SPG: Software for Convex-Constrained Optimization

Ernesto G. Birgin * José Mario Martínez † Marcos Raydan ‡

February 13, 2001

Abstract

Fortran 77 software implementing the SPG method is introduced. SPG is a nonmonotone projected gradient algorithm for solving large-scale convex-constrained optimization problems. It combines the classical projected gradient method with the spectral gradient choice of steplength and a nonmonotone line search strategy. The user provides objective function and gradient values, and projections onto the feasible set. Implementation details are presented and the usage of the software is described. Some recent numerical tests are reported on very large location problems. The main conclusion is that SPG compares favorably with existing software.

Categories and Subject Descriptors:
D.3.2 [**Programming Languages**]: Language Classifications - *Fortran 77*; G.1.6 [**Numerical Analysis**]: Optimization - *gradient methods*; G.4 [**Mathematics of Computing**]: Mathematical Software

General Terms: Algorithms

Additional Key Words and Phrases: Projected gradients, nonmonotone line search, large-scale problems, bound constrained problems,

*Department of Computer Science, Institute of Mathematics and Statistics, University of São Paulo, Rua do Matão 1010 Cidade Universitária, 05508-900 São Paulo, SP - Brazil (egbirgin@ime.usp.br). Sponsored by FAPESP (Grant 99/08029-9) and CNPq.

†Departamento de Matemática Aplicada, IMECC-UNICAMP, CP 6065, 13081-970 Campinas SP, Brazil (martinez@ime.unicamp.br). Sponsored by FAPESP (Grant 90/3724-6), CNPq and FAEP-UNICAMP.

‡Departamento de Computación, Facultad de Ciencias, Universidad Central de Venezuela, Ap. 47002, Caracas 1041-A, Venezuela (mraydan@reacciun.ve). Sponsored by the Center of Scientific Computing at UCV.

spectral gradient method.

1 Introduction

In this paper we describe Fortran 77 software that implements the nonmonotone spectral projected gradient (SPG) algorithm. The SPG method applies to problems of the form

$$\min f(x) \text{ subject to } x \in \Omega,$$

where Ω is a closed convex set in $\mathbb{R}^n$. It is assumed that f is defined and has continuous partial derivatives on an open set that contains Ω. Users of the software must supply subroutines to compute the function $f(x)$, the gradient $\nabla f(x)$ and projections of an arbitrary point x onto Ω. Information about the Hessian matrix is not required and the storage requirements are minimal. Therefore, the algorithm is appropriate for large-scale convex-constrained optimization problems with affordable projections onto the feasible set. Notice that the algorithm is also suitable for unconstrained optimization problems simply by setting $\Omega = \mathbb{R}^n$.

The algorithm is fully described in [6] and combines the projected gradient method [2] with two new features in optimization. First, it extends the typical globalization strategies associated with these methods to the nonmonotone line search scheme developed by Grippo, Lampariello and Lucidi [12]. Second, it uses the spectral steplength, introduced by Barzilai and Borwein [1] and analyzed by Raydan [21]. This choice of steplength requires little computational work and greatly speeds up the convergence of gradient methods for unconstrained problems [22].

It is worth noting that two different versions of the projected gradient method were considered in [6]. The new features were applied to the classical curvilinear path (piecewise linear if Ω is a polyhedral set) to introduce Algorithm SPG1. They were also applied to the feasible continuous projected path to produce Algorithm SPG2. Based on numerical experimentation reported in [6], we concluded that there are no meaningful differences between the performances of SPG1 and SPG2. Therefore, since SPG1 tends to require more projections onto Ω per iteration, we only consider the SPG2 version of the method in the software presented and described in this paper.

2 Algorithm

Given $\hat{x} \in I\!R^n$ we define $P_\Omega(\hat{x})$ to be the projection with respect to a given norm $\|\cdot\|$ onto Ω, i.e., $P_\Omega(\hat{x}) = \arg\min_{x\in\Omega} \|x-\hat{x}\|$. We denote $g(x) = \nabla f(x)$. The algorithm starts with $x_0 \in I\!R^n$ and uses an integer $m \geq 1$; a small parameter $\alpha_{\min} > 0$; a large parameter $\alpha_{\max} > \alpha_{\min}$; a sufficient decrease parameter $\gamma \in (0,1)$; and safeguarding parameters $0 < \sigma_1 < \sigma_2 < 1$. Initially, $\alpha_0 \in [\alpha_{\min}, \alpha_{\max}]$ is arbitrary.

