Analisi Numerica: Metodi e Software Matematico

Progetto cofinanziato dal MURST - 1998–2000

A Random Generator for Large–Scale Linearly Constrained Quadratic Programming Test Problems

C. Durazzi V. Ruggiero

Dipartimento di Matematica
Università di Ferrara

L. Zanni

Dipartimento di Matematica
Università di Modena e Reggio Emilia

Monografia n. 6

Ferrara 2000

Monografie del Programma di Ricerca Scientifica 1998–2000
“Analisi Numerica: Metodi e Software Matematico”
cofinanziato dal Ministero per l’Università e
la Ricerca Scientifica e Tecnologica nell’anno 1997.

Monographs of the 1998–2000 Scientific Research Program
“Numerical Analysis: Methods and Mathematical Software”
supported in 1997 by the Italian Ministry for the University
and the Scientific and Technological Research.

http://www.unife.it/AnNum97

Abstract

In this paper we describe a random generator for large and sparse quadratic programming problems that frequently arise in different areas of applied science. This generator is an useful tool in testing algorithms on QP problems with different features, since it allows us to vary many parameters which characterize the problems.

The procedure used to generate a QP problem as well as some details for its implementation are explained.

Finally, we report an analysis of the numerical results, obtained by the routine E04NFK of the NAG library on the test problems produced by the generator.

C. Durazzi V. Ruggiero
Dipartimento di Matematica, Università di Ferrara, Via Machiavelli 35, Ferrara, Italy.
E-mail: {`dzc,rgv`}`@dns.unife.it`.

L. Zanni
Dipartimento di Matematica, Università di Modena e Reggio Emilia, Via Campi 213/b, 41100 Modena, Italy. E-mail: `zanniluca@unimo.it`.

Contents

Chapter 1

Introduction

In many areas of science and technology it is required to solve the following linearly constrained quadratic programming (QP) problem

$$\begin{array}{ll} \textit{minimize} & f(x) = \frac{1}{2}x^TGx + q^Tx \\ \textit{subject to} & Cx = d, \qquad Ax \geq b, \end{array} \tag{1.1}$$

where G is a symmetric matrix of order n, C is an $m_e \times n$ matrix of full row rank ($m_e \leq n$) and A is an $m_i \times n$ matrix.

At the moment, attention is greatly focused on large QP problems where G, C and A may be sparse matrices, without a particular structure. These large problems are basic to many applications of data analysis, such as image restoration [1], pattern recognition based upon support vector machine technique [3] and constrained bivariate interpolation [5]. Furthermore, large QP problems arise as subproblems of the class of symmetric cost approximation methods [13] for large–scale nonlinear programming or variational inequality problems when the auxiliary function is a convex quadratic function or the gradient map of a convex quadratic function. See, for example, the sequential quadratic programming method or, for the variational inequalities, the linear approximation methods [12] and the descent methods based upon projective gap functions [14].

In order to make clear the "effectiveness" of the solvers proposed in literature for large QP problems, it is necessary to evaluate the numerical behaviour of these methods by an extensive experimentation on vaste sets of test problems with different features.

Well–known collections of QP problems of the form (1.1) are contained in CUTE [2] (http://www.cse.clrc.ac.uk/Activity/CUTE) and in the repository at the URL http://www.doc.ic.ac.uk/∼im/#DATA [9] (see also the following URL for a list of testcases in optimization: http://plato.la.asu.edu/topics/testcases.html).

The aim of this work is to describe a generator for random, large and sparse quadratic programs. This generator enables us to prefix the most part of the parameters that characterize a problem and, above all, it permits to specify the distribution of the singular values of the matrices of the constraints and the spectrum of the hessian and the reduced hessian of the objective function. Thus we can generate poorly or well–conditioned problems of arbitrary size, number of constraints and level of sparsity. Varying in a convenient way the parameters which the test problems depend on, we can reflect the features of the quadratic programs arising in many practical applications.

