Kybernetika

Miroslav Tůma

A quadratic programming algorithm for large and sparse problems

Kybernetika, Vol. 27 (1991), No. 2, 155--167

Persistent URL: http://dml.cz/dmlcz/124517

KYBERNETIKA — VOLUME 27 (1991), NUMBER 2

A QUADRATIC PROGRAMMING ALGORITHM FOR LARGE AND SPARSE PROBLEMS

MIROSLAV TŮMA

We describe a primal feasible, convergent algorithm for the quadratic programming problem. It is especially designed to cope with the large and sparse instances of this type with their structural and numerical algorithmic specialties. It makes use of the new improvements that have not been used for the general quadratic programming or that have not been used so far.

1. INTRODUCTION

We consider the quadratic programming problem (QP), i.e. the problem of minimizing a quadratic function subject to a set of linear constraints. We assume the constraints to be in the canonical form:

$$\left.\begin{array}{ll} \min f(x) = q^{\mathrm{T}}x + 0{\cdot}5x^{\mathrm{T}}Qx, & \ldots (1a) \\ Ax = b, & \ldots (1b) \\ l \leqq x \leqq u\,, & \ldots (1c) \\ x, q, l, u \in \mathbb{R}^n,\ b \in \mathbb{R}^m, & \ldots (1d) \\ A \in \mathbb{R}^{m,n},\ Q \in \mathbb{R}^{n,n} & \ldots (1e) \end{array}\right\} \quad (1)$$

We are concerned with developing algorithms to find a local minimum of (1) when these matrices A, Q are large and sparse. This problem often arises when nonlinear function is minimized using sequential quadratic programming methods (SQP) (see [12], [13]). Without loss of generality we use the form of constraints that can be obtained, for instance, with the aid of the transformation of the linear programming solvers using the primal simplex algorithm (see [19]).

To cope with this problem we use reduced gradient methods (see [2]) in the active set framework (see [14]). In contrast with the MINOS system (cf. [22]) we have specialized for the large and sparse systems where most of the variables is included in the quadratic terms of the objective function. General algorithmic scheme makes use of some principles that have not been used for the general large and sparse QP

so far – specialization on multiple pricing with the protection against many degenerate steps, new procedures for the sparse matrix algebra, rules to keep partition. Data structures used are suitable to this strategy.

Inner loop of the direction search makes use of the truncated Newton method for the reduced Hessian matrix (cf. [9]).

2. ALGORITHMIC OVERVIEW

To solve this problem, we will form a sequence of iterates $\{x_i\}_{i \geq 1}$. Let us call the set of *active* constraints for the iterate $x_i \in \mathbb{R}^n$ the set of the constraints that hold at x_i as equates. Components of individual iterates are called *variables*: we will write $x_i = (x_i^1, \ldots, x_i^n)$. When we are dealing with the general ith iteration we often omit subscripts of the iterates and of the matrices and vectors relating to them.

Set of constraints is composed of the two parts. Linear constraints (1b) form a manifold. Without loss of generality we assume matrix A to have full row rank. In practical computation the problem of the row rank of A is usually solved in some preprocessing phase, e.g. in simplex linear programming (LP) solver. Bounds (1c) correspond to individual variables. At each phase of the iteration process we can divide bounds into two groups: *lower* or *upper active* and the rest that we will call *passive* ones. *Variables* that correspond to active bounds will be called (*lower* or *upper*) *active*. The rest of the variables are called *passive*. At each iteration there is a *working* set of indices of bounds $I_W \subset \{1, \ldots, n\}$ that are to be kept *fixed*, that means that the variables determined by that bounds are not allowed to change their values throughout the iteration. We also speak about *fixed* variables. Thus we are to solve a sequence of problems:

$$x_{i+1} = \min \{f(x) \mid (1b) - (1e) \text{ are satisfied},$$
$$x^j \text{ are fixed for } j \in I_W\}.$$

The problems are solved only approximately even when the objective is quadratic.

Let us have some feasible iterate x. First feasible iterate x_1 was obtained, for instance, using the first phase of the simplex method of linear programming (LP). Computing a feasible direction amounts to finding a nonzero vector $d \in \mathbb{R}^n$, such that for some nonzero scalar α it holds:

$$g(x)^T d < 0,$$
$$Ad = 0,$$
$$l \leqq x + \alpha d \leqq u,$$
$$x^j \text{ is fixed for } j \in I_W.$$

A convenient and a practical way to compute such a direction is to eliminate some variables (basic) and express them in terms of the rest (nonbasic) ones using the constraints (1b). Set of basic variables we call simply the basis.