Algorithm SPG

`Set` $k \leftarrow 0$. `If` $x_0 \notin \Omega$, `replace` x_0 `by` $P(x_0)$.

`While` (the stopping criterion is not satisfied) `do`

> `Compute` $d_k = P(x_k - \alpha_k g(x_k)) - x_k$, λ_k using the line search algorithm described below `and` $x_{k+1} = x_k + \lambda_k d_k$.
>
> `Compute` $s_k = x_{k+1} - x_k$, $y_k = g(x_{k+1}) - g(x_k)$ `and` $\beta_k = \langle s_k, y_k \rangle$.
>
> `If` $(\beta_k \leq 0)$ `set` $\alpha_{k+1} \leftarrow \alpha_{\max}$
> `else compute` $\alpha_{k+1} = \min\{\alpha_{\max}, \max\{\alpha_{\min}, \langle s_k, s_k\rangle/\beta_k\}\}$.
>
> `Set` $k \leftarrow k+1$.

`end while`

`Set` $x_* \leftarrow x_k$.

This algorithm is based on the spectral projected gradient direction $P(x_k - \alpha_k g(x_k)) - x_k$, where α_k is the safeguarded "inverse Rayleigh quotient" $\frac{\langle s_{k-1}, s_{k-1}\rangle}{\langle s_{k-1}, y_{k-1}\rangle}$. (Observe that $\frac{\langle s_{k-1}, y_{k-1}\rangle}{\langle s_{k-1}, s_{k-1}\rangle}$ is a Rayleigh quotient corresponding to the average Hessian matrix $\int_0^1 \nabla^2 f(x_{k-1} + t s_{k-1}) dt$.)

The line search is based on a safeguarded quadratic interpolation. The safeguarding procedure acts when the minimum of the one-dimensional quadratic lies outside $[\sigma_1, \sigma_2\lambda]$, and not when it lies outside $[\sigma_1\lambda, \sigma_2\lambda]$ as usually implemented. This means that, when interpolation tends to reject 90% (for $\sigma_1 = 0.1$) of the original search interval ($[0,1]$), we judge that its prediction is not reliable and we prefer the more conservative bisection. This procedure turned out to be more efficient than the classical one. The complete line search procedure is described below.

Line search

`Compute` $f_{\max} = \max\{f(x_{k-j}) \mid 0 \leq j \leq \min\{k, m-1\}\}$, $x_+ \leftarrow x_k + d_k$, $\delta \leftarrow \langle g(x_k), d_k\rangle$ `and set` $\lambda \leftarrow 1$.

```
While (f(x+) > fmax + γλδ) do
        Compute λtemp ← −½λ²δ/(f(x+) − f(xk) − λδ).
        If (λtemp ≥ σ1 and λtemp ≤ σ2λ) set λ ← λtemp
        else set λ ← λ/2.
        Compute x+ ← xk + λdk.
end while
Set λk ← λ.
```

In the case of rejection of the first trial point, the next ones are computed along the same direction. As a consequence, the projection operation is performed only once per iteration.

We use the convergence criteria given by

$$\|P(x_k - g(x_k)) - x_k\|_\infty \le \epsilon_1 \tag{1}$$

or

$$\|P(x_k - g(x_k)) - x_k\|_2 \le \epsilon_2. \tag{2}$$

In addition, the algorithm is also stopped when the number of iterations is larger than *maxit* or the number of functional evaluations exceeds *maxfun*.

In the experiments presented in Section 5, we chose $\gamma = 10^{-4}$, $\sigma_1 = 0.1$, $\sigma_2 = 0.9$, $m = 10$, $\alpha_{\min} = 10^{-3}$, $\alpha_{\max} = 10^3$, $\alpha_0 = 1$, $\epsilon_1 = 0$, $\epsilon_2 = 10^{-6}$, $maxit = 1000$, $maxfun = 2000$. The stopping criterion (1) was inhibited in order to encourage a fair comparison with an alternative algorithm.