In Chapter 2, we describe the technique for randomly generating QP problems with assigned features. In Chapter 3, we report the way of storing the sparse matrices of the QP problem and some details of the implementation of the generator. In Chapter 4, we consider a set of test problems produced by this generator and we show some numerical results obtained by solving the test problems with the routine E04NKF of the NAG library [11].

CHAPTER 2

A procedure for randomly generating a QP problem

We assume that the following data are given:

- the sparsity of the $n \times n$ matrix G ($spars(G)$) and of the $m \times n$ matrix B ($spars(B)$), where $B = \begin{pmatrix} C \\ A \end{pmatrix}$ denotes the constraint matrix ($m = m_i + m_e$); furthermore we set $f = \begin{pmatrix} d \\ b \end{pmatrix}$;
- the rank $r(G)$, the spectral condition number $K(G)$ and the maximum and the minimum nonzero eigenvalues, denoted by $\phi_{max}(G)$ and $\phi_{min}(G)$, of the Hessian matrix G of (1.1); for convex QP problems, the maximum eigenvalue of G is obtained by $\phi_{max}(G) = K(G)\phi_{min}(G)$;
- the spectral condition number $K(B)$ and the minimum nonzero singular value $\sigma_{min}(B)$ of the matrix B; the rank of B is chosen equal to $min(m, n)$ and the maximum singular value of B is $\sigma_{max}(B) = K(B)\sigma_{min}(B)$;
- a solution x^* of (1.1), with entries x_i randomly generated from an uniform distribution in $(-1, 1)$;
- the number m_a of the inequality constraints that are active in x^*; we denote by nac the value $m_a + m_e$, so that $nac \geq m_e$;
- an m-vector $\begin{pmatrix} \mu^* \\ \lambda^* \end{pmatrix}$ of the multipliers associated to x^*, with null elements for the entries corresponding to the inactive inequality constraints in x^*, $\mu_i^* = 10^{-z_i \cdot ndeg}$ for the indices corresponding to the equality constraints and $\lambda_i^* = 10^{-z_i \cdot ndeg}$ for the indices corresponding to the active inequality constraints; here z_i is randomly generated in $(0, 1)$ and $ndeg$ is an integer parameter which specifies the level of degeneracy;
- the rank $r(Z^T G Z)$ and the spectral condition number $K(Z^T G Z)$ of the reduced Hessian $Z^T G Z$, where Z is chosen as an orthogonal basis of the null space of the matrix given by the equality constraints and the inequality constraints active in x^*.

The matrix G of (1.1) is generated by its spectral decomposition

$$G = VDV^T = V \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} V^T \tag{2.1}$$

where D is a diagonal matrix with maximum and minimum nonzero eigenvalues $\phi_{max}(G)$ and $\phi_{min}(G)$ respectively, and $r(G)-2$ diagonal entries randomly generated in $(\phi_{min}(G), \phi_{max}(G))$; in particular, the diagonal submatrix D_2 of order $n-nac$ has $r(Z^TGZ)$ positive diagonal entries chosen so that the condition number of D_2 is $K(Z^TGZ)$.
The matrix V is an orthogonal matrix obtained as product of a number of random Givens rotations R_{ij} such that the matrix G is filled until the prefixed level of sparsity is attained. A random Givens rotation R_{ij} is obtained by randomly generating two different positive integers i and j in the interval $[1, n]$. Then we randomly generate a real number α in the interval $[-1, 1]$ and we set $\alpha = r_{j,j} = r_{i,i} = \cos(\theta)$; consequently, $r_{i,j} = -r_{j,i} = \sqrt{1 - r_{i,i}^2}$.
Now, we partition the matrix V in $\begin{pmatrix} V_1 & V_2 \end{pmatrix}$, where V_1 and V_2 represent the first nac and the last $n - nac$ columns of V respectively.
In order to generate the matrix of constraints B, we construct the $m \times n$ matrix $\begin{pmatrix} \tilde{B} \\ \bar{B} \end{pmatrix}$, where $\tilde{B}$ is the $nac \times n$ matrix of the equality constraints and the active inequality constraints in x^*, and $\bar{B}$ is the $(m - nac) \times n$ matrix composed by the remaining rows of B. We write $\begin{pmatrix} \tilde{B} \\ \bar{B} \end{pmatrix}$ as follows:

$$B = \begin{pmatrix} \tilde{B} \\ \bar{B} \end{pmatrix} = \begin{pmatrix} U_1 & 0 \\ 0 & U_2 \end{pmatrix} \begin{pmatrix} S_1 & 0 \\ 0 & S_2 \end{pmatrix} \begin{pmatrix} V_1^T \\ V_2^T \end{pmatrix} = \begin{pmatrix} U_1S_1V_1^T \\ U_2S_2V_2^T \end{pmatrix} \tag{2.2}$$

where S_1 is a diagonal matrix of order nac with prefixed condition number $K(B)$ and nac positive diagonal entries included in $[\sigma_{min}(B), \sigma_{max}(B)]$; S_2 is an $(m - nac) \times (n - nac)$ matrix with $min(m, n) - nac$ positive diagonal entries randomly generated and such that the condition number of $\begin{pmatrix} \tilde{B} \\ \bar{B} \end{pmatrix}$ is still $K(B)$. The rank of $\begin{pmatrix} \tilde{B} \\ \bar{B} \end{pmatrix}$ is $min(m, n)$.
The matrices U_1 and U_2 are orthogonal matrices of order nac and $m - nac$ respectively, generated as product of random Givens rotations, so that we achieve the prefixed level of sparsity for B in the matrix $\begin{pmatrix} \tilde{B} \\ \bar{B} \end{pmatrix}$.
From (2.2) we deduce that V_2 represents an orthonormal basis for the null space of $\tilde{B}$:

$$\tilde{B}V_2 = U_1S_1V_1^TV_2 = 0$$

Thus, if we put $Z = V_2$, from the following relation

$$V_2^TGV_2 = V_2^T \begin{pmatrix} V_1 & V_2 \end{pmatrix} \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} \begin{pmatrix} V_1^T \\ V_2^T \end{pmatrix}$$

$$= V_2^T(V_1D_1V_1^T + V_2D_2V_2^T)V_2 = D_2$$

we have that the reduced Hessian has rank $r(Z^TGZ)$ and condition number $K(Z^TGZ)$, as prefixed.
Then, we put

$$f = \begin{pmatrix} \tilde{B} \\ \bar{B} \end{pmatrix} x^* - \begin{pmatrix} 0 \\ \epsilon \end{pmatrix}$$

where ϵ is a vector with positive random elements.
Finally, we obtain the n-vector q of the objective function in (1.1) from the Kuhn Tucker conditions:

$$q = -Gx^* + A^T\lambda^* + C^T\mu^*.$$

CHAPTER 3

Implementation of the test problem generator

A random test problem generator for large–scale sparse linearly constrained convex QP problems, based on the technique described in the previous Chapter, is written in Fortran 77.

The source program, contained in the file **test98.f**, is available at the following URL: http://www.unife.it/AnNum97/index2.htm.

An exhaustive documentation, about the input data that the user has to supply and about the output files produced by the program, is reported in the comments of the source file. At the same URL, it is possible to get a file containing an example of the input data (**inputtest98**) and a file showing the messages produced by the program (**outputtest98**).

In the following, we explain some details of the implementation of the test problem generator.

The nonzero eigenvalues of G and the nonzero singular values of B can be randomly generated in a convenient interval, according to a log–uniform distribution or an uniform distribution or they can be equally spaced between the extrema of the interval in question. If the input variable *indg* (or *indb* for the matrix B) is equal to 0, a log–uniform distribution for randomly generating the eigenvalues of G (or the singular values of B) is used; if $indg = 1$ ($indb = 1$, respectively) an uniform distribution is used; otherwise, the eigenvalues of G (or the singular values of B) are equally spaced between the smallest and the largest eigenvalues (or singular values). For the matrices G and B, the user has to supply $condg = \log_{10}(K(G))$, $condb = \log_{10}(K(B))$, $condzgz = \log_{10}(Z^T GZ)$ ($\leq condg$) and $condba = \log_{10}(\tilde{B})$ ($\leq condb$). Furthermore, the following data have to be introduced:

- for the matrix G, the minimum positive eigenvalue *glmin* and the rank *ngrango* (G is a positive semidefinite matrix); the maximum eigenvalue of G is $10^{condg} \cdot glmin$;
- for the matrix $Z^T GZ$, the minimum positive eigenvalue *zgzlmin* ($\geq glmin$) and the rank *nzgzrango* ($\leq ngrango - nac$); the maximum eigenvalue of $Z^T GZ$ is $10^{condzgz} \cdot zgzlmin$;
- the minimum nonzero singular value *blmin* of the matrix B; the rank of B is $\min(m, n)$;
- the minimum nonzero singular value *balmin* ($\geq blmin$) of the matrix $\tilde{B}$; the rank of $\tilde{B}$ is *nac*.

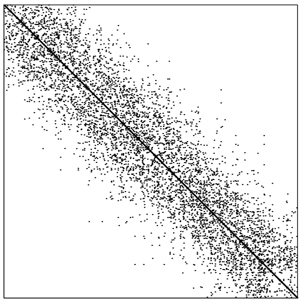

Figure 3.1: Matrix G with a band structure ($n = 2000$, spars(G)=99.8%).

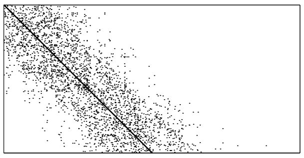

Figure 3.2: Matrix B with a band structure ($m = 1000$, spars(B)=99.8%).

Furthermore, the user has to supply the level of sparsity of G and B. We can also generate band matrices with a prefixed bandwidth (see Figures 1–2).

Finally, we describe the technique used for storing the sparse matrices G and B. Each matrix is stored in compressed form [8]. If we denote by $kmax$ the largest number of nonzero elements per row of a generic sparse $nr \times nc$ matrix A, the storage of A is made in two $nr \times kmax$ matrices AS and IAS as follows:

$$AS(i,1) = A(i,i) \qquad\qquad IAS(i,1) = i$$

$$AS(i,j) = A(i,k) \quad \text{and } IAS(i,j) = k \qquad\quad \text{if } A(i,k) \neq 0$$

An nr vector KAS contains, in the i–th element, the number of nonzero entries of the i-th row of A. For example, when the matrix G is defined as

$$G = \begin{pmatrix} 3 & 0 & 2 \\ 0 & 4 & 1 \\ 2 & 1 & 4 \end{pmatrix} \tag{3.1}$$

the compressed form is given by

$$GS = \begin{pmatrix} 3 & 2 & \\ 4 & 1 & \\ 4 & 2 & 1 \end{pmatrix}, \ KGS = \begin{pmatrix} 2 \\ 2 \\ 3 \end{pmatrix} \text{ and } IGS = \begin{pmatrix} 1 & 3 & \\ 2 & 3 & \\ 3 & 1 & 2 \end{pmatrix} \tag{3.2}$$

The program **test98.f** produces two unformatted files, containing the information about the objective function and the constraints respectively. The nonzero elements of the upper triangular part of the matrix G are stored in a set of consecutive records of the first file, following a row wise ordering; each record contains the row index, the column index and the value of four nonzero elements of G. For example, the matrix G of (3.1)–(3.2) is stored in the first file as:

1 1 3 1 3 2 2 2 4 2 3 1 first record
3 3 4 second record.

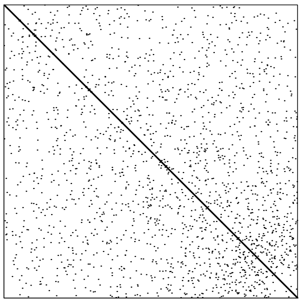

Figure 3.3: Matrix G without structure ($n = 2000$, spars(G)=99.9%).