Thus we can assume partition $A = [B, N]$, where B is a square nonsingular basic matrix. Working set then induces partition of nonbasic part of the matrix: $N = [E, W]$. We do not use notation of [4], [20] to discern our slightly different concepts. This matrix partitions induce partitions for vectors the x, d, u, l and for the other vectors introduced in he sequel ($x = (x_B, x_N)$, $x_N = (x_E, x_W)$, ...) and for the index sets $\{1, \ldots, n\} = T_B \cup I_N, I_N = I_E \cup I_W$. Let $n_B = |I_B|$, $n_N = |I_N|$, $n_W = |I_W|$, $n_E = |I_E|$. Our problem (1) can thus be rewritten as:

$$\min f^R(x_N),$$
$$l_B \leqq B^{-1}(b - Nx_N) \leqq u_B,$$
$$l_N \leqq x_N \leqq u_N,$$

where $f^R(x_N) = f(B^{-1}(b - Nx_N), x_N)$ is called the reduced function. Its gradient and Hessian at x will be denoted by $g^R(x_N)$ and $H^R(x_N)$, respectively.

2.1 Computing of the basis part of the direction vector and general computing scheme

Feasible directions d are formed in the following manner. First, there is a computed part d_N (see the next section). Then we get $d_B = -B^{-1}Nd_N$. When the basis is generated (some of its variables are active), arbitrarily small move along the direction d_N can induce infeasibilities in basis variables. Such a pathological direction vector we call degenerate. (Note, that the zero direction vector we call also degenerate.)

To evade this situation, concept of *maximal basis* was in [7] introduced. In more detail, basis is constructed and updated, so that it contains *as many columns as possible corresponding to the passive variables.* For the set of the columns of A together with all linearly independent subsets form the matroid, we know, that all maximal independent subsets of this matroid obtained by sequential exchanges of the columns of the matrix A are of the same cardinality. Every maximal basis can be thus decomposed into some maximal independent set of the above mentioned matroid and the set of columns corresponding to the active variables to complete it.

Construction of the basis part of vector d_i can be rewritten as the sum of transformed contributions of individual nonbasics:

$$d_B = -B^{-1}Nd_N = -\sum_{j \in I_N} B^{-1}N(d)_j,$$

where $(d)_j \in \mathbb{R}^{n_N}$, $(d)_j = ((d)_j^{k_1}, \ldots, (d)_j^{k n_N})$, $(d)_j^j = d_N^j$, $(d)_j^k = 0$ for $k \in I_B$, $k \neq j$. When the basis is maximal, then any passive nonbasics will only induce changes in passive basics (see [4], [5], [7]).

Definition 2.1. Let us denote by $J(B)+$ and by $J(B)-$ index sets, subsets of I_B, containing indices of (degenerate) variables upper and lower active, respectively.

Theorem 2.2. Let $d \in \mathbb{R}^n$ be a feasible direction vector. It is not degenerate iff it holds for the indices $j \in J(B)+ \cup J(B)-$: $d^j \leqq 0$ for $j \in J(B)+$, $d^j \geqq 0$ for $j \in J(B)-$.

The proof of this theorem is trivial. Thus, when some of the transformed contributions corresponding to some active variable that we are summing up to get d_B do not conform to the condition in Theorem 2.2, we can omit them.

To get the direction vector we thus use the following scheme. Let the basis be maximal.

Algorithm 2.3.

Step 1. Compute d_N. Methods for this task will be described later.

Step 2. Set $d_B = -B^{-1}d_N$. If d_B is not degenerate, then d is found, end.

Step 3. Compute contributions $B^{-1}N\ (d)_j$ for some restricted number of vectors $\{(d)_j \mid j \in I_N,\ j = j_1, \dots j_\phi\}$. Direction vector d_B is then a sum of those contributions of these ones that satisfy condition of the Theorem 2.2.

The value of ϕ we have found experimentally so that the complexity of calculating of these contributions is relatively small (For our problems we have used $\phi = 3 - 4$ – chosen so that the computing time of *Step* 3 is approximately twice the time used in *Step* 2. Indices of I_N used are chosen so that corresponding columns of N have small number of nonzeros.

If none of the contributions in *Step* 3 of Algorithm 2.3 conforms to the condition of Theorem 2.2, direction vectors is zero and we use Bland rule to remove (theoretical) cycling (cf. [1]). This rule is extremely simple and consists of two steps. (In fact, [1] contains two different rules. The second of them is more complicated and we have not used it.):

a) among all candidates to enter be basis select variable with the smallest superscript,

b) if more variables complete for leaving the basis select variable with the smallest superscript.