The choice of $\alpha_{\min}$, $\alpha_{\max}$, α_0, ϵ_1 and ϵ_2 is sensitive because these are dimensional parameters. Roughly speaking, the adopted choice is adequate if the problem is scaled in such a way that the Hessian eigenvalues are $O(1)$.

3 Usage of the package

The call statement for SPG is:

$$\texttt{call spg(n}^I\texttt{,x}^{IO}\texttt{,m}^I\texttt{,eps1}^I\texttt{,eps2}^I\texttt{,maxit}^I\texttt{, maxfun}^I\texttt{,output}^I\texttt{,}$$
$$\texttt{+f}^O\texttt{,ginfn}^O\texttt{,gtwon}^O\texttt{,iter}^O\texttt{,nfun}^O\texttt{,ngrad}^O\texttt{,flag}^O\texttt{),}$$

where the superscripts I and O mean *input* and *output*, respectively.

The user must supply the external subroutines `EvalF`, `EvalG` and `Proj` to evaluate the objective function and its gradient and to project an arbitrary point onto the feasible region. Subroutine `Proj` depends on the description of the feasible region. For the case of boxes it could be

```
subroutine Proj(n, x, l, u)
integer n
double precision x(n), l(n), u(n)

do i = 1, n
    x(i) = max(l(i),min(x(i),u(i))
end do

end
```

Each gradient evaluation is necessarily preceeded by a functional evaluation at the same point. For this reason, the user can take advantage of possible common expressions or tests in his/her gradient subroutine implementation.

4 Test problems

In [6] we showed the performance of SPG on a set of large-scale box-constrained problems from the CUTE collection [7] and compared it against LANCELOT [9]. The main conclusion of those tests is that SPG is competitive for box-constrained problems. However, the SPG theory allows one to deal with convex constraints, so it is natural to test the method in more general situations. The only restriction is that the projection onto the feasible region must be easy (affordable) to compute. In our software, we leave this task to the user-given subroutine `Proj`. Since most general nonlinear programming algorithms do not take explicit advantage of the possible simplicity of projections, our feeling is that SPG could outperform general NLP solvers in that case.

Here we will consider a family of *location* problems. Given a set of $npol$ disjoint polygons in $I\!R^2$ we wish to find the point y that minimizes the sum of the distances to those polygons. Therefore, the problem is

$$\min_{z^i,y} \sum_{i=1}^{npol} \|z^i - y\|_2$$

$$\text{subject to} \quad z^i \in P_i,\ i = 1, \ldots, npol.$$

Let us write $x = (z_1^1, z_2^1, \ldots, z_1^{npol}, z_2^{npol}, y_1, y_2)$. We observe that the problem has $2 \times (npol + 1)$ variables. The number of (linear inequality) constraints is $\sum_{i=1}^{npol} \nu_i$, where ν_i is the number of vertices of the polygon P_i. Each constraint defines a half-plane in $I\!R^2$.

We generated 45 problems of this class, varying *npol* and choosing randomly the localization of the polygons and the number of vertices of each one. The details of the generation, including the way in which we guarantee empty intersections, are rather tedious but, of course, are available for interested readers. In Table 1 we display the main characteristics of each problem (number of polygons, number of constraints and dimension of the problem). Figure 1 shows the solution of a small five-polygons problem.

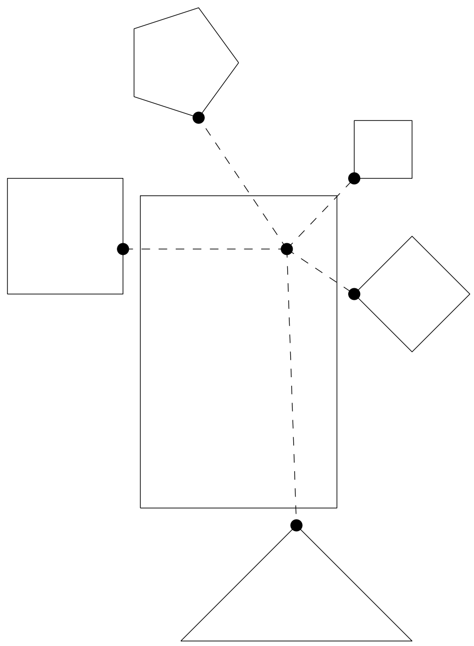

Figure 1: Five-polygons problem.

