Facultad
de
Ciencias

Software de programación cuadrática general y generalizada para MATLAB: teoría y práctica

(General Quadratic Programming Software for MATLAB: Theory and Practice)

Trabajo de Fin de Grado
para acceder al

GRADO EN MATEMÁTICAS

Autor: Carlos Crespo Menéndez
Directora: Cecilia Pola Méndez
Junio-2021

Abstract

The objective of this work is the study and the implementation in MATLAB of an active set numerical method to solve general quadratic optimization problems (GQP), that is to say, problems that include the nonconvex case.

Given any symmetric matrix, we study how to calculate a Cholesky factorization, total or partial. This factorization allows to calculate descent directions for any Hessian of the objective function whether positive definite or not. Furthermore, we address the resolution of quadratic problems with l_1 terms in the objective function (generalized general quadratic programming, GGQP). This is a procedure to relax some constraints in some applications and also control the violation of this constraints in order to obtain an initial feasible vector, which is necessary to start the iterative process of the active-set methods.

For the programming part of this work, we start with an implementation of the algorithm already done in FORTRAN (6600 lines of code, approximately), we have studied each routine (except the update of the Cholesky factors) and we have carried out a new implementation in MATLAB. Our code has been compared with the quadratic programming codes of MATLAB and OCTAVE. Our code was superior to them in the nonconvex case (solving some problems that they do not) and all three performed similarly when solving convex quadratic problems.

Key words: General quadratic programming, descent and feasible direction, Cholesky factorization.

Resumen

El objetivo de este trabajo es el estudio y la implementación en MATLAB de un método numérico de conjunto activo para resolver problemas de optimización cuadrática general (GQP), es decir, problemas que incluyen el caso no convexo.

Dada cualquier matriz simétrica, estudiamos la forma de calcular una factorización de Cholesky, total o parcial. Dicha factorización nos permite obtener direcciones de descenso para cualquier matriz Hessiana de la función objetivo, sea definida positiva o no. Además, abordamos la resolución de problemas cuadráticos con términos l_1 en la función objetivo (programación cuadrática general y generalizada, GGQP). Esto permite relajar algunas restricciones (útil para muchas aplicaciones) y controlar su violación para la obtención de un vector inicial admisible, necesario para comenzar el proceso iterativo de los métodos de conjunto activo.

En cuanto a la parte de programación en MATLAB, a partir de una implementación del algoritmo ya realizada en FORTRAN (6600 líneas de código, aproximadamente), hemos estudiado cada rutina (a excepción de las readaptaciones de los factores de Cholesky) y hemos llevado a cabo una nueva programación en MATLAB. Nuestro código ha sido comparado con los de programación cuadrática de MATLAB y OCTAVE (por ser dos paquetes científicos de amplia difusión), mostrándose superior a ellos en la resolución de problemas no convexos (resolviendo problemas que ellos no resuelven),y teniendo un comportamiento muy similar a estos en la resolución de problemas convexos.

Palabras clave: Programación cuadrática general, dirección de descenso admisible, factorización de Cholesky.

Contents

Chapter 1

Introduction

Numerical optimization or mathematical programming is a mathematics branch used in a several sciences to take decisions, whose main goal is minimize or maximize a function that represents the quantity that we want to optimize: profits, distances, energy or any quantitative magnitude. It is called **objective function**. Moreover, the formulation of an optimization problem includes the conditions that its variables have to satisfy. This last part of the problem is called the **feasible set** or the set of constraints. A constraint that is satisfied as equality at a point is called **active constraint** at this point. The active set is particularly important in this framework, as it determines which constraints will influence the final result of the optimization process.

If the problem consists of a quadratic objective function and linear constraints, it is called **quadratic programming problem**. In this report we will be concern about this type of problems. Quadratic functions are a powerful modelling tool in mathematical programming. They appear in various disciplines such as statistics, machine learning (Lasso regression), finance (portfolio optimization), engineering and control theory. Moreover, in mathematical programming, quadratic problems are used for solving the nonlinear ones. General quadratic programming (GQP) covers QP problems including the nonconvex case. Some examples with positive semidefinite Hessian appear in linear least squares problems for data fitting. On the other hand, generalized general quadratic problems (GGQP) allow relaxing some constraints by including l_1 terms in the objective function. (GGQP) formulation is used in some applications as in image restoring problems [11].

To solve optimization problems we have to use iterative algorithms, which means that starting from an initial vector, they will generate a sequence of approximants, each one better than the previous one, until ideally they reach a solution. There are different types of algorithms: active-set methods, inertia-controlling methods, interior point methods or sequential methods, to name a few of the most relevant, and each one uses different strategies to move from one point to the next. We have studied and implemented in MATLAB an active-set method developed by E. Casas and C. Pola in FORTRAN (see for instance [13] or [6]).

The programming work has not been a translation task from FORTRAN to MATLAB,

but it has led to an in-depth study of the approximately 6600 lines of FORTRAN code to make a new implementation taking advantages of the MATLAB facilities. In relation to the code, we have modified some aspects, making an implementation of approximately 1300 lines (excluding comments). The code has been made in MATLAB, which is a software that has a much wider diffusion than FORTRAN nowadays. We have been more concerned with the indefinite Hessian case, paying special attention to the partial Cholesky factorization. In this report we illustrate with examples the different types of factorizations depending on the Hessian, and we analize step by step the process. In terms of computing, it should be noted that we have not updated the Cholesky factors but we have done for the QR factors. On the other hand, we have compared the performance of our code with other softwares: quadprog and qp which are included in two of the most popular numerical packages, MATLAB and OCTAVE, respectively.

The interest of this work lies in the fact that in MATLAB currently there is no function that solves the (GQP) problems with indefinite Hessian and moreover it does not solve the (GGQP) problems. Furthermore, there is no an active set algorithm available for convex cases. On the other hand, OCTAVE cannot solve most of the (GQP) problems and for those that does it is much less efficient than ours and finally, Scilab has the toolbox quapro for solving (GQP) but no for (GGQP) problems and it is not available for all the operating systems.

This report is divided into two chapters excluding this introduction. Chapter 2 is intended for the study of (GQP) problems. In the first three sections we present different results about factorizations and descent directions related to the three types of Hessians: positive definite, singular positive semidefinite and indefinite, providing examples to illustrate the internal process of the factorization algorithm. In this chapter, for the stopping test of the (GQP) algorithm, we consider second order optimality conditions for the nonconvex case and first order optimality conditions for the convex case. We also present the algorithm scheme for solving (GQP). As usual in active set methods it needs to start at a feasible point. Moreover, we present some numerical results and we compare our results with those provided by MATLAB and OCTAVE. In Chapter 3 we focus on the (GGQP) problems. We will make a short introduction to subdifferential calculus for obtaining the corresponding optimality conditions. Last but not least, we introduce an algorithm for solving (GGQP) problems and we use it for computing the initial feasible point for starting the resolution of (GQP) problems.

Chapter 2

General Quadratic Programming (GQP)

In this chapter we are concerned with solving a general quadratic programming problem, (GQP), which can be stated as:

$$(GQP)\begin{cases} \text{Minimize } F(x) = \dfrac{1}{2}x^T H x + p^T x \\ x \in \mathbb{R}^n \\ \text{subject to} \\ \qquad a_j^T x = b_j \ , \quad j = 1, \ldots, n_I; \\ \qquad a_j^T x \leq b_j \ , \quad j = n_I + 1, \ldots, n_I + n_D \ ; \\ \qquad l_j \leq x_j \leq u_j \ , \quad j = 1, \ldots, n \ ; \end{cases}$$

where $H \in \mathbb{R}^{n \times n}$ is a symmetric matrix[1], p and $a_j \in \mathbb{R}^n$, $b_j \in \mathbb{R}$ and l_j, $u_j \in [-\infty, +\infty]$ satisfying $-\infty \leq l_j < u_j \leq +\infty$. In the theorical results the bound constraints will be included in the general constraints.

The difficulty to find a solution for a quadratic problem depends strongly on the characteristics of the objective function, mainly if the Hessian H is positive definite or not. If H is positive definite, then (GQP) has only one solution. In other case, the problem can have several local minimums, if there is any solution. All the solutions of the positive semidefinite case are global. Before starting to study these situations we will give some key definitions to properly follow the development of the chapter.

Taking into account the objective function, a (GQP) algorithm computes some descent directions.

Definition 2.0.1. *A vector $d \in \mathbb{R}^n$ is a **descent direction** in $x \in \mathbb{R}^n$, if there is a number $\overline{\rho} \in \mathbb{R}_+$ such that*

$$F(x + \rho d) < F(x), \ \textit{for all } \rho \in (0, \overline{\rho}).$$

[1]For any matrix $M \in \mathbb{R}^{n \times n}$, $x^T M x = x^T \left(\frac{1}{2}(M + M^T)\right) x$ and $\frac{1}{2}(M + M^T)$ is symmetric.

Proposition 2.0.1. *If $\nabla F(x)^T d < 0$, then d is a descent direction in x.*

Proof. Using the definition of directional derivative, the inequality $\nabla F(x)^T d < 0$ implies that there exists $\overline{\rho} > 0$ such that

$$\frac{F(x + \rho d) - F(x)}{\rho} < 0 \text{ for all } \rho \in (0, \overline{\rho}).$$

From where $F(x + \rho d) < F(x)$ for all $\rho \in (0, \overline{\rho})$. □

There are three types of directions d depending on the type of the curvature:

Definition 2.0.2. *Let be $d \in \mathbb{R}^n$:*

- *It is a **positive curvature direction** if $d^T H d > 0$.*
- *It is a **zero curvature direction** if $d^T H d = 0$.*
- *It is a **negative curvature direction** if $d^T H d < 0$.*

In general quadratic programming no assumptions are made about the matrix H, which implies that the function F can be nonconvex. In the strictly convex case only positive curvature descent directions are computed, but in all other cases both zero and negative as well as positive curvature directions can be used.

We consider an active-set method (see [12] for instance): associated to each feasible point, $x^{(k)}$, we have a subset of constraints, $\mathcal{W}_k$, that are active at $x^{(k)}$: $a_i^T x^{(k)} = b_i$, for all $i \in \mathcal{W}_k$, and such that the vectors $\{a_i\}_{i \in \mathcal{W}_k}$ are linearly independent. The subset $\mathcal{W}_k$ is called **working set**.

Associated to each working set, $\mathcal{W}_k$, we have a matrix A_k with the vectors $\{a_i\}_{i \in \mathcal{W}_k}$ in its columns. Moreover, we have two subspaces:

- The **null space**: $N(A_k^T) = \{z \in \mathbb{R}^n : A_k^T z = 0\}$.
- The **range space**: $R(A_k) = \{z \in \mathbb{R}^n : A_k y = z \text{ for some y} \in \mathbb{R}^{m_k}\}$, where m_k is the number of constraints in $\mathcal{W}_k$.

Definition 2.0.3. *A vector $d \in \mathbb{R}^n$ is a **feasible direction** for a working set $\mathcal{W}_k$, if $d \in N(A_k^T)$.*

Proposition 2.0.2. *If $\mathcal{W}_k$ is a working set at $x^{(k)}$ and $d^{(k)}$ is a feasible direction, then*

$$a_i^T (x^{(k)} + \rho d^{(k)}) = b_i, \text{ for all } i \in \mathcal{W}_k \text{ and for all } \rho \in \mathbb{R}.$$

So, given a point $x^{(k)}$ and a working set $\mathcal{W}_k$, if $x^{(k)}$ does not minimize the function, the method calculates a new point

$$x^{(k+1)} = x^{(k)} + \rho_k d^{(k)}$$

where $d^{(k)}$ is a feasible and descent direction and the steplength $\rho_k > 0$ is a positive value for which all constraints are satisfied at $x^{(k+1)}$.