Although the importance of this rule for linear programming problems, it has been originally designed for, is primarily theoretical, it can be effectively used in this case together with the whole above mentioned technology of direction vectors finding.

2.2 General scheme of the algorithm

The whole computation can be described as follows. Its steps will be cleared further. The sets W_i are kept so that they always contain indices of all active nonbasics.

Algorithm 2.4.

Step 1. Start with some feasible iterate $x_1 \in \mathbb{R}^n$. Form B so that it is maximal. Set $W_1 = N$, $i = 1$.

Step 2. Test on the optimality TO1. If TO1 satisfied then go to *Step* 5.
Step 3. Update working set and induced partitions.
Step 4. Compute nonbasic part of the direction vector using DV2.
Step 5. Test on optimality TO2. If satisfied then end.
Step 6. Update working set and induced partitions.
Step 7. Compute nonbasic part of direction vector using DV1.
Step 8. Compute basic part of the direction vector. If direction vector degenerate then go to *Step* 10.
Step 9. Compute step length and form a new iterate x_{i+1}.
Step 10. Set $i = i + 1$, form new working set and partitions, go to *Step* 2.

2.3 Nonbasic part of the direction vector

Now we will describe procedures DV1 and DV2. First, we will mention procedure DV2 for direction vector finding that makes use of second order information in the objective function. We will obtain it solving inexactly the Newton equation (see [6]) in the space of nonbasics:

$$\left.\begin{aligned} H^R(x_E)\, d_E &= -g^R(x_E), \quad \ldots \text{ (2a)} \\ d_W &= 0 \qquad\qquad\quad \ldots \text{ (2b)} \end{aligned}\right\} \tag{2}$$

Considering the data structures for large sparse problems, the one alternative is to use the conjugate gradient methods (CG), where (possibly dense) reduced Hessian matrix can be stored implicitly. We use the Reid form of implementation (see [25]) in the truncated Newton environment (see [9]): We are looking for the solution of the system $H^R(x_E)\, d_E = -g^R(x_E) + r$, where $\|r\|/\|g^R(x_E)\| \leqq \eta$, $\eta = \max(\varepsilon^{0.5}, \min((1/\text{NIT } 1 + 1), g^R(x_E))$ is specified in concordance with [9]; ε is a machine accuracy for EC 1045; NIT 1 denotes number of *Steps* 7 of Algorithm 2.4 made so far. On the output we get solution of this system or some other descent direction, when indefiniteness of the reduced Hessian is faced. Third natural way to leave the algorithm is used, when accuracy of the solution is lost and we overpass maximum number of iteration having been set up.

As an alternative DV 1 we will use a direction of a negative reduced gradient, obtained as follows:

$$d_N^j = \begin{cases} -g^R(x_N)^j \text{ for } x^j \text{ passive}, \\ \max(0, -g^R(x_N)^j) \text{ for } x^j \text{ lower active}, \\ \min(0, -g^R(x_N)^j) \text{ for } x^j \text{ upper active}, \\ 0 \quad \text{for} \quad j \in I_W. \end{cases}$$

This possibility is used relatively infrequently for radical changes in the space of nonbasics when the reduced gradient has been made small having used repeated steps with DV 2. Using direction vector obtained by DV 1, more active variables are to leave their bounds as detected by the Lagrange multipliers calculation in the procedure TO 2 of the algorithm.

2.4 Determination of the steplength

When the nondegenerate direction vector is found, we have to find a new iterate. We set, as usual, $x_{i+1} = x_i + \alpha_i d_i$, scalar α is set to $\min(\alpha', \alpha_{\max})$, where

$$\alpha_{\max} = \min\left(\min_{d_i^j<0} \frac{l_j - x_i^j}{d_i^j}, \min_{d_i^j>0} \frac{u_j - x_i^j}{d_i^j}\right).$$

For small quadratic problems it is usual to set $\alpha' = 1$. Here we solve the system (2) inexactly. In the large instances of the problem, rounding errors make the situation even more complicated. We usually need one or even more steps of simple quadratic interpolation/extrapolation procedure. So that we generally choose α' such that the Armijo conditions hold (cf. [13]):

$$d_i^{\mathrm{T}} g(x_i + \alpha' d_i) \geqq \mu_1 d_i\, g(x_i)$$

$$f(x_i + \alpha' d_i) - f(x_i) \leqq \mu_0 \alpha' d_i\, g(x_i)$$

We have chosen $\mu_0 = 0{\cdot}1$, $\mu_1 = 0{\cdot}9$.