For projecting x onto the feasible set observe that we only need to project each z^i separately onto the corresponding polygon P_i. In the projection subroutine we consider the half-planes that define the polygon. If z^i belongs to all these half-planes, then z^i is the projection onto P_i. Otherwise, we consider the set of half-planes to which z^i does not belong. We project z^i onto these half-planes and we discard the projected points that do not belong to P_i. Let A_i be the (finite) set of nondiscarded half-plane projections and let V_i be the set of vertices of P_i. Then, the projection of z^i onto P_i is the point of $A_i \cup V_i$ that is closest to z^i. Both the subroutine for generating the problems and the projection subroutine are included in the test driver for SPG method.

The test problems were solved both with SPG and with FSQP, the general nonlinear programming solver described in [28]. See also [8, 14, 19, 27]. This is a sequential quadratic programming algorithm that uses feasible iterates and has good global and local convergence properties. For running FSQP we used the default parameters indicated in the code documentation [27].

For both methods, we used the origin as initial approximation. Both algorithms found the solutions of the problems and stopped with the diagnostic of convergence. The quality of the solutions was always the same.

Tables 2 and 3 display the performance of SPG and FSQP, respectively. The columns mean: *Problem*, problem number; n, dimension of the problem; *iter*, iterations needed to reach the solution; *fe*, function evaluations; *Time*, CPU time (seconds); *f*, function value at the solution.

All the experiments were run on a Sun SparcStation 20 with the following main characteristics: 128Mbytes of RAM, 70MHz, 204.7 mips, 44.4 Mflops. Codes are in Fortran77 and the compiler option adopted was "-O".

We ran only 15 problems using FSQP because, for larger problems, the computation time for this algorithm becomes unaffordable. One should be careful when looking at the time for this algorithm because the implementation FSQP of the feasible sequential quadratic programming algorithm does not take advantage of sparsity of the matrix of constraints or the Hessian of the objective function. An implementation using sparse factorizations would be much more efficient in terms of computer time. However, we decided to maintain the comparison against FSQP because the results in terms of number of iterations and function evaluations would not change (at least, certainly, could not be better) in a sparse implementation of the method.

Clearly, independently of the linear-algebra savings of sequential quadratic programming (SQP) implementations, any SQP iteration is (much) more expensive than a single iteration of the algorithm described in this paper. So, it is remarkable that the number of iterations used by our algorithm is from 8 to 20 times smaller than the number of iterations used by FSQP. Approximately the same relation exists between the number of function and gradient evaluations used by both methods. Therefore, the advantage of solving the

Problem	# polygons	# constraints	n
1	86	343	174
2	86	700	174
3	85	1018	172
4	174	701	350
5	181	1450	364
6	176	2104	354
7	257	1033	516
8	256	2103	514
9	262	3216	526
10	359	1437	720
11	349	2854	700
12	349	4328	700
13	435	1743	872
14	432	3522	866
15	430	5257	862
16	935	3729	1872
17	928	7399	1858
18	940	11200	1882
19	1841	7362	3684
20	1825	14574	3652
21	1841	22049	3684
22	2766	11102	5534
23	2795	22511	5592
24	2831	34445	5664
25	3779	15104	7560
26	3815	30699	7632
27	3852	46704	7706
28	4735	18937	9472
29	4767	38247	9536
30	4836	58622	9674
31	9680	38790	19362
32	9644	77413	19290
33	9639	116195	19280
34	19117	76550	38236
35	19093	153156	38188
36	19114	230190	38230
37	28770	115301	57542
38	28799	230878	57600
39	28767	346048	57536
40	38409	153784	76820
41	38568	309252	77138
42	38506	463718	77014
43	48004	192152	96010
44	48209	386121	96420
45	48126	578648	96254

Table 1: Problems set.