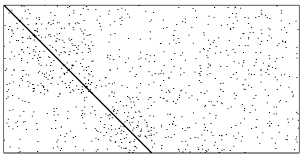

Figure 3.4: Matrix B without structure ($m = 1000$, spars(B)=99.9%).

All the nonzero entries of the matrix B are stored in a set of consecutive records of the second file with the same format used for G in the first file.

CHAPTER 4

Numerical evaluation of the routine E04NKF

In this Chapter we report the numerical results obtained by the routine E04NKF of the NAG library [11] on a set of test problems generated by the technique described in the previous Chapters. All the experiments are carried out on a Digital Alpha 500/333 Mhz workstation, using the double precision (macheps= $2.22 \cdot 10^{-16}$) and a Fortran program, named **test_nag.f**, available at the URL: http://www.unife.it/AnNum97/index2.htm.

This program reads the two unformatted files produced by **test98.f** and rearranges the data according to the format required by E04NKF; then, this routine is called and its results are written in the file nag.dat.

The routine E04NKF is designed to solve large–scale linear and quadratic problems with the following constraints:

$$l \leq \left\{ \begin{array}{c} x \\ Fx \end{array} \right\} \leq u$$

where F is a sparse $m \times n$ matrix. In the experiments reported in this Chapter, we consider convex quadratic problems of the form (1.1), where simple box constraints are not present and we have only general constraints ($F = B$). In this case, the user has to provide a subroutine that computes the matrix–vector product Gx for any given vector x.

The routine E04NKF is based on an active–set method. This is an iterative procedure with two phases: a *feasibility* phase, in which the sum of infeasibilities is minimized to find a feasible point [4, p. 166] and an *optimality* phase in which $f(x)$ is minimized by constructing a sequence of iterations that lies within the feasible region; each iteration requires to solve a QP problem with equality constraints only [4, p. 240]. E04NKF is based on SQOPT, which is part of the SNOPT package [6], that in turn utilizes routines from the MINOS package [10]. It uses stable numerical methods throughout and includes a reliable basis package for maintaining sparse LU factors of a convenient submatrix of the constraint matrix [7].

In the following Tables, *time* denotes the elapsed time in seconds to execute E04NKF, using default setting for all the optional parameters of the routine. The values er_x and er_f denote the relative errors of the computed minimum point $\|x^* - \overline{x}\|_2/\|x^*\|_2$ and of the minimum value of the objective function $|f(x^*) - f(\overline{x})|/|f(x^*)|$ respectively. Here $\overline{x}$ is the approximate solution obtained by E04NKF and x^* is the prefixed solution of the test problem. When $rank(G) < n$, the minimum point is not necessarily unique and er_x is not meaningful. In all the test problems the eigenvalues of the matrix G or the singular values of matrix B are uniformly distributed.

Table 1. Different levels of the sparsity of the constraint matrices

$n = 5000$				
spars$(G) = 99.90\%$ $\quad \|G\|_2 = 1$ $\quad K(G) = 10^4$				
	spars(B)	*time*	er_x	er_f
$m_e = 2500$	99.96	19.7	7.2(-15)	4.0(-16)
$m_i = 0$	99.94	21.9	3.8(-15)	5.2(-16)
$\|B\|_2 = 1$	99.92	22.1	5.4(-15)	1.4(-16)
$K(B) = 10^3$	99.90	28.9	4.1(-14)	4.0(-16)
$m_e = 3500$ $\quad m_i = 1000$	99.93	75.9	7.3(-16)	1.0(-17)
$nac = 500$	99.87	78.0	4.3(-15)	1.3(-16)
$\|B\|_2 = 1$	99.85	78.5	6.2(-15)	2.7(-16)
$K(B) = 10^3$	99.80	89.1	1.1(-14)	2.7(-16)

In the first set of experiments, we show that the performance of E04NKF is strongly dependent on the sparsity of the matrices of the constraints and of the objective function. In particular, strictly convex test problems with decreasing value of the sparsity of the constraint matrix and of the matrix G are considered in Tables 1 and 2 respectively. In both Tables, the matrices G and B of the second test problems have a band structure. In these cases, the execution times have a slower increase as the sparsity of the matrices decreases.