2.5 Test on optimality TO 1

In *Step* 2 we have at first the test described in Algorithm 2.4. The idea behind the first rule of this routine is to test the absolute size of the reduced gradient against some weak limit and to tighten this limit from iteration to iteration. Thus using the parameters η_G and η_M we simulate a sequence $\{\eta_i\}_{i\geqq 1} \to \eta_{\mathrm{opt}}$.

Algorithm 2.5. Parameters: $\varepsilon_x = 0{\cdot}1$, $\varepsilon_f = 0{\cdot}01$, $\varepsilon_g = 0{\cdot}0001$ (see [13]). This parameters help to specify when to declare that TO 1 is satisfied due to "small" changes in variables, functional value or reduced gradient size. RGNORO is the reduced gradient norm computer after the latest move along the direction vector obtained using DV 1 – initialized by 0; $\eta_G = 0{\cdot}2$; $\eta_M = 0{\cdot}7$ – it is set to $\eta_M \,.\, \eta_M$ every time when DV 1 is used; RGFL = $(\max(\eta_G \,.\, \eta_M \,.\, \mathrm{RGNORO}, 10^{-3})$; RGSL = = $\max(0{\cdot}01 \,.\, \mathrm{RGFL}, \eta_{\mathrm{opt}})$, where $\eta_{\mathrm{opt}}(10^{-3} - 10^{-8})$ is the demanded output accuracy of the reduced gradient; $\varepsilon = 10^{-8}$ – it is a machine accuracy for EC 1045; π is the first order estimate to Lagrange multipliers to (1b); $\Delta(x_i)_{N\setminus W}$ or Δf_i are vector of changes in the appropriate variables or change in the function value, respectively – they are initialized by zero vector resp. by zeros.

If ZSFL = 1 then TO 1 is satisfied

if $(\|g^R(x_{N\setminus W})\| \leqq \mathrm{RGFL})$ then
  if $(\|\Delta x_E\| \geqq (\varepsilon_x + \varepsilon^{0.5}) \,.\, (1 + \|x^0\|) \wedge$
    $\Delta f| \leqq (\varepsilon_f + \varepsilon) \,.\, (1 + |f(x)|) \vee$
    $\|g^R(x_E)\| \leqq \varepsilon_G \,.\, (\mathrm{RSFL} \,.\, \max(1, \|\pi\|)$ then TO 1 is satisfied
  end if
end if.

When solution cannot be improved in some sequence of iterations, parameter ZSFL has been set to one. Here, when ZSFL = 1, TO 1 is thus also satisfied.

2.6 Test on optimality TO 2

In this procedure we compute first order estimate of the Lagrange multipliers corresponding to the constraints captured by the set W (see Section 3).

Kuhn-Tucker conditions for the system (1) can be written as follows:

$$\begin{aligned} g(x) + A^{\mathrm{T}}\mu + \bar{\lambda} + \tilde{\lambda} &= 0 \\ \bar{\lambda} &\leqq 0 \\ \tilde{\lambda} &\geqq 0 \\ \tilde{\lambda}^j(x^j - u_j) &= 0 \\ \bar{\lambda}^j(x^j - l_j) &= 0 \\ j \in \{1, \ldots, n\}\ ; \quad \mu, \bar{\lambda}, \tilde{\lambda} &\in \mathbb{R}^n \end{aligned}$$

Both elements $\bar{\lambda}^j$, $\tilde{\lambda}^j$ cannot be nonzero at the same moment for any $j \in \{1, \ldots, n\}$. Our multiplier estimate thus always includes one of these elements according to the type of activity of individual variables. For $j \in I_W$ is λ^j set to $\bar{\lambda}^j$ when x^j is upper active. Otherwise, λ^j is set to $\tilde{\lambda}^j$. Vector λ we will call in the sequel the vector of multipliers. The above conditions thus imply the following strategy.

When there are upper or lower active nonbasics $x^{k_1}, \ldots, x^{k_p}$, such that for the corresponding estimate of multipliers $\lambda^{k_1}, \ldots, \lambda^{k_p}$ it holds $\lambda^{k_j} > \eta_\lambda$ or $\lambda^{k_j} < -\eta_\lambda$ ($\eta_\lambda \approx 10^{-6}$ is a tolerance on positiveness respectively, then the bounds corresponding to these constaints are excluded from the working set.