To obtain feasible directions we consider the following formula

$$d^{(k)} = Z_k d_{Zk}, \tag{2.1}$$

where Z_k is an $n \times (n - m_k)$ matrix, whose columns form a basis of $N(A_k^T)$. The following equality will be useful for this purpose

$$F(x^{(k)} + \rho d^{(k)}) = F(x^{(k)}) + \rho (d^{(k)})^T \nabla F(x^{(k)}) + \frac{1}{2}\rho^2 (d^{(k)})^T H d^{(k)}. \tag{2.2}$$

Using (2.1) in the above equality, we can observe the main role of

$$H_k = Z_k^T H Z_k, \tag{2.3}$$

to decrease the objective function value. The matrix H_k is called **reduced Hessian**.

The next three sections focus on the calculation of a descent direction from a factorization of the reduced Hessian, depending on the type of matrix involved.

2.1 Positive definite reduced Hessian

If the reduced Hessian, H_k, is positive definite, we calculate the direction $d^{(k)}$ solving the linear system

$$H_k d_{Zk} = -Z_k^T \nabla F(x^{(k)}), \tag{2.4}$$

with the Cholesky decomposition of H_k (Theorem A.1.1) and taking $d^{(k)} = Z_k d_{Zk}$.

Algorithm 2.1.1. ***(Cholesky Algorithm)*** *Given a symmetric definite positive matrix $\widetilde{H} \in \mathbb{R}^{n\times n}$ and its Cholesky factorization $\widetilde{H} = \widetilde{L}^T\widetilde{L}$ (see Theorem A.1.1), where $\widetilde{L}$ is an upper triangular matrix, then this factor is computed row by row using the following scheme (see, for example, [8]):*

for i=1,2,...,n

for k=i,i+1,...,n

$$\widetilde{L}(i,i) = \sqrt{\widetilde{H}(i,i) - \sum_{j=1}^{i-1} (\widetilde{L}(i,j))^2};$$

$$\widetilde{L}(i,k) = \frac{1}{\widetilde{L}(i,i)} \left(\widetilde{H}(i,k) - \sum_{j=1}^{k-1} \widetilde{L}(i,j)\widetilde{L}(k,j) \right);$$

end

end.

Now, we prove that $d^{(k)}$ is a descent direction and that we can take the steplength ρ as large as 1. This value is important in the optimization process as we will see in Proposition 2.5.1.

Proposition 2.1.1. *If H_k is positive definite, $Z_k^T \nabla F(x^{(k)}) \neq 0$ and $d^{(k)}$ is calculated by solving the system (2.4), then $d^{(k)}$ verifies*

$$F(x^{(k)} + \rho d) < F(x^{(k)}), \quad \text{for all } \rho \in (0, 1]. \tag{2.5}$$

Proof. If we multiply both sides of (2.4), using (2.1) and (2.3), then

$$\nabla F(x^{(k)})^T d^{(k)} = -(d^{(k)})^T H d^{(k)}.$$

As $Z_k^T \nabla F(x^{(k)}) \neq 0$, the solution of the system (2.4) is a nonzero vector and, so $d^{(k)}$. Hence, using that H is positive definite, we get $-(d^{(k)})^T H d^{(k)} < 0$ and from the above equality $\nabla F(x^{(k)})^T d^{(k)} < 0$.

Taking this into account and using (2.2) and $\rho \in (0, 1]$, we have

$$F(x^{(k)} + \rho d^{(k)}) = F(x^{(k)}) + (\rho - \frac{\rho^2}{2}) \nabla F(x^{(k)})^T d^{(k)} < F(x^{(k)}).$$

□

2.1.1 Numerical results

In this section we calculate a descent direction using the factorization obtained by our code cholpar (see A.4) for a positive definite matrix. In particular with this example, taken from the resolution of the problem presented by Bunch-Kaufman [5], we illustrate that it does not mind if the Hessian H is positive definite or not, what it is important is the reduced Hessian and there are cases where H is indefinite but H_k is positive definite.

- **Matrix factorization**

  Consider

$$H = \begin{pmatrix} 1.69 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 1.69 & 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 1 & 1.69 & 1 & 2 & 3 & 4 & 5 \\ 3 & 2 & 1 & 1.69 & 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 & 1.69 & 1 & 2 & 3 \\ 5 & 4 & 3 & 2 & 1 & 1.69 & 1 & 2 \\ 6 & 5 & 4 & 3 & 2 & 1 & 1.69 & 1 \\ 7 & 6 & 5 & 4 & 3 & 2 & 1 & 1.69 \end{pmatrix} \quad \text{and} \quad Z_k = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

  So the reduced Hessian is

$$Z_k^T H Z_k = \begin{pmatrix} 1.69 & 1 \\ 1 & 1.69 \end{pmatrix}.$$

  The matrix H is indefinite with two negative eigenvalues (-11.44707, -2.52418). Nonetheless, the reduced Hessian, H_k, is a symmetric positive definite matrix. Thus, we can apply directly the Cholesky formulas (see Algorithm (2.1.1)) and we get

$$L^T = \begin{pmatrix} 1.3000 & 0.7692 \\ 0 & 1.0480 \end{pmatrix}.$$

- **Calculation of a descent direction**

  Taking $\nabla F(x^{(k)}) = (1,2)^T$, we get, solving the system (2.4), the direction

$$d^{(k)} = \begin{pmatrix} 0.1670169 \\ -1.2822585 \end{pmatrix}.$$

  Now we check if this direction is a descent direction: $\nabla F(x^{(k)})^T d^{(k)} = -2.3975 < 0$.

2.2 Positive semidefinite and singular reduced Hessian

In this section we present a theorem which establishes the existence of a Cholesky factorization for matrices which are positive semidefinite and singular [2](see [6]). In the numerical resolution of the problem (GQP), we will apply this result to the reduced Hessian (which will be denoted by $\widetilde{H}$ in the following theorem).

Theorem 2.2.1. *A nonzero matrix $\widetilde{H} \in \mathbb{R}^{n\times n}$ is positive semidefinite and singular if and only if there exists a permutation matrix P such that*

$$P\widetilde{H}P^T = \begin{pmatrix} L & 0 \\ B & 0 \end{pmatrix} \begin{pmatrix} L^T & B^T \\ 0 & 0 \end{pmatrix} \tag{2.6}$$

where $L \in \mathbb{R}^{m\times m}$ is a lower triangular matrix with strictly positive diagonal elements and $B \in \mathbb{R}^{(n-m)\times m}$. The rank of $\widetilde{H}$ is m and a basis of the null space of $\widetilde{H}$ is formed by the vectors $\{P^T u_j\}_{j=1}^{n-m}$ defined by

$$u_j = \begin{pmatrix} \hat{u}_j \\ 0 \end{pmatrix} - e_{m+j}, \tag{2.7}$$

where e_{m+j} is the (m+j)—th column of the identity matrix $Id_{n\times n}$ and $\hat{u}_j$ is the m-vector solution of the system $L^T\hat{u}_j = B_j^T$ with B_j the j-th row of B.

Proof. If we have the factorization (2.6) it is easy to see that $\widetilde{H}$ is positive semidefinite. If $d \in \mathbb{R}^n$, then taking $\widetilde{d} = P^T d$

$$\widetilde{d}^T \begin{pmatrix} L & 0 \\ B & 0 \end{pmatrix} \begin{pmatrix} L^T & B^T \\ 0 & 0 \end{pmatrix} \widetilde{d} = \left[\begin{pmatrix} L^T & B^T \\ 0 & 0 \end{pmatrix} \widetilde{d} \right]^T \begin{pmatrix} L^T & B^T \\ 0 & 0 \end{pmatrix} \widetilde{d} = \left\| \begin{pmatrix} L^T & B^T \\ 0 & 0 \end{pmatrix} \widetilde{d} \right\|_2^2 \geq 0.$$

To see the reciprocal we use an algorithm that determines if $\widetilde{H}$ is positive definite, positive semidefinite and singular or indefinite. The algorithm returns the Cholesky decomposition in the first case, the factorization (2.6) in the second one and a partial factorization in the third case.

[2]all eigenvalues are nonnegative and at least one of them is zero.

Factorization Algorithm

1. *Set k=1,* $\widetilde{H}^{(k)} = \left(h_{ij}^{(k)}\right) = \widetilde{H}$, $P^{(k)} = Id$, *where* Id *is the identity matrix and*

$$\beta = 1.2 \cdot (\max\{|h_{jj}| : j = 1, ..., n\})^{1/2}.$$

2. *Find q such that*

$$h_{qq}^{(k)} = \max\{h_{jj}^{(k)} : j = k, ..., n\}.$$

- *If* $h_{qq}^{(k)} < 0$

$$\textbf{\textit{Indefinite matrix. STOP the algorithm}} \tag{2.8}$$

- *If* $h_{qq}^{(k)} = 0 \Longrightarrow$ *Find t such that*

$$h_{tt}^{(k)} = \min\{h_{jj}^{(k)} : j = k, ..., n\}.$$

  - *If* $h_{tt}^{(k)} < 0$

$$\textbf{\textit{Indefinite matrix. STOP the algorithm}} \tag{2.9}$$

  - *If* $h_{tt}^{(k)} = 0$
    - *If* $j = n$

$$\textbf{\textit{End of factorization. STOP the algorithm.}} \tag{2.10}$$

    - *Else find r and s such that*

$$|h_{rs}^{(k)}| = max\{|h_{ij}^{(k)}| : n \geq i > j \geq k\}.$$

      - *if* $|h_{rs}^{(k)}| \neq 0$

$$\textbf{\textit{Indefinite matrix. STOP the algorithm.}}$$

      - *if* $|h_{rs}^{(k)}| = 0$

$$\textbf{\textit{End of factorization. STOP the algorithm.}} \tag{2.11}$$

- *If* $h_{qq}^{(k)} > 0 \Longrightarrow$ *Interchange the rows q and k of* $\widetilde{H}^{(k)}$ *and* $P^{(k)}$*. The new matrices will be denoted by* $\widetilde{H}^{(k)}$ *and* $P^{(k+1)}$ *respectively.*

3. *Now the elements of the k-th column of the lower triangular factor are calculated. For j=1 to k-1*

$$h_{ij}^{(k+1)} = h_{ij}^{(k)}, \quad i=j,...,n,$$

$$h_{kk}^{(k+1)} = \sqrt{h_{kk}^{(k)}}.$$

If k=n

End of factorization. STOP the algorithm.

If $k < n \Longrightarrow$ *Continue.*

For i=k+1 to n

$$h_{ik}^{(k+1)} = \frac{h_{ik}^{(k)}}{h_{kk}^{(k+1)}}.$$

If $\max\{|h_{ij}^{(k+1)}| : i = k+1, ..., n\} > \beta \Longrightarrow$ ***Indefinite Matrix.STOP the algorithm***.

Now the elements to the right of the k-th column of the factor are updated using the Cholesky's formulas and the last calculated elements.