Table 2. Different levels of the sparsity of the Hessian matrix

$\|G\|_2 = 1$ $\quad K(G) = 10^4$			
spars$(B) = 99.95\%$ $\quad \|B\|_2 = 1$ $\quad K(B) = 10^2$			
spars(G)	*time*	er_x	er_f
$n = 3000$ $m_i = 1000$ $m_e = 2000$ $nac = 200$			
99.95	14.8	3.7(-16)	2.3(-16)
99.85	22.8	8.9(-15)	4.4(-16)
99.80	26.0	6.0(-15)	1.1(-16)
99.60	46.3	2.6(-15)	1.1(-16)
$n = 3000$ $m_i = 1000$ m_e $\quad nac = 800$			
99.95	10.7	2.9(-16)	1.0(-17)
99.85	17.1	9.9(-16)	4.4(-16)
99.70	20.9	1.9(-15)	1.2(-16)
99.60	22.6	5.1(-15)	3.4 (-16)

Table 3 enables us to evaluate the performance of the first phase of the method used in E04NKF; indeed, when we have strictly convex test problems with equality constraints only and the number m_e of these constraints increases, the number $n - m_e$ of the variables that are free to move after the constraints are satisfied decreases. Consequently, the time for improving the value of the objective function and for determining its optimal value, decreases.

In Table 4, we consider strictly convex test problems with inequality constraints only, where the number *nac* of active constraints in the solution increases. In these cases, the routine E04NKF, since it is based on an active-set stategy, benefits by a small value of *nac*, while its performance degrades as *nac* increases.

The numerical results of Table 5 are concerning convex QP problems with equality and inequality constraints. These results confirm the previous considerations about the strictly

Table 3. QP problems with equality constraints

<table>
<tr><td colspan="4">$n = 5000$ $m_i = 0$
spars$(G) = 99.95\%$ $\|G\|_2 = 1$ $K(G) = 10^4$
spars$(B) = 99.95\%$ $\|B\|_2 = 1$ $K(B) = 10$</td></tr>
<tr><td>m_e</td><td>$time$</td><td>er_x</td><td>er_f</td></tr>
<tr><td>500</td><td>46.1</td><td>8.4(-15)</td><td>5.2(-16)</td></tr>
<tr><td>1000</td><td>51.9</td><td>2.0(-15)</td><td>1.0(-17)</td></tr>
<tr><td>1500</td><td>24.5</td><td>5.2(-16)</td><td>5.2(-16)</td></tr>
<tr><td>2000</td><td>20.8</td><td>3.3(-13)</td><td>1.0(-17)</td></tr>
</table>

Table 4. QP problems with inequality constraints

<table>
<tr><td colspan="4">$n = 3000$ $m_i = 1500$ $m_e = 0$
spars$(G) = 99.9\%$ $\|G\|_2 = 1$ $K(G) = 10^4$
spars$(B) = 99.9\%$ $\|B\|_2 = 1$ $K(B) = 10$</td></tr>
<tr><td>nac</td><td>$time$</td><td>er_x</td><td>er_f</td></tr>
<tr><td>50</td><td>50.2</td><td>7.6(-16)</td><td>1.0(-17)</td></tr>
<tr><td>100</td><td>60.6</td><td>8.0(-16)</td><td>2.1(-16)</td></tr>
<tr><td>500</td><td>188.7</td><td>1.2(-15)</td><td>4.4(-16)</td></tr>
<tr><td>1000</td><td>286.0</td><td>1.9(-15)</td><td>1.0(-17)</td></tr>
<tr><td>1500</td><td>285.1</td><td>1.0(-13)</td><td>4.3(-16)</td></tr>
</table>

convex QP problems.

The chance of varying the features of the test problems produced by the generator, enables us to draw the following numerical conclusions about E04NKF: this routine may be an effective approach to solve large and very sparse QP problems when the number of general inequality constraints that are active in the solution is small and/or we have a large number of equality constraints.