Algorithm fits into the Dembo-Sahi framework (see [8]) and it is thus globally convergent for there hold following conditions:

a) Directions chosen according to DV 1 are *gradient related* as shown in [9]. That is, for a convergent subsequence $\{x_k\} \to x'$ with x' not optimal it holds $\lim_{k \to \infty} g(x_k)^{\mathrm{T}} d_k < 0$.

b) First order multiplier estimates are consistent. (When $\{x_k\} \to x^*$ then $\{\lambda_k\} \to \lambda^*$, where x^* and λ^* are a Kuhn-Tucker point and corresponding vector of multipliers.)

c) Test on optimality TO 2 is implemented with the mentioned tolerance η_λ on the elements of the vector of multipliers. This implies dropping only those bounds within some fixed fraction of the most negative and positive multiplier for the lower and upper bounded variables, respectively.

Stronger convergence results would be possible with the nondegeneracy assumption (but this is unreasonable for the large problems) or with another form of the algorithm (see [2]).

2.7 Update of working sets and of the matrix partitions

Let us first note, that the exchange of columns is being done only by the changes in some index permutation vector that keepts track of the status of variables (basic, nonbasic, fixed, active, ...).

Working set is constructed at the beginning of Algorithm 2.4. We have always obtained x_1 by using LP, so we can set $W_1 = N$. When some partial or complete solution was given at the beginning then to construct the maximal basis we have also used the first phase of the primal simplex algorithm to compute x_1.

Correction of W can be used in *Step* 3. We want sometimes to keep the dimension of the reduced space for the procedure DV 2 under some predefined limit. It can happen for number of reasons (to suppress influence of the rounding errors, to exclude variables corresponding to dense columns of N for some iterations, when one exchanges the CG routine for some variable metric solver (see [20]), to solve Newton system one could use such a restriction to cope with the lack of memory, to make use of some specific (block) structure of the reduced Hessian). Further, all active nonbasics are to be captured by W.

In *Step* 6 we usually free all of the active nonbasics from the set W.

More comprehensive update in *Step* 10 covers two cases. When degenerate direction vector is detected in *Step* 8, we are only to exchange one degenerate basis variable for the fixed one. Otherwise, update of W and the parallel update of B proceeds in three phases:

Step 10α All active nonbasics are moved to W.

Step 10β All active basics, that were passive before the latest step will be tried to be exchanged for the free nonbasics. This step will be described in our data structures later.

Step 10γ All active nonbasics are moved to W.

3. IMPLEMENTATION

This implementation is intended for general large sparse problems and data structures are especially designed for these types of problems (see also [10], [11], [23], [24], [31]).

The initial matrix A, obtained using MPS-file or subroutine with specified conventions, is transformed into a classical columnwise sparse data structure. In case of the sequence of the QP problems there is input simply in this form. It makes use of two "long" vectors A, SA (their length is equal to the number of matrix nonzeros) where numerical values of nonzero matrix elements and their row indices are stored, respectively, at corresponding positions. Then the one pointer vector PTC of the length $n + 1$ is used. Nonzeros of the matrix form a compact block, beginning of the column i that is stored at the positions with indices PTC(i). That means that

the numerical value of the first nonzero element of the column i is stored in A(PTC(i)), its row index in SA(PTC(i)), its last nonzero element at PTC($i + 1$) $- 1$.

Symmetric Hessian matrix of the quadratic function has rowwise stored diagonal and upper triangular part using two "long" parallel arrays and one pointer vector again.

Columnwise storing of A is advantageous for the prevailing operations with it: rewritte of the nonzero of some column into the working structure, test on the lengths of columns (numbers of their nonzeros).

Reduced gradient and Hessian are computed using a matrix Z spanning the nullspace of A (see [20]). Define Z by: $Z^T = (-E^T B^{-T}, I, 0)$. Then we set $g^R(x_E) = Z^T g(x)$, $H^R(x_E) = Z^T H(x) Z$. Reduced gradient is recomputed when iterate or projection matrix changes, reduced Hessian is used implicitly, for in the CG procedure used to solve Newton equations, only matrix-vector products are needed.

Neither matrix Z, possibly dense, is stored. Instead we keep the LU-factorized basis matrix B. Its factors are updated when there are exchanged columns in the course of the computation. For the update we have used original algorithms that make possible to use the Reid interchanges to minimize increase in the condition number of factorization and the necessary space (see [29]) without using its space-consuming data structure (see [25], [26]). The basis matrix is regularly refactorized by the Markowitz-like procedure (see [18], [29]). Thus to multiply some vector by B^{-1} we need only two backward steps.

Having introduced basic data structures, we can describe how to compute the Lagrange multipliers estimates (see [15]). By $\pi \in \mathbb{R}^m$ and $\lambda \in \mathbb{R}^n$ we will denote the vector of estimates of the Lagrange multipliers for the constraints (1b) and for the fixed bounds, respectively. We will first set $\pi = B^{-T} g(x_B)$, where B^{-T} is the inverse of B^T. This number is recomputed very often, for that it is used for the reduced gradient computation. Then we set $\lambda = g(x_W) - W^T\pi$. Estimates λ are then recomputed only when needed for the use in TO 2.