For j=k+1 to n

$$h_{jj}^{(k+1)} = h_{jj}^{(k)} - \left(h_{jk}^{(k+1)}\right)^2, \tag{2.12}$$

$$h_{ij}^{(k+1)} = h_{ij}^{(k)} - h_{ik}^{(k+1)} h_{jk}^{(k+1)}, \;\; i = j+1, \ldots, n.$$

4. Set k=k+1. Go to step 2.

Firstly if the algorithm stops in Step 3 with k=n,we will have the Cholesky factorization:

$$P\widetilde{H}P^T = LL^T, \tag{2.13}$$

where $P = P^{(n+1)}$ and L is a lower triangular matrix with $l_{ii} > 0$ for i=1,. . .,n. Thus, in this case $\widetilde{H}$ is a positive definite matrix.

On the other hand, if the algorithm ends in the Step 2, in (2.10) or (2.11), the factorization (2.6) is obtained.

If the algorithm stops at (2.8) or (2.9) with $k = 1$, some diagonal element is negative and obviously the matrix is indefinite.

In the other cases the algorithm stops with the following factorization

$$P\widetilde{H}P^T = \begin{pmatrix} L & 0 \\ B & I_{n-m} \end{pmatrix} \begin{pmatrix} I_m & 0 \\ 0 & D \end{pmatrix} \begin{pmatrix} L^T & B^T \\ 0 & I_{n-m} \end{pmatrix}, \tag{2.14}$$

where $m = k-1$, $L \in \mathbb{R}^{m\times m}$ is a nonsingular lower triangular matrix being $l_{ij} = h_{ij}^{(k)}$,$1 \leq j \leq i \leq m$, $B \in \mathbb{R}^{(n-m)\times m}$ with $b_{i,j} = h_{m+i,j}^{(k)}$, $1 \leq i \leq n-m$, $1 \leq j \leq m$ and $D \in \mathbb{R}^{(n-m)\times(n-m)}$ is a symmetric matrix with $d_{i,j} = h_{m+i,m+j}^{(k)}$, $1 \leq j \leq i \leq n-m$. We will see that in this case there are negative curvature directions.

- First we suppose that $h_{jj}^{(k)} < 0$, for some index such that $k \leq j \leq n$.

  Let B_{j-m} be the (j-m)-th row of the matrix B and $\hat{u}$ an m-dimensional vector obtained from $L^T\hat{u} = -B_{j-m}^T$. Then, we calculate a descent direction of negative curvature whose form is P^Tu where

$$u = \begin{pmatrix} \hat{u} \\ 0 \end{pmatrix} + e_j, \tag{2.15}$$

  and e_j is the j-th vector of the canonical basis.

  Now, taking into account the following equalities

$$\begin{pmatrix} L^T & B^T \\ 0 & I_{n-m} \end{pmatrix} u = \begin{pmatrix} L^T\hat{u} + B_{j-m}^T \\ 0 \end{pmatrix} + e_j = e_j.$$

  It follows that

$$(P^T u)^T \widetilde{H}(P^T u) = e_j^T \begin{pmatrix} I_m & 0 \\ 0 & D \end{pmatrix} e_j = d_{j-m,j-m} = h_{jj}^{(k)} < 0. \tag{2.16}$$

  So P^Tu is a negative curvature direction of $\widetilde{H}$.

- Now we consider the case $h_{jj}^{(k)} = 0$, for all $j = k, \ldots, n$ and $h_{rs}^{(k)} \neq 0$, $n \geq r > s \geq k$.

  Be B_{s-m} and B_{r-m} the rows s-m and r-m of the matrix B respectively. We define the vector

$$u = \begin{pmatrix} \hat{u} \\ 0 \end{pmatrix} - h_{rs}^{(k)} e_s + e_r, \tag{2.17}$$

  where $\hat{u}$ is the solution of the system $L^T\hat{u} = -B_{r-m}^T + h_{rs}^{(k)} B_{s-m}^T$, while the vectors e_r and e_s are the r-th and s-th vectors of the canonical basis of $\mathbb{R}^n$, respectively. Then,

$$\begin{pmatrix} L^T & B^T \\ 0 & I_{n-m} \end{pmatrix} u = \begin{pmatrix} L^T\hat{u} + B_{r-m}^T - h_{rs}^{(k)} B_{s-m}^T \\ 0 \end{pmatrix} - h_{rs}^{(k)} e_s + e_r = -h_{rs}^{(k)} e_s + e_r.$$

  So using this equality, it is easy to see

$$\begin{aligned} (P^T u)^T \widetilde{H}(P^T u) &= (-h_{rs}^{(k)} e_s + e_r)^T \begin{pmatrix} I_m & 0 \\ 0 & D \end{pmatrix} (-h_{rs}^{(k)} e_s + e_r) = \\ &= (h_{rs}^{(k)})^2 d_{s-m,s-m} + d_{r-m,r-m} - 2h_{rs}^{(k)} d_{r-m,s-m} = \\ &= (h_{rs}^{(k)})^2 h_{ss}^{(k)} + h_{rr}^{(k)} - 2(h_{rs}^{(k)})^2 = -2(h_{rs}^{(k)})^2 < 0. \end{aligned} \tag{2.18}$$

  So $P^T u$ is a negative curvature direction.

- If the algorithm stops because $|h_{ik}^{(k)}| > \beta$ for some i, $k+1 \leq i \leq n$, it is only necessary to note that if we calculate $h_{ii}^{(k+1)}$, this will be negative. Using (2.12)

$$h_{ii}^{(k+1)} = h_{ii}^{(k)} - (h_{ik}^{(k+1)})^2 < h_{ii}^{(k)} - \beta^2 \leq h_{ii}^{(k)} - \beta^2 < 0.$$

  So, we are again in a situation with some negative diagonal element and in this case we have seen before that a negative curvature direction exists.

So, if the matrix $\widetilde{H}$ is positive semidefinite and singular, only the factorization (2.6) is possible.

Finally, we have to verify that $\{ P^T u_j\}_{j=1}^{n-m}$ is a basis of the null space of $\widetilde{H}$. Using that P is orthogonal, i.e. $P^T P = I$, it is possible to write $\widetilde{H}$ as

$$\widetilde{H} = P^T \begin{pmatrix} L & 0 \\ B & 0 \end{pmatrix} \begin{pmatrix} L^T & B^T \\ 0 & 0 \end{pmatrix} P. \tag{2.19}$$

It follows that the rank of $\widetilde{H}$ is the same as that of L. Thus, every basis of the null space of H_k has $n - m$ elements.

On the other side using the definition of u_j we have

$$\begin{pmatrix} L^T & B^T \\ 0 & 0 \end{pmatrix} P P^T u_j = \begin{pmatrix} L^T \hat{u}_j - B_j^T \\ 0 \end{pmatrix} = 0.$$

So, using (2.19), $\widetilde{H} P^T u_j$=0. The linear independence follows from the definition of $\{P^T u_j\}_{j=1}^{n-m}$. □

Now, as a consequence of the Theorem 2.2.1 we present some results about the descent direction.

Observation 2.2.1. *In practice we will try to factorize as much as posible before starting the pivotation. That is, until the code detect that the matrix is nonpositive definite, it does not start to exchange elements.*

Corollary 2.2.1. *Be $H_k = Z_k^T H Z_k \in \mathbb{R}^{n_k \times n_k}$ a singular positive semidefinite reduced Hessian associated with $x^{(k)}$, and its factorization given by*

$$H_k = P^T \begin{pmatrix} L & 0 \\ B & 0 \end{pmatrix} \begin{pmatrix} L^T & B^T \\ 0 & 0 \end{pmatrix} P. \tag{2.20}$$

where P, L and B are as in the Theorem 2.2.1.

If U is a matrix whose columns are the vectors $P^T u_j$, j=1,..., $n_k - m$, where u_j is given by (2.7) and

$$\hat{d} = -Z_k U U^T Z_k^T \nabla F(x^{(k)}), \tag{2.21}$$

the following statements are verified:

a) If $\hat{d} \neq 0$, then $\hat{d}$ is a null curvature descent direction.

b) Otherwise, there exists a vector $d_{Zk}^{(k)}$ solution of the system (2.4) and $d^{(k)} = Z_k d_{Zk}^{(k)}$ is a positive curvature descent direction.

Proof. a) Using that the columns of U are a basis of the null space of H_k and (2.21), then

$$\hat{d}^T H \hat{d} = 0. \tag{2.22}$$

So, $\hat{d}$ is a null curvature direction for H. Moreover, using the hypotesis $\hat{d} \neq 0$, we also have

$$(\hat{d})^T \nabla F(x^{(k)}) = -||(Z_k U)^T \nabla F(x^{(k)})||_2^2 < 0. \tag{2.23}$$

Using (2.22) and (2.23) together with (2.2) we have that $\hat{d}$ is also a descent direction.

b) We assume that $\hat{d} = 0$ and we will see that there exists a solution for the system (2.4). Using the linear independence of the columns of the matrices Z_k and U, from $\hat{d} = 0$ we conclude that

$$U^T Z_k^T \nabla F(x^{(k)}) = 0.$$

This means that $Z_k^T \nabla F(x^{(k)})$ is orthogonal to a basis of the null space $N(H_k^T)$, using that H_k is symmetric. So, $Z_k^T \nabla F(x^{(k)}) \in R(H_k)$, which implies that

$$PZ_k^T \nabla F(x^{(k)}) \in R(PH_k) = R(PH_kP^T)$$

and, using the factorization (2.20), $PZ_k^T \nabla F(x^{(k)}) \in R \begin{pmatrix} L & 0 \\ B & 0 \end{pmatrix}$ and, as L is a regular matrix, there is only one vector v such that

$$\begin{pmatrix} L & 0 \\ B & 0 \end{pmatrix} \begin{pmatrix} v \\ 0 \end{pmatrix} = -PZ_k^T \nabla F(x^{(k)}). \tag{2.24}$$

Taking $\widetilde{v}$ such that $L^T \widetilde{v} = v$, then $d_{Zk}^{(k)} = P^T \begin{pmatrix} \widetilde{v} \\ 0 \end{pmatrix}$ is a solution of the system (2.4):

$$\begin{aligned} H_k P^T \begin{pmatrix} \widetilde{v} \\ 0 \end{pmatrix} &= P^T \begin{pmatrix} L & 0 \\ B & 0 \end{pmatrix} \begin{pmatrix} L & 0 \\ B & 0 \end{pmatrix}^T PP^T \begin{pmatrix} \widetilde{v} \\ 0 \end{pmatrix} = \\ &= P^T \begin{pmatrix} L & 0 \\ B & 0 \end{pmatrix} \begin{pmatrix} L^T \widetilde{v} \\ 0 \end{pmatrix} = P^T \begin{pmatrix} L & 0 \\ B & 0 \end{pmatrix} \begin{pmatrix} v \\ 0 \end{pmatrix} = -Z_k^T \nabla F(x^{(k)}). \end{aligned}$$

Now, we check that $Z_k d_{Zk}^{(k)}$ is also a positive curvature direction

$$\begin{aligned} d_{Zk}^T Z_k^T H Z_k d_{Zk} &= -d_{Zk}^T Z_k^T \nabla f(x^{(k)}) = -(\widetilde{v}^T 0) PZ_k^T \nabla F(x^{(k)}) = \\ &= (\widetilde{v}^T 0) \begin{pmatrix} L & 0 \\ B & 0 \end{pmatrix} \begin{pmatrix} v \\ 0 \end{pmatrix} = \widetilde{v}^T LL^T \widetilde{v} = \left\| L^T \widetilde{v} \right\|_2^2 = \left\| v \right\|_2^2 \geq 0, \end{aligned}$$

taking into account (2.24) and $Z^T \nabla f(x^{(k)}) \neq 0$.

