wächter and L. T. Biegler, On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming, *Mathematical Programming* 106, pp. 25–57, 2006.

[62] C. Y. Wang and Q. Liu, Convergence properties of inexact projected gradient methods, *Optimization* 55, pp. 301–310, 2006.

[63] C. Y. Wang, Q. Liu, and X. M. Yang, Convergence properties of nonmonotone spectral projected gradient methods, *Journal of Computational and Applied Mathematics* 182, pp. 51–66, 2005.

[64] Y-X. Yuan, A new stepsize for the steepest descent method, *Journal of Computational Mathematics* 24, pp. 149–156, 2006.

[65] N. Zeev, O. Savasta, and D. Cores, Non-monotone spectral projected gradient method applied to full waveform inversion *Geophysical Prospecting* 54, pp. 525–534, 2006.

[66] H. Zhang and W. W. Hager, A nonmonotone line search technique and its application to unconstrained optimization, *SIAM Journal on Optimization* 14, pp. 1043–1056, 2004.

[67] L. Zhang and W. J. Zhou, Spectral gradient projection method for solving nonlinear monotone equations, *Journal of Computational and Applied Mathematics* 196, pp. 478–484, 2006.