Problem	iter	fe	Time	f
1	14	15	0.01	1.853D+02
2	14	15	0.01	1.959D+02
3	14	15	0.02	1.914D+02
4	24	27	0.03	1.945D+02
5	23	26	0.05	2.021D+02
6	20	21	0.05	1.948D+02
7	19	20	0.04	1.952D+02
8	24	25	0.07	1.989D+02
9	34	36	0.13	1.948D+02
10	19	20	0.05	1.969D+02
11	29	30	0.11	1.969D+02
12	23	24	0.11	1.943D+02
13	14	15	0.05	1.968D+02
14	25	26	0.11	1.983D+02
15	26	29	0.16	1.974D+02
16	20	21	0.14	1.286D+03
17	30	33	0.30	1.324D+03
18	28	30	0.36	1.304D+03
19	65	78	0.86	1.311D+03
20	51	63	0.98	1.315D+03
21	42	47	1.06	1.293D+03
22	33	35	0.67	1.308D+03
23	55	67	1.62	1.266D+03
24	80	117	3.20	1.312D+03
25	69	103	1.93	1.311D+03
26	50	60	2.05	1.301D+03
27	37	42	2.03	1.321D+03
28	144	219	4.99	1.305D+03
29	50	60	2.57	1.306D+03
30	44	47	3.20	1.309D+03
31	18	20	1.44	1.968D+03
32	24	30	2.89	1.966D+03
33	20	21	3.24	1.959D+03
34	14	15	2.35	1.968D+03
35	15	17	3.67	1.964D+03
36	19	21	6.00	1.973D+03
37	14	15	3.58	1.973D+03
38	18	20	6.49	1.974D+03
39	13	14	6.25	1.974D+03
40	14	15	4.82	1.975D+03
41	25	28	12.19	1.980D+03
42	24	26	15.01	1.980D+03
43	18	20	7.49	1.973D+03
44	13	14	7.99	1.978D+03
45	17	19	12.97	1.980D+03

Table 2: SPG performance.

Problem	iter	fe	Time	f
1	114	114	142.30	1.853D+02
2	147	147	464.93	1.959D+02
3	151	151	844.37	1.914D+02
4	220	220	3720.98	1.945D+02
5	304	304	12817.44	2.021D+02
6	278	278	15758.56	1.948D+02
7	282	282	18787.54	1.952D+02
8	390	390	49819.97	1.989D+02
9	412	412	81976.09	1.948D+02
10	325	325	89058.47	1.969D+02
11	395	395	126319.88	1.969D+02
12	371	371	171442.94	1.943D+02
13	438	438	149931.47	1.968D+02
14	507	507	316525.41	1.983D+02
15	524	524	465043.66	1.974D+02

Table 3: FSQP performance.

problems using SPG in place of a general nonlinear programming algorithm is quite impressive.

5 Conclusions

We have presented a computational algorithm for minimizing functions of many variables restricted to a convex set. The algorithm tends to work well when projections onto the feasible set are easy to compute. The user is required to provide a subroutine to compute these projections. A previous paper [6] shows that the new algorithm is effective for solving many large-scale box-constrained problems. Besides the tests of [6], SPG has been shown to be efficient in several applied box-constrained problems. See [3, 4, 5, 18]. Here we show that, perhaps, this effectiveness is even more evident when the constraints are given in some other form, provided that projections are not complicated. The key point is that most general nonlinear programming algorithms do not take advantage of the easy-projection property at all.

An interesting family of problems to which SPG can be applied is the norm-constrained regularization problem [13, 16, 17, 26], defined by

$$\min f(x) \text{ subject to } x^T Ax \leq r \tag{3}$$

where A is symmetric positive definite. This problem can be reduced to ball-constrained minimization by a change of variables and, in this case, projections can be trivially computed. A particular case of (3) is the classical trust-region problem where f is quadratic. Recently (see [15, 20]) procedures for escaping from nonglobal stationary points of this problem have been found and, so, it becomes increasingly important to obtain fast algorithms for finding critical points, especially in the large-scale case. See [23, 24, 25].

Acknowledgement

The authors are indebted to professors Tim Hopkins and Michael Saunders, whose comments helped a lot to improve this work.