□

2.2.1 Numerical results

In this subsection we consider some examples to illustrate step by step how our code cholpar (see A.4) calculates the factorization (2.20) for a singular positive semidefinite reduced Hessian. In fact, the code calculates the factor on the right, $\widetilde{L}$. We will also calculate positive or null curvature directions.

The pivotation starts when the first nonpositive radicant of the Cholesky formulas (see Algorithm (2.1.1)) appears in the process.

Note 2.2.1. *We will use a vector, ipvt, to store the information of the matrix P. Given a vector y*

$$ipvt(i) = j \quad means \quad (Py)_i = y_j$$

- **Factorization of a singular reduced Hessian**.
  We consider the following matrix

$$\widetilde{H} = \begin{pmatrix} 4 & -2 & 2 & 2 \\ -2 & 2 & 2 & 1 \\ 2 & 2 & 10 & 7 \\ 2 & 1 & 7 & 5 \end{pmatrix}. \tag{2.25}$$

  The eigenvalues of this matrix $\widetilde{H}$ are $\{0, 0, 5.1849, 15.8151\}$, so, the matrix is positive semidefinite and singular.

  - **Iteration 1**[3]
    We inicialize the pivotation vector: ipvt=(1,2,3,4).
    We start with the diagonal elements of $\widetilde{H}$ and as $\widetilde{H}(1,1) > 0$, we calculate the diagonal element $\widetilde{L}(1,1)$=2 and the elements in the first row, using the Cholesky formulas (see Algorithm (2.1.1))

$$\widetilde{L} = \begin{pmatrix} 2 & -1 & 1 & 1 \\ & - & - & - \\ & & - & - \\ & & & - \end{pmatrix}.$$

    The k-th row will remain unchanged for the rest of the process.
  - **Iteration 2**
    For the element $\widetilde{L}(2,2)$, we first calculates its radicant and as it is positive, following the algorithm, we calculate the rest of the elements in the second row.

$$\widetilde{L} = \begin{pmatrix} 2 & -1 & 1 & 1 \\ & 1 & 3 & 2 \\ & & - & - \\ & & & - \end{pmatrix}.$$

  - **Iteration 3**
    Using the Cholsesky formulas, we get $\widetilde{L}(3,3) = 0$. Since it is not positive the rest of the elements on the diagonal are calculated: $\widetilde{L}(4,4) = 0$. And now we have to find $\max\{\widetilde{L}(j,i); k \leq j < i \leq n\}$. For our example, using again the Cholesky formulas: $\widetilde{L}(3,4) = 0$, so the factorization is finished with

$$\widetilde{L} = \begin{pmatrix} 2 & -1 & 1 & 1 \\ 0 & 1 & 3 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}; \quad \text{ipvt} = (1,2,3,4).$$

[3]In the k-th iteration the k-th diagonal element for $\widetilde{L}$ is chosen and k-1 is the size of the factorized matrix.

If we compare with the factorization (2.20), for our example, we get the following submatrices

$$L = \begin{pmatrix} 2 & 0 \\ -1 & 1 \end{pmatrix}; \quad \text{and} \quad B = \begin{pmatrix} 1 & 3 \\ 1 & 2 \end{pmatrix};$$

- **Calculation of a descent direction**
  - Here we take $\nabla F(x^{(k)})$=(1,2,3,4)^T and $Z_k = Id$.
    The reduced Hessian H_k, is positive semidefinite and singular, so we calculate the descent direction as Corollary 2.2.1 states. Using (2.21), we obtain

$$\hat{d} = \begin{pmatrix} -12.2500 \\ -18.0000 \\ 5.0000 \\ 1.5000 \end{pmatrix}.$$

    As $\hat{d} \neq 0$, we take $d^{(k)} = \hat{d}$. We check that this direction meets the requirements
    1. $(d^{(k)})^T \nabla F(x^{(k)})$=-27.2500 <0. So, it is a descent direction.
    2. $(d^{(k)})^T H_k d^{(k)}$=0. So, it is a null curvature direction.
  - Now we choose a different gradient vector: $\nabla F(x^{(k)})$=(1,-1.5,-2.5,-1.5)^T, and the same matrix Z_k. Repeating the same steps that for the previous case, using (2.21), we have $\hat{d} = 0$. So, the system $H_k d_{Zk} = -\widetilde{Z} \nabla F(x^{(k)})$ has to be solved to obtain the direction $d_{Zk}^{(k)}$ and, then $d^{(k)} = \widetilde{Z} d_{Zk}^{(k)}$. In our example, $v = (-0.5, 1)^T$, $\widetilde{v} = (0.25, 1)^T$, $d_{Zk}^{(k)} = (0.25, 1, 0, 0)^T$ and finally

$$d^{(k)} = \begin{pmatrix} 0.2500 \\ 1.0000 \\ 0 \\ 0 \end{pmatrix}.$$

    We check that this direction meets the requirements
    1. $(d^{(k)})^T \nabla F(x^{(k)})$=-1.2500 <0. So, it is a descent direction.
    2. $(d^{(k)})^T H_k d^{(k)}$=1.2500. So, it is a positive curvature direction.

Observation 2.2.2. *It is important underline the fact that GQP algorithm can work with matrices which have two or more null eigenvalues. This is not the case for the inertia-controlling methods, which only allow one nonpositive eigenvalue (see, for example, [9]).*

2.3 Indefinite reduced Hessian

If the reduced Hessian is indefinite, we have the following factorization

$$P H_k P^T = \begin{pmatrix} L & 0 \\ B & I_{n-m} \end{pmatrix} \begin{pmatrix} I_m & 0 \\ 0 & D \end{pmatrix} \begin{pmatrix} L^T & B^T \\ 0 & I_{n-m} \end{pmatrix}, \tag{2.26}$$

Now we give a result to have a formula to compute the direction in the indefinite reduced Hessian case.

Corollary 2.3.1. *If H_k is indefinite, $\sigma = -sign((Z_k P^T u)^T \nabla F(x^{(k)}))$, P is the permutation matrix associated to the factorization* (2.26) *and u is given by* (2.15) *or* (2.17)*, then $d^{(k)} = \sigma Z_k P^T u$, is a negative curvature direction for the Hessian H and a descent direction for the objetive function F at $x^{(k)}$.*[4]

Proof. Taking $\widetilde{H} = Z_k^T H Z_k$ as in Theorem 2.2.1, using (2.16) and (2.18) we conclude that $d^{(k)}$ is a negative curvature direction for H: $d^{(k)} H d^{(k)} < 0$. Moreover, using the formula of $d^{(k)}$, $\nabla F(x^{(k)}) d^{(k)} < 0$. Hence, using (2.2)

$$F(x^{(k)} + \rho d^{(k)}) < F(x^{(k)}), \ \ \forall \rho > 0.$$

Therefore, $d^{(k)}$ is a negative curvature direction for H and a descent direction for the objective function. □

2.3.1 Numerical results

In this subsection, we illustrate in detail how our code calculates the factorization of a matrix with one negative eigenvalue.

- **Matriz factorization**
  We consider the following matrix

$$\widetilde{H} = \begin{pmatrix} 7.3148 & -2.7420 & 1.6468 & -0.3105 \\ -2.7420 & -1.1770 & 5.7882 & 0.9989 \\ 1.6468 & 5.7882 & 1.9622 & -3.6600 \\ -0.3105 & 0.9989 & -3.6600 & 5.900 \end{pmatrix}.$$

  The eigenvalues of $\widetilde{H}$ are $\{8, 4, 9, -7\}$, i.e., the matrix is indefinite, therefore, it will not be possible to factorize the matrix completely with the Cholesky decomposition. The idea is to factorize as large part as possible without pivoting.
  We start by calculating the factorization of the matrix $\widetilde{H}$ as in Subsection 2.2.1.
  - **Iteration 1**
    We initialize the pivoting vectot $ipvt = (1, 2, 3, 4)^T$.
    We start with the diagonal elements of $\widetilde{H}$ and as $\widetilde{H}(1,1) > 0$, we calculate the elements in the first row.

$$\widetilde{L} = \begin{pmatrix} 2.7046 & -1.0138 & 0.6089 & -0.1148 \\ & - & - & - \\ & & - & - \\ & & & - \end{pmatrix}.$$

  - **Iteration 2** In this case we calculate the element $\widetilde{L}(2,2) = -2.2049 < 0$, so in this situation, we calculate the rest of the diagonal elements in the same way.

$$\widetilde{L} = \begin{pmatrix} 2.7046 & -1.0138 & 0.6089 & -0.1148 \\ & \textbf{-2.2049} & - & - \\ & & 1.5915 & - \\ & & & 5.8868 \end{pmatrix}.$$

[4]The function sign: $\mathbb{R} \longrightarrow \{-1, 1\}$ is defined by: sign(x)=1 if $x \geq 0$ and sign(x)=-1 if $x < 0$.

As the Cholesky factorization can no longer be applied, the permutation is started. The first step is to look for the largest element of the diagonal in the unfactorized part of the matrix, which turns out to be $\widetilde{L}(4,4)$=5.8868. Then, we permute the fourth row with the second row and we do the same permutation with second and fourth columns.
In this iteration we calculate the elements in the second row and modified the diagonal elements using the Cholesky formulas (see Algorithm (2.1.1)).
This iteration finish with the following

$$\widetilde{L} = \begin{pmatrix} 2.7046 & -0.1148 & 0.6089 & -1.0138 \\ & 2.4263 & -1.4797 & 0.3637 \\ & & \mathbf{-0.5980} & - \\ & & & \mathbf{-2.3372} \end{pmatrix}; \quad \text{ipvt=(1,4,3,2).}$$

– **Iteration 3**
All the elements in the diagonal are negative in the unfactorized matrix. We search for the minimum of them and the rows and columns needed to place it in the position (3,3) are permuted. In this case it involves the second and third rows and columns obtaining

$$\widetilde{L} = \begin{pmatrix} 2.7046 & -0.1148 & -1.0138 & 0.6089 \\ 0 & 2.4263 & 0.3637 & -1.4797 \\ 0 & 0 & \mathbf{-2.3372} & * \\ 0 & 0 & * & \mathbf{-0.5980} \end{pmatrix} \quad \text{and} \quad \text{ipvt(1,4,2,3).}$$

This factorization has the same estructure as the represented in (2.26), with

$$P^T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}; L = \begin{pmatrix} 2.7046 & 0 \\ -1.0138 & 2.4263 \end{pmatrix};$$

$$B = \begin{pmatrix} -0.1148 & 0.3637 \\ 0.6089 & -1.4797 \end{pmatrix}; D = \begin{pmatrix} -2.3372 & * \\ * & -0.5980 \end{pmatrix};$$

- **Calculation of a descent direction**
We take $\nabla F(x^{(k)}) = (1,2,3,4)^T$ and Z_k the identity matrix. The direction obtained is:

$$d^{(k)} = \begin{pmatrix} -0.36849 \\ -1 \\ 0 \\ 0.14990 \end{pmatrix}$$

We check that this direction meets the requirements:

1. $(d^{(k)})^T \nabla F(x^{(k)}) = -1.768876 < 0$. So, it is a decent direction.
2. $(d^{(k)})^T H_k d^{(k)} = -2.337154$. So, it is a negative curvature direction.

2.4 Optimality conditions

These conditions provide a mechanism to stop the algorithms and a criteria to decide when a vector is a solution to the optimization problems.