Now we will describe *Step* 10β of the update of matrix partition, as mentioned in 2.6. Let the unit vector corresponding to the jth variable be denoted by $(e)_j$. Dimension of such a vector is clear from the context. All active basics that were passive before the latest *Step* 9 of Algorithm 2.4 are processed in the next three phases:

Let us process such an active basic variable, say j. Then

a) Compute a vector $v \in \mathbb{R}^m$ by solving the equation $B^T v = (e)_j$.

b) Compute a vector $y \in \mathbb{R}^E$, $y = E^T v$.

c) For all free nonbasic variables x^k compute $d^k = \min(|x^k - l^k|, |x^k - u^k|)$ and set $y_{m5} = 0{\cdot}5 \max |y^k|$. Then find index $i \in I_E$, such that $z^i/\sqrt{c^i} = \max(z^k/\sqrt{c^k} \,|\, |y^k| \geqq \geqq y_{m5}/\min(3, \sqrt{c^k})$, where c^k is the number of nonzero elements in the column k (that is PTC$(k + 1)$ $-$ PTC(k)).

Motivation for this process was the following. We can write $y^j = y^T(e_j)$. If there is during the update exchanged the basis variable x_j for the free nonbasic variable x_i. Then for the updated matrix B^+ that was obtained from B by column exchange

we can write:

$$B^+ = B + (E(e)_i - B(e)_j)\,(e)_j^{\mathrm{T}} \quad \text{that is}$$
$$B^{-1}B^+ = I + (B^{-1}\,E(e)_i - (e)_j)\,(e)_j^{\mathrm{T}}\,.$$

Thus, the demand to have the matrix B^+ regular equivalent to the demand to have y^i nonzero. The effort to have the condition number of B^+ small can be positively influenced by taking y^i with the absolute value relatively large. We have chosen the compromise such that the variable entering the basis is far from its bounds and the output matrix is preferably sparse.

4. PROGRAM SYSTEM AND EXPERIMENTS WITH IT

The whole algorithmic schema was programmed in Fortran (ANSI 77) and it contains about 10 000 commented source lines now (see [30]). The structure of the program system is completely modular up to the basic vector and matrix-vector operations. Thus it is possible to exchange some of the modules of the program system (e.g. linesearch solver, routines for the LU-factorization) by another ones. For the data input we can choose one of the three possibilities: simple interactive input, MPS-file input and the one using a subroutine conforming to our conventions.

Individual program modules conform to the convention of the system UFO (see [17]) and they will serve there as the large sparse QP solver.

We have debugged this program system using problems of [16]. To demonstrate its properties we will describe the test problem adapted from the optimal control of some simple dynamic system as described in [21], [28]. In this example we will use subscripts to discern variables instead to discern iterations.

Problem 4.1.

$$\min f(x)$$
$$f(x) = 0{\cdot}5 \sum_{i=1}^{k} c_i x_i^2 - 0{\cdot}5 \sum_{i=1}^{k} d_i y_i^2$$
$$P = \{x \mid x_{i+1} = x_i + 0{\cdot}2y_i\,,$$
$$y_{i+1} = y_i - 0{\cdot}01y_i^2 - 0{\cdot}004x_i + 0{\cdot}2u_i\,,$$
$$-0{\cdot}2 \leqq u_i \leqq 0{\cdot}2\,,$$
$$y_i \geqq -1\,, \quad x_0 = 10\,, \quad y_k = 0\,,$$
$$i = 0\,, \ldots, k-1\,;\ c_i = 1,\ d_i = 1 \text{ for } i = 1, \ldots, k\}$$

This problem was linearized at point with $x_i = 10$ for $i = 1, \ldots, k$; $y_i = -1$ for $i = 1, \ldots, k-1$; $y_k = 0$; $u_i = 0{\cdot}1$ for $i = 1, \ldots, k$. Instead of the looking for a better feasible point of the nonlinear problem, we have modified the right-hand

side of the set of linearized constraints so that this point is feasible also for the QP problem. Basic output data are displayed in Table 1 – instead of k we use parameter $n = 3(k + 1)$.