References

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

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

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

[4] Birgin, E.G., Chambouleyron, I., and Martínez, J.M. 1999. Estimation of the optical constants and the thickness of thin films using unconstrained optimization. *J. of Computational Physics 151*, 862–880.

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

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

[7] Bongartz, I., Conn, A.R., Gould, N.I.M., and Toint, Ph.L. 1995. CUTE: Constrained and Unconstrained Testing Environment. *ACM Transactions on Mathematical Software 21*, 123–160.

[8] Bonnans, J.F., Panier, E., Tits, A., and Zhou, J. 1992. Avoiding the Maratos effect by means of a nonmonotone line search: II. Inequality Problems - Feasible Iterates, *SIAM J. on Numerical Analysis 29*, 1187–1202.

[9] Conn, A.R., Gould, N.I.M., and Toint, Ph.L. 1988. Global convergence of a class of trust region algorithms for optimization with simple bounds. *SIAM J. on Numerical Analysis 25*, 433–460. See also *SIAM J. on Numerical Analysis 26*, 764–767.

[10] Conn, A.R., Gould, N.I.M., and Toint, Ph.L. 1991. A globally convergent augmented Lagrangean algorithm for optimization with general constraints and simple bounds. *SIAM J. on Numerical Analysis 28*, 545–572.

[11] Friedlander, A., Martínez, J.M., and Santos, S.A. 1994. A new trust region algorithm for bound constrained minimization. *Applied Mathematics and Optimization 30*, 235–266.

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

[13] Heinkenschloss, M. 1993. Mesh independence for nonlinear least squares problems with norm constraints. *SIAM J. on Optimization 3*, 81–117.

[14] Lawrence, C.T., and Tits, A.L. 1996. Nonlinear equality constraints in feasible sequential quadratic programming. *Optimization Methods and Software 6*, 265–282.

[15] Lucidi, S., Palagi, L., and Roma, M. 1998. On some properties of quadratic programs with a convex constraint. *SIAM J. on Optimization 8*, 105–122.

[16] Martínez, J.M., and Santos, S.A. 1995. A trust region strategy for minimization on arbitrary domains. *Mathematical Programming 68*, 267–302.

[17] Martínez, J.M., and Santos, S.A. 1997. Convergence results on an algorithm for norm constrained regularization and related problems. *RAIRO Operations Research 31*, 269–294.

[18] Mulato, M., Chambouleyron, I., Birgin, E.G., and Martínez, J.M. 2000. Determination of thickness and optical constants of a-Si:H films from transmittance data, *Applied Physics Letters 77*, 2133–2135.

[19] Panier, E., and Tits, A. 1993. On combining feasibility, descent and superlinear convergence in inequality constrained optimization. *Mathematical Programming 59*, 261–276.

[20] Pham Dinh, T., and Hoai An, L.T. 1998. A D.C. optimization algorithm for solving the trust region problem. *SIAM J. on Optimization 8*, 476–505.

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

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

[23] Rendl, F., and Wolkowicz, H. 1997. A semidefinite framework to trust region subproblems with application to large scale minimization. *Mathematical Programming 77*, 273–299.

[24] Rojas, M., Santos, S.A., and Sorensen, D.C. 2000. A new matrix-free algorithm for the large-scale trust-region subproblem. *SIAM J. on Optimization 11*, 611–646.

[25] Sorensen, D.C. 1997. Minimization of a large scale quadratic function subject to an ellipsoidal constraint. *SIAM J. on Optimization 7*, 141–161.

[26] Vogel, C.R. 1990. A constrained least-squares regularization method for nonlinear ill-posed problems. *SIAM J. on Control and Optimization 28*, 34–49.

[27] Zhou, J.L., and Tits, A. 1993. Nonmonotone line search for minimax problems. *J. of Optimization Theory and Applications 76*, 455–476.

[28] Zhou, J.L., and Tits, A. 1997. User's Guide for FFSQP Version 3.7: A Fortran code for solving optimization programs, possibly minimax, with general inequality constraints and linear equality constraints, generating feasible iterates. Technical Report SRC-TR-92-107r5, Institute for Systems Research, University of Maryland.