In the general quadratic programming framework it is posible that the hypothesis of the sufficient first-order optimality conditions (see Theorem A.2.1) do not hold true, because if the matrix H has some negative eigenvalue the function is not convex. In this situation we have to use the second-order optimality conditions [7]. In the following result we give the optimality conditions used by our code.

Theorem 2.4.1. *Be $\overline{x} \in \mathbb{R}^n$ a feasible point for the problem (GQP) and $\overline{\lambda} \in \mathbb{R}^{n_I+n_D}$ the Lagrange multiplier vector satisfying* (A.1),(A.2) *and* (A.3). *Be $\overline{\mathcal{W}}$ the index set corresponding to the equality constraints and also the inequality constraints with strictly positive Lagrange multiplier:*

$$\overline{\mathcal{W}} = \{j : j = 1, \ldots, n_I\} \cup \{j : j > n_I \text{ and } \lambda_j > 0\}.$$

Be $\overline{A}$ the working set matrix: $\overline{A} = (a_j)_{j\in\overline{\mathcal{W}}} \in \mathbb{R}^{(n\times m)}$, and $\overline{Z} \in \mathbb{R}^{n\times(n-m)}$ with full rank and such that $\overline{A}^T\overline{Z} = 0$.

If m=n or $\overline{Z}^T H\overline{Z}$ is positive semidefinite, then $\overline{x}$ is a local minimum for the (GQP) problem.

Observation 2.4.1. *An important detail is that we need that the Lagrange multipliers associated with inequality constraints be positive to ensure that $\overline{x}$ is a local minimum. We illustrate this in the following example from [6]:*

$$\begin{cases} Min\ F(x) = x_3^2 - 2x_1x_2 \\ subject\ to \\ \quad 0 \le x_1 + x_2 \le 2 \\ \quad x_1 - x_2 \le 2 \end{cases}.$$

Let $\overline{x} = (-1, 1, 0)^T$ and $\lambda = (0, 2)^T$. Then

$$Z_k^T H_k = (0, 0, 1) \begin{pmatrix} 0 & -2 & 0 \\ -2 & 0 & 0 \\ 0 & 0 & 2 \end{pmatrix} = 2$$

is positive definite. However, taking $\epsilon \neq 0$ and $x_\epsilon = (\epsilon - 1, \epsilon + 1, 0)^T$, $F(x_\epsilon) = 2(1 - \epsilon^2) < 2 = F(\overline{x})$. So, $\overline{x}$ is not a local minimum.

2.4.1 Numerical method for computing a basis of null space

Following [10], we calculate a basis for the subspace $N(A_k^T)$ using the factorization QR. We supose that there are m_k constraints in the working set: $A_k \in \mathbb{R}^{n\times m_k}$, and we consider

the QR factorization for the matrix A_k:

$$A_k = Q_k R_k, \tag{2.27}$$

where $Q_k \in \mathbb{R}^{n\times n}$ is an orthogonal matrix and $R_k \in \mathbb{R}^{n\times m_k}$ an upper triangular matrix.

We distinguish the following submatrices in Q_k: the first m_k columns, Y_k, and the rest of the columns, Z_k. On the other hand, in R_k we extract from the first m_k rows, an upper triangular matrix $\hat{R}_k$.

Proposition 2.4.1. *It is verified:*

a) The column set of Y_k is a basis of $R(A_k)$.

b) The column set of Z_k is a basis of $N(A_k^T)$.

Proof. Since Q_k is orthogonal, its columns are linearly independents and so the columns of Y_k and Z_k are linearly independents. Moreover, $A_k = Q_k R_k = Y_k \hat{R}_k$, so the columns of Y_k are a generator system of $R(A_k)$ and, as $Q_k^T Q_k = Id$, $Y_k^T Z_k = 0$ and therefore, the columns of Z_k belongs to $N(A_k^T)$.

2.5 An algorithm for GQP

At this point we present the algorithm scheme used to solve the problem (GQP).

Algorithm for general quadratic programming

1. *Set k=0. Calculate an initial feasible vector, $x^{(0)}$, the initial working set, $\mathcal{W}_0$, its QR factorization and the total or partial Cholesky factorization of the first reduced Hessian, H_k.*
2. - *If $Z_k^T \nabla F(x^{(k)}) \neq 0$ or H_k is not positive semidefinite, go to Step 3.*
   - *In other case, calculate Lagrange multiplier candidates solving the linear system*

$$\hat{R}_k \lambda = -Y_k^T \nabla f(x^{(k)}) \tag{2.28}$$

     - *If all the Lagrange multipliers candidates associated with inequality constraints of $\mathcal{W}_k$ are strictly positive, then $x^{(k)}$ is a local minimum. STOP.*
     - *Else remove from $\mathcal{W}_k$ the inequality constraint associated with the minimum multiplier, update the QR factorization and compute the descomposition of the new reduced Hessian. Go to Step 3.*
3. *Calculate a descent direction:*
   - *If H_k is a positive definite matrix, calculate $d_{Zk}^{(k)}$ solving the system (2.4) with the Cholesky decomposition. Take $d^{(k)} = Z_k d_{Zk}^{(k)}$. Go to Step 4.*
   - *If H_k is a positive semidefinite matrix, calculate $\hat{d}$ from (2.21):*
     - *If $\hat{d} \neq 0, d^{(k)} = \hat{d}$. Go to Step 5.*
     - *If $\hat{d} = 0$, solve the system (2.4) with factorization (2.6) and take $d^{(k)} = Z_k d_{Zk}^{(k)}$. Go to Step 4.*

- *If H_k is an indefinite matrix, calculate a negative curvature descent direction using* (2.15) *or* (2.17)*. Go to Step 5.*

4. *Calculate ρ_k using the following formula*

$$\rho_k = \min\left\{1, \min_{j\notin\mathcal{W}_k, a_j^T d^{(k)}>0} \frac{b_j - a_j^T x^{(k)}}{a_j^T d^{(k)}}\right\}. \tag{2.29}$$

Take $x^{(k+1)} = x^{(k)} + \rho_k d^{(k)}$ and k=k+1. If ρ_k=1 and it is not associated with any constraint, then go to Step 2, in other case go to Step 6.

5. *Calculate ρ_k using the next formula*

$$\rho_k = \min_{j\notin\mathcal{W}_k, a_j^T d^{(k)}>0} \frac{b_j - a_j^T x^{(k)}}{a_j^T d^{(k)}}. \tag{2.30}$$

Take $x^{(k+1)} = x^{(k)} + \rho_k d^{(k)}$ and k=k+1. If the working set $\mathcal{W}_k$ is empty, then the quadratic problem is unbounded in the feasible region. STOP.

6. *If the lengthstep ρ_k is associated with a constraint whose index is j_k, add j_k to the working set $\mathcal{W}_k$. Modify QR factorization and compute the new factors for the new reduced Hessian. Go to Step 2.*

Note 2.5.1. *In the Step 2 we do not calculate Lagrange multipliers candidates when H_k is not positive semidefinite because, using the necessary second-order optimality conditions (Theorem A.2.2), we know that $x^{(k)}$ is not a solution for the optimization problem.*

Now we will give a result which justify the importance of the case $\rho_k = 1$, when H_k is positive semidefinite and $d^{(k)}$ is obtained by solving the system (2.4).

Proposition 2.5.1. *Let the matrix H_k be positive semidefinite and $d^{(k)}$ a solution of* (2.4)*. If $x^{(k)}$ is a feasible point, then $x^{(k)} + d^{(k)}$ is a global solution of the following problem*

$$(P)_{\mathcal{W}_k}\begin{cases} Min\ F(x) \\ x\in\mathbb{R}^n \\ subject\ to \\ \quad a_j^T x = b_j, j\in\mathcal{W}_k \\ \quad a_j^T x \le b_j, j\notin\mathcal{W}_k \end{cases}.$$

Proof. First of all we study a similar problem but only with equality constraints:

$$(P)_{Eq}\begin{cases} \text{Min } F(x) \\ x\in\mathbb{R}^n \\ \text{subject to} \\ \quad a_j^T x = b_j, j\in\mathcal{W}_k \end{cases}.$$

Its feasible set verifies:

$$\{x\in\mathbb{R}^n : a_j^T x = b, j\in\mathcal{W}_k\} = \{x^{(k)} + d : a_j^T d = 0, j\in\mathcal{W}_k\} = \{x^{(k)} + Z_k y : y\in\mathbb{R}^{n-m_k}\}.$$

If we also use (2.2), $(P)_{Eq}$ is equivalent to the following unconstrained problem:

$$(\widetilde{P}) \begin{cases} \text{Min } \phi(y) = F(x^{(k)}) + \nabla F(x^{(k)})^T Z_k y + \dfrac{1}{2} y^T H_k y \\ y \in \mathbb{R}^{n-m_k} \end{cases},$$

where $H_k = Z_k^T H Z_k$. As H_k is positive semidefinite, ϕ is a convex function. At this point let us note that $d_{Zk}^{(k)}$ verifies the sufficient first-order optimality condition for $(\widetilde{P})$, and, then: $\nabla\phi(d_{Zk}^{(k)}) = 0$.

So, $d_{Zk}^{(k)}$ is a global solution for $(\widetilde{P})$ and $x^{(k)} + d^{(k)}$ is a solution for $(P)_{Eq}$.

Finally, we will see that, in fact, $x^{(k)} + d^{(k)}$ is a global solution for $(P)_{\mathcal{W}_k}$.

- $x^{(k)} + d^{(k)}$ is a feasible point for $(P)_{\mathcal{W}_k}$
  - If $j \notin \mathcal{W}_k$, using the formula for the steplength (2.29) and $\rho_k = 1$, we get:

    $$\frac{b_j - a_j^T x^{(k)}}{a_j^T d^{(k)}} \geq 1$$

    or the equivalent inequality $a_j^T(x^{(k)} + d^{(k)}) \leq b_j$.
  - If $j \in \mathcal{W}_k$, we have

    $$a_j^T(x^{(k)} + d^{(k)}) = b_j,$$

    using that $d^{(k)}$ is a feasible and $d^{(k)} = Z_k d_{Zk}^{(k)}$ with $Z_k \in N(A_k^T)$.
- Using that feasible set of $(P)_{\mathcal{W}_k}$ is a subset of the feasible set of $(P)_{Eq}$, we deduce that $x^{(k)} + d^{(k)}$ is also a global solution for $(P)_{\mathcal{W}_k}$.

□

Now we present a result about the convergence of the algorithm which will depend on degeneracy, so, first we introduce this concept

Definition 2.5.1. *We will say that the **degeneracy** occurs at a point $x^{(k)}$ if there exists a constraint j such that*

$$a_j^T x^{(k)} = b_j, j \notin \mathcal{W}_k \text{ and } a_j^T d^{(k)} > 0.$$

Theorem 2.5.1. *If the objective function is bounded below in the feasible set and degeneracy does not occur at stationary points (see Definition A.1.7), then the GQP algorithm converges to a local minimum in a finite number of iterations.*

Proof. When we removed a constraint from the working set, $\mathcal{W}_k$ (in Step 2), $x^{(k)}$ is a global minimum for the problem $(\text{P})_{\mathcal{W}_k}$ (see Proposition 2.5.1). In fact, it is enough to realize that $Z_k^T \nabla F(x^k)$=0 and $Z_k H Z_k$ is positive semidefinite. So,

$$F(x^{(k)} + Z_k y) = F(x^{(k)}) + \nabla F(x^{(k)})^T Z_k y + \frac{1}{2} y^T H_k y \geq F(x^{(k)}) \text{ for all } y \in \mathbb{R}^{n-m_k}.$$

Furthermore, as degeneracy does not occur at $x^{(k)}$ we have the strictly decreasing of the objective function: $F(x^{(k+1)}) < F(x^{(k)})$. And we conclude that the working set $\mathcal{W}_k$ does not appear a second time in the optimization process. To obtain the conclusion we take into account:

- There exists a finite number of different working sets.
- At least each n iterations the algorithm has stationary points.