Table 1. Solving of Problem 4.1.

n	n_{maj}	n_{cyc}	n_{CG}	RGNORM	t
180	2	60	0	$0{\cdot}3\,.\,10^{-14}$	8·40
240	4	78	40	$0{\cdot}2\,.\,10^{-13}$	19·18
300	14	282	459	$0{\cdot}4\,.\,10^{-13}$	118·72
396	6	124	699	$0{\cdot}8\,.\,10^{-6}$	128·66
480	14	203	1572	$0{\cdot}1\,.\,10^{-4}$	306·65
600	11	162	2667	$0{\cdot}3\,.\,10^{-4}$	552·04
690	12	156	3516	$0{\cdot}6\,.\,10^{-4}$	842·01

n is the parameter for the size of the problem 4.1, n_{maj} is the number of computed direction vectors according to thr procedure DV 1, n_{cyc} is the whole number of the iterations, n_{CG} is the number of CG iterations, RGNORM is the output norm of the reduced gradient, t is the time of the computation at EC 1045 in seconds.

Comparative tests of the specialized network solver NLPNET (cf. [9]) and of the modification of the classical system MINOS (see [12], [22]) show clear advantages of the radical changes in the working set and in the partition of the matrix A. We have created a system with the advantages of both these systems – implicit data structures of the projection matrix can be used for broader spectrum of problems than NLPNET can handle. The philosophy of working with constraints is different from MINOS. Thus we can use our system for a large general setting of QP with all variables in nonlinear terms of the objective function and we have verified solvability of such problems. We had not faced degenerate direction vectors in any of the test problems (direction vector obtained using DV 2 cannot be degenerate for the basis is maximal).

Distribution of the CG iterations in the various phases of the computation is very interesting. First, very little iterations are used for the determination of the direction vector in DV 2. Their number radically increases towards the end. This procedure is then left very often for the maximum number of CG iterations MITCG is overpassed. This number is set to twice the size of the dimension of the reduced space. In this case, such a situation is promoted by the fact that the reduced objective function is relatively flat. To demonstrate it, we have experimented with various limits on the output accuracy. Relative accuracy *prec* is related to the value of the objective function obtained with a great effort and with adaptively changed inner constants. Output residuum was in this case less than $1\,.\,0{\cdot}10^{-7}$. If we denote this value by f^*, then the parameter *prec* in Tables 2 and 3 is defined by $|f^* - f|/|f^*|$.

Our main task was to verify practical computability of the quadratic subprograms of such dimensions, where most of the variables are included in the quadratic part

Table 2. Accuracy and number of iterations for the QP problem with $n = 480$.

RGNORM	n_{maj}	n_{cyc}	n_{CG}	t	prec
$0{\cdot}50 \,.\, 10^{-1}$	9	217	799	203·06	10^{-6}
$0{\cdot}42 \,.\, 10^{-2}$	10	225	917	216·68	10^{-8}
$0{\cdot}21 \,.\, 10^{-4}$	12	196	1307	270·99	10^{-12}
$0{\cdot}11 \,.\, 10^{-4}$	14	203	1572	306·65	10^{-12}

RGNORM is the output value of the reduced gradient, n_{maj} is the number of computed direction vectors using the procedure DV 1, n_{cyc} is the whole number of iterations in Algorithm 4.2, n_{CG} is the number of CC iterations, t is the optimization time in seconds, prec is the above described parameter of precision.

Table 3. Accuracy and number of iterations for the QP problem with $n = 480$.

RGNORM	n_{maj}	n_{cyc}	n_{CG}	t	prec
$0{\cdot}56 \,.\, 10^{-1}$	8	130	821	201·45	10^{-7}
$0{\cdot}37 \,.\, 10^{-3}$	8	139	1182	272·64	10^{-9}
$0{\cdot}26 \,.\, 10^{-4}$	11	162	2667	552·04	10^{-11}

Notation is the same as in Table 3.

of the objective function. The obtained results have shown that by the sequences of the QP problems it will be possible to solve even rather large problems of nonlinear programming.

(Received April 4, 1990.)

REFERENCES

[1] R. G. Bland: New finite pivoting rules for the simplex method. Math. Oper. Res. *2* (1977), 103—107.

[2] P. H. Calamai and J. J. Moré: Projected gradient methods for linearly constrained problems. Math. Programming *39* (1987), 93—116.

[3] T. F. Coleman: Large Sparse Numerical Optimization. (Lecture Notes in Computer Science 165.) Springer-Verlag, Berlin—Heidelberg—New York—Tokyo 1984.

[4] R. S. Dembo: NLPNET — User's Guide and System Documentation, School of Organization and Management Working Paper Series B # 70, Yale University, New Haven, CT 1983.

[5] R. S. Dembo: A primal truncated Newton algorithm with application to large-scale nonlinear network optimization. Math. Programming Study *31* (1987), 43—71.