Table 5. Convex QP problems with equality and inequality constraints

<table>
<tr><td colspan="7">$n = 5000$
spars$(G) = 99.90\%$ $\|G\|_2 = 1$ $K(G) = 10^4$
spars$(B) = 99.95\%$ $\|B\|_2 = 1$ $K(B) = 10^2$</td></tr>
<tr><td>rank(G)</td><td>rank(Z^TGZ)</td><td>m_e</td><td>m_i</td><td>nac</td><td>$time$</td><td>er_f</td></tr>
<tr><td>4500</td><td>400</td><td>4000</td><td>1000</td><td>50</td><td>31.5</td><td>2.0(-15)</td></tr>
<tr><td>4500</td><td>50</td><td>4000</td><td>1000</td><td>400</td><td>22.9</td><td>4.9(-15)</td></tr>
<tr><td>4500</td><td>1000</td><td>2500</td><td>1000</td><td>400</td><td>65.4</td><td>5.9(-15)</td></tr>
<tr><td>4000</td><td>500</td><td>3000</td><td>1000</td><td>400</td><td>35.5</td><td>3.0(-15)</td></tr>
<tr><td>4000</td><td>500</td><td>3000</td><td>1000</td><td>50</td><td>30.4</td><td>1.3(-15)</td></tr>
<tr><td>4000</td><td>2000</td><td>1500</td><td>1000</td><td>50</td><td>258.6</td><td>1.0(-15)</td></tr>
<tr><td>3500</td><td>400</td><td>3000</td><td>1000</td><td>50</td><td>26.4</td><td>2.8(-14)</td></tr>
<tr><td>3500</td><td>50</td><td>3000</td><td>1000</td><td>400</td><td>20.6</td><td>2.7(-14)</td></tr>
</table>

Bibliography

[1] H. C. Andrews - B. R. Hunt, *Digital Image Restoration*, Prentice Hall, Englewood Press, New Jersey, (1977).

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

[3] C. Cortes - V. N. Vapnik, *Support vector network*, Machine Learning, **20**, pp. 1–25 (1995).

[4] R. Fletcher, *Practical Methods od Optimization, Second Edition*, John Wiley & Son Ltd., (1987).

[5] E. Galligani, C^1 *surface interpolation with constraints*, Numerical Algorithms, **5**, (1993), pp. 549–555.

[6] P. E. Gill - W. Murray - M. A. Saunders, *SNOPT: An SQP Algorithm for Large–Scale Constrained Optimization*, Numerical Analysis Report 96-2, Departement of Mathematics, University of California, San Diego, (1996).

[7] P. E. Gill - W. Murray - M. A. Saunders - M. H. Wright, *Maintaining LU Factors of a General Sparse Matrix*, Linear Algebra and its Applics., **88/89**, (1987), pp. 239–270.

[8] D. R. Kinkaid - T. C. Oppe - J. R. Respess - D. M. Young, *ITPACKV2 2C User's Guide*, Report CNA–191, Centre for Numerical Analysis, University of Texas, Austin, (1984).

[9] I. Maros - Cs. Mèszàros, *A Repository of Convex Quadratic Programming Problems*, Optimization Methods and Software, (1999).

[10] B. A. Murtagh - M. A. Saunders, *MINOS 5.4, User's Guide*, Report SOL 83-20R, Department of Operations Research, Stanford University, (1995).

[11] *NAG Fortran Library Manual, Mark 18*, (1998).

[12] J. S. Pang - D. Chan, *Iterative methods for variational and complementarity problems*, Mathematical Programming, **24**, (1982), pp. 284–313.

[13] M. Patriksson, *Nonlinear Programming and Variational Inequality Problems: A Unified Approach*, Kluwer Academic Publisher, Dordrecht, (1999).

[14] D. L. Zhu - P. Marcotte, *An extended descent framework for variational inequalities*, Journal of Optimization Theory and Applications, **80**, (1994), pp. 349–366.