Therefore, after a finite number of iterations the algorithm will find a working set for which the reduced Hessian is positive semidefinite, the reduced gradient is null and also the Lagrange multipliers associated to the inequality constraints of $\mathcal{W}_k$ are strictly positives. Then, by Theorem 2.4.1 the point $x^{(k)}$ is a local minimum. □

2.5.1 Numerical results

In this subsection we will describe how the algorithm solves some non-convex quadratic problems analysing each iteration of the process.

We explain the notation that we will use to refer to the constraints in the following results. Each constraint of the problem is identified with an integer value as Table 2.1 shows

Integrer	**Type of constraint**
$-i \in [-n, -1]$	the lower bound constraint: $lb_i \leq x_{(i)}$
$i \in [1, n]$	the upper bound constraint: $x_{(i)} \leq ub_i$
$i \in [n+1, n+n_I]$	the equality constraint: $a_{i-n}^T x = b_{i-n}$
$i > (n + n_I)$	the inequality constraint: $a_{i-n}^T x \leq b_{i-n}$

Table 2.1: Integer assignment criteria for the problem constraints.

- **Example 1:** This problem, taken from Bunch-Kaufman [5], is interesting because the Hessian matrix is indefinite with at least two local minimums. It has eigth variables, the Hessian matrix

$$h_{ij} = \begin{cases} |i-j| & \text{if i}\neq \text{ j,} \\ 1.69 & \text{if i=j;} \end{cases}$$

  and the linear term of the objective function is given by

$$p = \begin{pmatrix} 7 \\ 6 \\ \vdots \\ 0 \end{pmatrix}.$$

  Moreover, It has seven general inequality constraints:

$$x_i - x_{i+1} \leq 1 + 0.05(i-1) \quad i=1,\ldots,7.$$

and the following bound constraints

$$-i - 0.1(i-1) \leq x_i \leq i \quad i=1,\ldots,8.$$

We start with the vector $x_i^{(0)} = -i$, for $i = 1, \ldots, 8$ and we reach a minimun in seven iterations with the following process:

ITERATION 1:

Initial constraints in the working set: -1,9.
A **negative curvature direction** has been calculated.
Constraint added to the working set: 10.
Number of nonpositive eigenvalues of the reduced Hessian: 2.

ITERATION 2:

Constraints in the working set: -1,9,10.
A **negative curvature direction** has been calculated.
Constraint added to the working set: 11.
Number of nonpositive eigenvalues of the reduced Hessian: 2.

ITERATION 3:

Constraints in the working set: -1,9,10,11.
A **negative curvature direction** has been calculated.
Constraint added to the working set: 12.
Number of nonpositive eigenvalues of the reduced Hessian: 1.

ITERATION 4:

Constraints in the working set: -1,9,10,11,12.
A **negative curvature direction** has been calculated.
Constraint added to the working set: -6.
Number of nonpositive eigenvalues of the reduced Hessian: 1.

ITERATION 5:

Constraints in the working set: -1,9,10,11,12,-6.
A **positive curvature direction** has been calculated.
Constraint added to the working set: 8.
Number of nonpositive eigenvalues of the reduced Hessian: 0.

ITERATION 6:

Constraints in the working set: -1,9,10,11,12,-6,8.
A **positive curvature direction** has been calculated.
Constraint added to the working set: 7.
Number of nonpositive eigenvalues of the reduced Hessian: 0.

ITERATION 7:

Constraints in the working set: -1,9,10,11,12,-6,8,7.
Constraint removed from the working set: -6.
A **positive curvature direction** has been calculated.
Constraint added to the working set: 6.

STOP execution: we have found a local minimum.

The process ends with

$$\overline{x}=(\text{-}1,\ \text{-}2,\ \text{-}3.05,\ \text{-}4.15,\ \text{-}5.3,\ 6,\ 7,\ 8)^T \quad \text{and} \quad F(\overline{x})=\text{-}621.4878250000002.$$

- **Example 2:** We consider an objective function defined with

$$H = \begin{pmatrix} 1 & 2 & 4 & 1 \\ 2 & 13 & 11 & 5 \\ 4 & 11 & 17 & 5 \\ 1 & 5 & 5 & 2 \end{pmatrix} \quad \text{and} \quad p = \begin{pmatrix} -3 \\ -15 \\ -15 \\ -6 \end{pmatrix},$$

with the variables subject to the following equality constraint:

$$x_1 + 2x_2 + 4x_3 + x_4 = 0.$$

and the inequality constraint:

$$x_1 - 7x_2 + x_3 - 2x_4 \leq 0.$$

For this example, where H has two null eigenvalues, we have an infinite number of solutions:

$$\begin{pmatrix} -2 \\ 1 \\ 0 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} -10 \\ -1 \\ 3 \\ 0 \end{pmatrix} + \gamma \begin{pmatrix} -1 \\ -1 \\ 0 \\ 3 \end{pmatrix}.$$

with $\beta, \gamma \in \mathbb{R}$. Starting at the point $x^{(0)}=(0,\ 0,\ 0,\ 0)^T$ we reach a minimun in two iterations with the scheme:

ITERATION 1:

Initial constraints in the working set: 5,6.
Constraint removed to the working set: 6.
A **positive curvature direction** has been calculated.
Number of nonpositive eigenvalues of the reduced Hessian: 2.

ITERATION 2:

Initial constraints in the working set: 5,6.
A **positive curvature direction** has been calculated.
Number of nonpositive eigenvalues of the reduced Hessian: 2.
STOP execution: we have found a local minimum.

The process ends with

$$\overline{x}=(\text{-}0.27273,\ 1.0909,\ \text{-}0.54545,\ 0.27273)^T \quad \text{and} \quad F(\overline{x})=\text{-}4.5.$$

- **Example 3:** We consider the function (taken from [6]), corresponding to

$$H = \begin{pmatrix} -1 & 0 & 0 & 0 & 0 \\ 0 & 0.36 & 0.48 & 0 & 0 \\ 0 & 0.48 & 0.64 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad p = \begin{pmatrix} 2 \\ 1.2 \\ 1.6 \\ 1 \\ -7 \end{pmatrix}.$$

The following inequality constraints:

$$-2 \leq 0.6x_2 + 0.8x_3 \leq 1,$$

$$x_1 - x_4 + x_5 \leq -10,$$

and the boundary constraints for the components

$$0 \leq x_1 \leq 1 \text{ and } -200 \leq x_4 \leq 5.$$

The matrix H has one negative and two null eigenvalues. Starting at the point $x^{(0)}$=(0, -5, 5, 5, -5)^T we reach a minimun in two iterations and the behaviour of the program was:

ITERATION 1:

Initial constraints in the working set: -1,4,6,8.
Constraint removed to the working set: 6.
A **positive curvature direction** has been calculated.
Constraint added to the working set: 7.
Number of nonpositive eigenvalues of the reduced Hessian: 1.

ITERATION 2:

Initial constraints in the working set: -1,4,8,7.
Constraint removed to the working set: 7.
Number of nonpositive eigenvalues of the reduced Hessian: 1.
STOP execution: we have found a local minimum.

The process ends with

$$\overline{x}=(\ 0,\ -6.8,\ 2.6,\ 5,\ -5)^T \text{ and } \quad F(\overline{x})=50.5.$$

- **Example 4:** In this example from [6], we have

$$hij = \begin{cases} -19801 & \text{if i=j=1} \\ -1963 & \text{if i=j>2} \\ -11692 & \text{if j=1 } 2 \leq i \leq 100 \\ -11692 & \text{if i=1 } 2 \leq j \leq 100 \\ -2044 & \text{in other case} \end{cases} \quad \text{and } p = \begin{pmatrix} -1 \\ \vdots \\ -1 \end{pmatrix}.$$

And the following inequality constraint:

$$-10 \leq x_1 + \ldots + x_{100} \leq 10.$$

The matrix H has one negative eigenvalue and the rest are strictly positive. Starting at the point $x^{(0)} = 0$ the algorithm find a solution in only two iterations:

ITERATION 1:

Initial constraints in the working set:0.

A **negative curvature direction** has been calculated.

Constraint added to the working set: 101.

Number of nonpositive eigenvalues of the reduced Hessian: 1.

ITERATION 2:

Initial constraints in the working set: 101.

A **positive curvature direction** has been calculated.

We have found a local minimum.

Number of nonpositive eigenvalues of the reduced Hessian: 0.

STOP execution: we have found a local minimum.

The problem has a local minimun where the function values is $f(\overline{x})$= -3125243.28905.

- **Example 5:** Now we take

$$H = -Id_{n\times n} \text{ and } p = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$$

subject to the following boundary constraints: $-1 \leq x_i \leq 1$ for $i = 1, \ldots, n$.

This problem has the peculiarity that has a minimum at any x such that $|x_i|$=1 for $i = 1, \ldots, n$.

For n=100 and starting at $x^{(0)} = (0, \ldots, 0)^T$ we have found a local minimum after one hundred and one iterations, where the objective function value is $F(\overline{x})$=-100.

Now, we analize a collection of examples from QPLIB [1], a library of quadratic programming instances. We choose those whose variables have real values (they are not restricted to integer values) and with linear constraints.

In Table 2.2 we present the problem in the first five columns (code number of the problem, number of variables, number of nonpositive eigenvalues of the Hessian, number of equality constraints and number of inequality constraints), then we show the initial vector, the number of iterations for solving the problem, the number of nonpositive curvature directions that have been computed, the objective function value at the computed solution and if this solution is a global or a local minimum (see Definition A.1.1 and Definition A.1.2). Some of the numerical solutions are local minimums, and to check it we have been used Theorem 2.4.1.

In Table 2.3 we show a comparative between the results obtained by our code and other softwares. We will consider the quadprog function from MATLAB [2] and the qp function from OCTAVE [3]. For each example (fisrt column), we present the number of iterations and the final function value (in the fourth and fifth column, respectively).

Software: Quaprog									
Problem	n	nNpeig	n_I	n_D	$x^{(0)}$	Niter	nNd	fval	Minimum
0018	50	24	1	0	e_{14}	3	0	-6.3860149815	Global
					e_{17}	3	0	-6.3860149815	Global
					e_{20}	3	0	-5.7679399370	Local
					e_{39}	3	0	-6.3860149815	Global
					e_{45}	4	0	-5.3386872077	Local
0343	50	24	1	0	e_1	5	0	-5.8237238705	Local
					e_{14}	4	0	-6.3860149816	Global
					e_{45}	5	0	-5.3386872077	Local
2712	200	100	1	0	e_{21}	11	3	0.0128688328	Global
					e_{100}	3	0	0.0555190572	Local
					e_{163}	8	1	0.1488494049	Local
2761	500	250	1	0	e_{48}	2	0	0.0010485291	Global
					e_{250}	5	0	0.0213262913	Local
					e_{442}	6	1	0.0213262914	Local

Table 2.2: Solving with our code some problems of the library QPLIB.