[6] R. S. Dembo, S. C. Eisenstat and T. Steihaug: Inexact Newton methods. SIAM J. Numer. Anal. *19* (1982), 400—408.

[7] R. S. Dembo and J. G. Klincewicz: Dealing with degeneracy in reduced gradient algorithms. Math. Programming *31* (1985), 357—363.

[8] R. S. Dembo and T. Sahi: A convergent active-set strategy for linearly-constrained optimization. School of Organization and Management Working Paper Series B # 80, Yale University 1984.

[9] R. S. Dembo and T. Steihaug: Truncated-Newton algorithms for large-scale unconstrained optimization. Math. Programming *26* (1983), 190—212.

[10] A. Drud: CONOPT: A GRG code for large sparse dynamic nonlinear optimization problems. Math. Programming *31* (1985), 153—191.
[11] L. F. Escudero: An Algorithm for Large-scale Quadratic Programming and its Extensions to the Linearly Constrained Case. IBM Scientic Centre Report SCR-01.81, Madrid 1981.
[12] Y. Fan, L. Lasdon and S. Sarkar: Experiments with successive quadratic programming algorithms. J. Optim. Theory Appl. *56* (1988), 359—383.
[13] R. Fletcher: Practical Methods of Optimization, Vol. 2: Constrained Optimization. J. Wiley, New York—Chichester—Brisbane—Toronto 1981.
[14] P. E. Gill and W. Murray: Newton-type methods for unconstrained and linearly constrained optimization. Math. Programming *28* (1974), 311—350.
[15] P. E. Gill and W. Murray: The Computation of Lagrange Multiplier Estimates for Constrained Minimization, Rep. NAC 77, National Physical Laboratory, England 1976.
[16] W. Hock and K. Schittkowski: Test Examples for NLP Codes. Springer-Verlag, Berlin—Heidelberg—New York 1981.
[17] L. Lukšan: System UFO. User's Guide-version 1989 (in Czech). Res. Rep. V-441, SVT ČSAV, Prague, 1989.
[18] H. M. Markowitz: The elimination form of the inverse and its applications to linear programming. Management Sci. *3* (1957), 225—269.
[19] B. A. Murtagh: Advanced Linear Programming: Computation and Practice. McGraw Hill New York 1981.
[20] B. A. Murtagh and M. A. Saunders: Large-scale linearly constrained optimization. Math. Programming *14* (1978), 41—72.
[21] B. A. Murtagh and M. A. Saunders: A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints. Math. Programming Study *16* (1982), 84—117.
[22] B. A. Murtagh and M. A. Saunders: MINOS 5.1 User's Guide, Tech. Rep. SOL 83-20 R, Dept. Oper. Res., Stanford University, Stanford, CA, 1983, revised 1987.
[23] O. Osterby and Z. Zlatev: Direct methods for sparse matrices. (Lecture Notes in Computer Science 157.) Springer-Verlag, Berlin—Heidelberg—New York—Tokyo 1983.
[24] S. Pissanetzky: Sparse Matrix Technology. Academic Press, London—Orlando—San Diego—New York—Austin—Toronto—Montreal—Sydney—Tokyo 1984.
[25] J. K. Reid: On the method of conjugate gradients for the solution of large sparse systems of linear equations. In: Large Sparse Sets of Linear Equations (J. K. Reid, ed.), Academic Press, London 1971, pp. 231—254.
[26] J. K. Reid: Fortran Subroutines for Handling Sparse Linear Programming Bases. Rep. AERE Harwell, R. 8269. Harwell 1976.
[27] J. K. Reid: A sparsity-exploiting variant of the Bartels-Golub decomposition for linear programming bases. Math. Programming *24* (1982), 55—69.
[28] P. S. Ritch: Discrete optimal control with multiple constraints I: Constraint separation and transformation technique. Automatica *9* (1973), 415—429.
[29] M. Tůma: Large and Sparse Quadratic Programming (in Czech). Ph.D. Thesis, SVT ČSAV, Prague 1989.
[30] M. Tůma: SPOPT-Program System for the Solution of Large Sparse Problems of Linear and Quadratic Programming (in Czech). Res. Rep. V-391, SVT ČSAV, Praha 1989.
[31] Z. Zlatev: A survey of the advances in the exploitation of the sparsity in the solution of large problems. Internat. Congress on Comp. and Appl. Math., University of Leuven, Belgium 1986.

Ing. Miroslav Tůma, CSc., Středisko výpočetní techniky ČSAV (General Computing Centre — Czechoslovak Academy of Sciences), Pod vodárenskou věží 2, 182 07 Praha 8. Czechoslovakia.