Example	Code	Iterations	Fval
1	Quaprog	7	-621.487825
	Quadprog(MATLAB)	*	($*^1$)
	qp(OCTAVE)	10000	-69.995 ($*^3$)
2	Quaprog	2	-4.500000
	Quadprog(MATLAB)	3	$-4.238135 \cdot 10^{-9}$
	qp(OCTAVE)	1000	$3.7278 \cdot 10^{-14}$($*^3$)
3	Quaprog	2	50.5
	Quadprog(MATLAB)	1	($*^2$)
	qp(OCTAVE)	*	($*^3$)
4	Quaprog	2	-3125243.28905
	Quadprog(MATLAB)	*	$*^1$
	qp(OCTAVE)	4	-3125243.28905
5	Quaprog	101	-100
	Quadprog(MATLAB)	*	$*^1$
	qp(OCTAVE)	201	-100

Table 2.3: Comparing different QP codes.

The initial vector $x^{(0)}$ is the same as the one used in the Section 2.5.1.

The MATLAB code quadprog is not be able to solve nonconvex quadratic problems. The algorithms that it uses are of interior-point or trust-region type. In Table 2.3:

- ($*^1$) means that the program stops because it detects that the problem is nonconvex.
- ($*^2$) indicates that it stops because it converges to an infeasible point.

- For the example 2, MATLAB sais that it solves the problem but it really does not, the computed vector is neither a global nor a local minimum.

On the other hand, OCTAVE solves Examples 4 and Example 5 although it does twice as many iterations than our code. Moreover, in the output OCTAVE identifies these problems as convex. Maybe this could be because the reduced Hessians used for solving the problem were positive definite (but the Hessians of these problems are not definite positive matrix). Furthermore, OCTAVE does not solve the rest of examples. The mark ($*^3$) means that it can not progress from the objective function value showed and gives a message saying that it exceeds the maximum iteration number, even for 10^6 iterations. In particular, for Example 3, qp is also unable to return any value because of computing difficulties.

Table 2.4 shows the numerical results obtaining for OCTAVE for the same problems of Table 2.2.

Software: qp (OCTAVE)							
Problem	n	nNpeig	n_I	n_D	$x^{(0)}$	Niter	fval
0018	50	24	1	0	e_{14}	7	-3.14317632338
					e_{17}	7	-3.14317632338
					e_{20}	10	-2.87665108035
					e_{39}	7	-3.14317632338
					e_{45}	9	-2.59490655613
0343	50	24	1	0	e_1	16	-2.77901982431
					e_{14}	8	-3.14317632338
					e_{45}	10	-2.59490655613
2712	200	100	1	0	e_{21}	20	0.01255528191 (**)
					e_{100}	8	0.04674979429
					e_{163}	32	0.03337821682
2761	500	250	1	0	e_{48}	6	0.00104791687 (**)
					e_{250}	19	0.04270710157
					e_{442}	28	0.02884602850

Table 2.4: Solving with OCTAVE problems of the library QPLIB.

In these examples OCTAVE always stops identifying the problems as convex and returning what it claims to be a global solution, although these points are not. Moreover, it can be appreciated that for each example and each initial vector, OCTAVE does more iterations than our code.

Let us remark that the objective function values with the mark (**), are better than ours, but if we evaluate the objective function at the final point given by the code, the values obtained are:

Problem	$x^{(0)}$	fval
2712	e_{21}	0.0131535824
2761	e_{48}	0.0010489173

Table 2.5: Objective function values for OCTAVE.

As a comment, quadprog from MATLAB has failed in all these four examples because it identifies these problems as nonconvex, independently of the initial point.

Now we will consider five numerical experiments related to the convex QP case. For each problem, we show the Hessian matrix, the vector of the linear term and the constraints:

Example 6 : $H = \begin{pmatrix} 1 & & \\ & \ddots & \\ & & n \end{pmatrix}$, $p(i) = 0$, $i = 1, \ldots, n$ and the constraints

$-x_1 - x_2 - \ldots - x_n \leq -10$ and $0 \leq x_i$, $i = 1, \ldots, n$, where n=100.

Example 7 : $H = \begin{pmatrix} 1 & & \\ & \ddots & \\ & & n \end{pmatrix}$, $p(i) = (-1)^i\sqrt{i}$, $i = 1, \ldots, n$ and the constraints

$-x_1 - x_2 - \ldots - x_n \leq -10$ and $0 \leq x_i$, $i = 1, \ldots, n$, where n=100.

Example 8 : $H = \begin{pmatrix} 6 & 2 & 1 \\ 2 & 5 & 2 \\ 1 & 2 & 4 \end{pmatrix}$, $p(i) = (-8, -3, -3)^T$ and the constraints

$x_1 + x_3 = 3$ and $x_2 + x_3 = 0$.

Example 9 : $H = \begin{pmatrix} 4 & 0 & -4 \\ 0 & 4 & 2 \\ -4 & 2 & 6 \end{pmatrix}$, $p(i) = (-2, 2, 1)^T$ and the constraints $x_i \geq 0$, i=1,2,3.

Example 10 : $H = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$, $p(i) = (-3, 0, 1)^T$ and the constraints

$x_1 + x_2 \leq 2$ and $0 \leq x_i \leq 1$, i=1,2,3.

So, from Table 2.6 we can see that our code get the same solution and requires a similar number of iterations that the softwares of MATLAB and OCTAVE. Only in the first case MATLAB requires fewer iterations than our code and OCTAVE. The initial point indicated is used by our code and OCTAVE, but not for MATLAB.

Convex problems				
Example	Software	$x^{(0)}$	fval	iter
6	Quaprog	$(1,\ldots,1)^T$	9.638781798697996	4
	Quadprog(MATLAB)		9.638781798703729	4
	qp (OCTAVE)		9.638781798698002	3
7	Quaprog	$(1,\ldots,1)^T$	-24.96886835221521	53
	Quadprog (MATLAB)		-24.96886834566575	5
	qp (OCTAVE)		-24.96886835221522	53
8	Quaprog	$(3,0,0)^T$	-3.5	2
	Quadprog (MATLAB)		-3.5	1
	qp (OCTAVE)		-3.5	2
9	Quaprog	$(0,0,0)^T$	-0.75	3
	Quadprog (MATLAB)		-0.75	5
	qp (OCTAVE)		-0.75	5
10	Quaprog	$(0,0,0)^T$	-2.25	3
	Quadprog (MATLAB)		-2.25	4
	qp (OCTAVE)		-2.25	5

Table 2.6: Solving quadratic convex problems.

2.6 A few practical details

This section is related to our code. We show some details of its implementation.

2.6.1 About QR factorization

In this subsection we will explain how we update the QR factors in the code depending on whether a constraint has been removed or added to the working set.

Let $\mathcal{W}_k$ be the working set whose associated matrix is A_k. The QR factorization of A_k is given by (2.27), where the factors Q and R has the same estructure that in Section 2.4.1.

When we add a new constraint, $a_j^T x \leq b_j$, to the working set, $\mathcal{W}_{k+1} = \mathcal{W}_k \cup \{j\}$, the associated matrix will have a new extra column which will be placed at the end of the matrix:

$$A_{k+1} = (A_k \;\; a_j).$$

So, we will have $A_{k+1} = Q_k \hat{R}_{k+1}$, with $\hat{R}_{k+1} = (R_k \;\; Q_k^T a_j)$, having the following structure:

$$\hat{R}_{k+1} = \begin{pmatrix} * & * & * & \dots & * \\ 0 & * & * & \dots & * \\ 0 & 0 & * & \dots & * \\ \vdots & \ddots & \ddots & \ddots & \ddots \\ \vdots & \ddots & \ddots & 0 & * \\ 0 & 0 & 0 & 0 & \times \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \times \end{pmatrix}$$

We obtain the QR factors by using only one Householder matrix, H_o, to transform the elements indicated by $\times$ into 0. So, $R_{k+1} = H_o \hat{R}_k$ and $Q_{k+1} = Q_k H_o$.

In the case that we remove the j-th constraint from the working set, the matrix A_{k+1} will have one less column than A_k. So, $A_{k+1} = Q_k \hat{R}_{k+1}$, where $\hat{R}_{k+1}$ has the following structure:

$$\hat{R}_{k+1} = \begin{pmatrix} * & * & \dots & * & \dots & * \\ 0 & * & \dots & * & \ddots & * \\ 0 & 0 & \ddots & \ddots & \ddots & * \\ \vdots & \ddots & \ddots & * & \ddots & \vdots \\ \vdots & \ddots & \ddots & \times & * & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & * \\ \vdots & \vdots & \vdots & \vdots & \vdots & \times \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Now, we update the QR factors by using $m_k - j$ Givens matrices to transform into 0 each element indicated by $\times$, one per column. We denote as $G^{j,j+1}$ the Givens matrix which cancels the element (j+1,j) and has the following structure:

$$G^{j,j+1} = \left(\begin{array}{c|cc|c} (Id)_{j-1} & & & \\ \hline & c_j & s_j & \\ & s_j & -c_j & \\ \hline & & & (Id)_{n-(j+1)} \end{array}\right).$$

Thus, the matrix $\hat{R}_{k+1} = G^{m_k-1,m_k} G^{m_k-2,m_k-1} \dots G^{j,j+1} \hat{R}_k$ is an upper triangular matrix and now $A_{k+1} = Q_{k+1} \hat{R}_{k+1}$, where

$$Q_{k+1} = Q_k G^{j,j+1} \dots G^{m_k-2,m_k-1} G^{m_k-1,m_k}.$$

As a final comment, note that the columns of the matrix Z_k remain unchanged, and Z_{k+1} is formed by the columns of Z_k and a new column, which is placed in the first position.

We have chosen the QR factorization to solve linear systems because it is more stable than other factorizations as LU. The use of QR factorization is not suitable for large scale problems.

2.6.2 Solving with Cholesky factorization and pivotation

If $H_k \in \mathbb{R}^{n_k \times n_k}$ is a reduced Hessian and its Cholesky factorization is given by $PH_kP^T{=}LL^T$, being P a pivotation matrix. If g is an n_k-dimensional vector, then the solution of the system $H_kd = g$, can be obtained by the following steps:

- $\bar{u}$ is calculated solving the system $Lu = Pg$.
- $\bar{v}$ is calculated solving the system $L^tv = \bar{u}$
- The solution is $\bar{d}{=}P^T\bar{v}$

Observation 2.6.1. *Let us notice that in computational practice we can avoid the storage of the pivotation matrix P, and also the multiplications, by using a vector with as many columns as P.*

Observation 2.6.2. *In relation to computational efficiency let us note that we only need to solve triangular linear systems for getting the directions.*

2.6.3 Cholpar code

We end up this section presenting the factorization code to illustrate the complexity of the algorithm:

```
function [LT,ipvt,ind]=cholpar(H,n,Z,NColZ,mode)
%%%%%%%%%%%
%Author:Carlos Crespo under the direction of Cecilia Pola
%Date:27/11/2020
%Reference: anfm03 FORTRAN function of Eduardo Casas and Cecilia Pola.
%OBJECTIVE: This program calculates a total or partial
%Cholesky
%                    P*M*(Z'*H*Z)*M'*P'
% where P is a permutation matrix and M=flip(eye(NColZ,1)).
%
% Input list:
% - H: nxn-dimensional matrix
% - n: number of rows (and columns) of H
% - Z: nxNColZ-dimensional matrix.
% - NColZ: number of columns of Z.
% - mode is an indicator that takes the values:
%    *0: if the matrix is indefinite the factorization is
%        not carried out.
%    *other number: the factorization is calculated (partial
%                   or complete).
%
% Output list :
% - LT: from row ind+1 it contains the upper triangular matrix
%       of the Cholesky factorization of the factorized box (the
%        factorization will be partial in the indefinite case)
% - ipvt: indicate the exchange of the columns and rows associated
%         with matrix P
% - ind: variable that takes the next values:
%   * n: if the factorizatiaon has not been done and modo=0
%   *(-1,ND+1): if the factorization is finished. ND-ind is
%                the size of the non-zero part of LT.
%   *(-ND-1,0):the factorization is not finished. A negative
%               element in the position -IND in the diagonal of
%               the matrix LT has been found.
%   *(-2*ND-1,-ND): the factorization is not complete because
%                   the matrix is indefinite: LT has in the position
%                   -IND-ND in the diagonal a null element and a non-zero
%                   element in the position (-ND-IND,-ND-IND+1)
%%%%%%%%%%%
eps1=eps^0.75;ind=0;LT=zeros(n,n);Z=Z(:,NColZ:-1:1);
if NColZ==0
    LT=[];ipvt=[];ind=0;
    return
end
if ind==0
    ndim=NColZ;iifact=0;ipvt=(1:ndim);
end
smax=1;
for i=1:ndim
    if ind==0
        ii=i;
    end
    s=Z(1:n,i)'*H(1:n,1:n)*Z(1:n,i);
    if mode==0 && s<-eps1
        ind=n;
        return
    end
    LT(ii,i)=s;smax=max(abs(s),smax);
end
if ndim==1
    ik=ind+1;s=LT(ik,1);
    if s>eps1
        LT(ik,1)=sqrt(s);ind=0;
    elseif s<-eps1
```

```
        ind=-1;
    else
        ind=1;
    end
    return
end
eps0=eps*smax;eps1=eps0*ind;s1=0;beta=max(eps0*ndim*10,sqrt(smax)*1.2);
for k=1:ndim-1
    eps1=eps1+eps0;kk=k+1;ik=k;ik0=ik-1;
    if iifact==0
        LT(k,k)=LT(k,k)-LT(1:ik0,k)'*LT(1:ik0,k);
        sk=LT(k,k);
        if s1>beta || sk<=eps1
            iifact=1;
            for i=kk:ndim
                LT(i,i)=LT(i,i)-LT(1:ik0,i)'*LT(1:ik0,i);
            end
        end
    else
        sk=LT(ik,k);
    end
    if s1>beta
        s=-1;
    elseif iifact==1
        j=k;s=sk;
        for i=kk:ndim
            ii=i+ind;Rii=LT(ii,i);
            if Rii>s
                j=i;s=Rii;
            end
        end
    else
        s=sk;j=k;
    end
    if s>eps1
        if iifact==1
           index=LT(1:ik0,k);LT(1:ik0,k)=LT(1:ik0,j);LT(1:ik0,j)=index;
           auxipvt=ipvt(k);ipvt(k)=ipvt(j);ipvt(j)=auxipvt;LT(ind+j,j)=sk;l=ipvt(k);
        else
            l=k;
        end
        sk=sqrt(s);LT(ik,k)=sk;w=H*Z(1:n,l);s1=0;
        for i=kk:ndim
            j=ipvt(i);s=Z(1:n,j)'*w;
            if ik0>0
                s=s-LT(1:ik0,i)'*LT(1:ik0,k);
            end
            Rik=s/sk;s1=max(s1,abs(Rik));LT(ik,i)=Rik;ii=i;
            if iifact==1
                LT(ii,i)=LT(ii,i)-Rik*Rik;
            end
        end
    else
        s=sk;j=k;
        for i=kk:ndim
            ii=ind+i;Rii=LT(ii,i);
            if Rii<s
                j=i;s=Rii;
            end
        end
        if s<-eps1
            if mode==0
                ind=n;
                return;
            end
            index=LT(1:ik0,k);LT(1:ik0,k)=LT(1:ik0,j);LT(1:ik0,j)=index;
            auxipvt=ipvt(j);ipvt(j)=ipvt(k);ipvt(k)=auxipvt;LT(ind+j,j)=sk;
            LT(ik,k)=s;ind=-k;
            return;
```

```
        else
            for j=k:ndim-1
                nj=ipvt(j);ij=j+ind;w=H*Z(1:n,nj);w(n)=s1;
                for i=j+1:ndim
                    j=ipvt(i);s1=Z(:,ipvt(i))'*w;
                    if ik0>0
                        s1=s1-LT(1:ik0,i)'*LT(1:ik0,j);
                    end
                    LT(ij,i)=s1;s1=abs(s1);
                    if s1>s
                        s=s1;l=i;
                    end
                end
                if s>eps1
                    if mode==0
                        ind=n;
                        return
                    end
                    index=LT(1:ik0,k);index2=LT(1:ik0,kk);
                    LT(1:ik0,k)=LT(1:ik0,j);LT(1:ik0,j)=index;
                    LT(1:ik0,kk)=LT(1:ik0,l);LT(1:ik0,l)=index2;
                    LT(ik,kk)=LT(ij,l);auxipvt=ipvt(j);ipvt(j)=ipvt(k);
                    ipvt(k)=auxipvt;auxipvt=ipvt(l);ipvt(l)=ipvt(kk);
                    ipvt(kk)=auxipvt;ind=-ndim-k;
                    return
                end
            end
            ind=ndim-k+1;
            return
        end
    end
end
eps1=eps0+eps1;in=ndim+ind;
if iifact==0
    LT(in,in)=LT(in,in)-LT(1:in-1,in)'*LT(1:in-1,in);
end
s=LT(in,ndim);
if s>eps1
    LT(in,ndim)=sqrt(s);ind=0;
elseif s<-eps1
    ind=-ndim;
else
    ind=1;
end
end
```

Chapter 3

Generalized General Quadratic Programming (GGQP)

Now we add to the quadratic programming problem (GQP) some extra nondifferentiable terms in the objective function, that are regulated by a parameter $\alpha > 0$,

$$(GGQP)\begin{cases} \text{Minimize } \widetilde{F}(x) = \dfrac{1}{2}x^T Hx + p^T x \ + \alpha \left(\displaystyle\sum_{j=1}^{n_{IF}} |\bar{a}_j^T x - \bar{b}_j| \ + \sum_{j=n_{IF}+1}^{n_{IF}+n_{DF}} (\bar{a}_j^T x - \bar{b}_j)_+ \right) \\ x \in \mathbb{R}^n \\ \text{subject to} \\ \qquad a_j^T x = b_j \ , \quad j = 1, \ldots, n_I; \\ \qquad a_j^T x \leq b_j \ , \quad j = n_I + 1, \ldots, n_I + n_D; \\ \qquad l_j \leq x_j \leq u_j \ , \quad j = 1, \ldots, n \ . \end{cases}$$

where $(\bar{a}_j^T x - \bar{b}_j)_+ = \max\{0, \bar{a}_j^T x - \bar{b}_j\}$ and α is called **penalty parameter**.

Before proceding further, we will establish a notation that might be useful in the following.

$$\xi(x) = \sum_{j=1}^{n_{IF}+n_{DF}} \xi_j(x) \quad \text{where} \quad \xi_j(x) = \begin{cases} |\bar{a}_j^T x - \bar{b}_j| & \text{for j=1,}\ldots, n_{IF} \\ (\bar{a}_j^T x - \bar{b}_j)_+ & \text{for } j = n_{IF}+1, \ldots, n_{IF} + n_{DF} \end{cases}$$

In relation to these terms it is interesting to note the following equivalence:

$$\xi(\overline{x}) = 0 \Longleftrightarrow \overline{x} \text{ verifies } \begin{cases} \bar{a}_j^T \overline{x} = \bar{b}_j & \text{for } j=1, \ldots, n_{IF} \\ \bar{a}_j^T \overline{x} \leq \bar{b}_j & \text{for } j = n_{IF}+1, \ldots, n_{IF} + n_{DF} \end{cases}$$

So, we can consider that l_1 penalty terms are associated with some relaxed constraints of a (GQP) problem. The level of verification of such constraints could be controled by

varying the value of the penalty parameter. Note that we distinguish with a bar the coefficients of such constraints. Moreover, problem (GGQP) leads to the same solutions that (GQP) provided when the penalty parameter is large enough and there exists feasible points for all the constraints.

This kind of problems appears frequently solving non-linear programming problems (NLP) by a sequential quadratic programming method (SQP) (at each iteration they calculate the descent direction by solving a quadratic problem). The non-linear problem may have linear constraints and non-linear constraints:

$$(LC)\begin{cases} a_j^T x = b_j; & \text{if } j = 1,\ldots,n_I \\ a_j^T x \leq b_j & \text{if } j{=}1,\ldots,n_D \end{cases};(NLC)\begin{cases} h(x) = 0; & \text{if } j = 1,\ldots,n_{IF} \\ g(x) \leq 0 & \text{if } j = 1+n_{IF},\ldots,n_{IF}+n_{DF} \end{cases}$$

If $x^{(k)}$ is the current approximation, $x^{(k)}$ is feasible for the linear constraints and (SQP) methods use linear approximations of the nonlinear constraints:

$$(NLCd)\begin{cases} \nabla h(x^k)^T d = -h(x^k) & \text{if } j = 1,\ldots,n_{IF} \\ \nabla g(x^k)^T d \leq -g(x^k) & \text{if } j = 1+n_{IF},\ldots,n_{IF}+n_{DF} \end{cases}.$$

For avoiding non feasible quadratic programs, the (NLCd) constraints can be introduced in the penalty term of the quadratic problem.

The (LC) constraints will be transformed into the following ones:

$$(LCd)\begin{cases} a_j^T d = 0 & \text{if } j{=}1,\ldots,n_I \\ a_j^T (d + x^k) \leq b_j & \text{if } j{=}1,\ldots,n_D \end{cases}.$$

In practical optimization, formulations with penalty terms l_1 are used in many applications as for image restoring (see, for instance, [11]).

3.1 Some results of subdifferential calculus

To raise an algorithm that solves the (GGQP) problem, first we need to determinate the optimality conditions associated to (GGQP).

The optimality conditions criteria , in Section 2.4 and in Appendix A.2 (Theorem A.2.1 and A.2.2), involve the derivates of the functions. However, now our objective function contains non-differentiable terms, so we have to use some results of subdifferental calculus for convex functions (see [4]).

Definition 3.1.1. *A function $f : K \longrightarrow (-\infty, +\infty]$, not identically $+\infty$, defined on a* ***convex set*** *$K \subset \mathbb{R}^n$, is said to be convex if*

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y), \quad \forall t \in (0,1), \quad \textit{for all } x, y \in K. \tag{3.1}$$

If the inequality is strict the function is said to be strictly convex.

Definition 3.1.2. *The **essential domain** for a funcion $f : \mathbb{R}^n \longrightarrow (-\infty, +\infty]$ is:*

$$dom(f) = \{x \in \mathbb{R}^n : f(x) < +\infty\} \tag{3.2}$$

Note that every convex function defined on a convex set $f : K \longrightarrow (-\infty, +\infty]$ can be expanded to $\mathbb{R}^n$ considering the function $\overline{f} : \mathbb{R}^n \longrightarrow (-\infty, +\infty]$ given by:

$$\overline{f}(x) = f(x) \quad \text{if } x \in K,$$

We can already present the main concept for our interests.

Definition 3.1.3. *A **subgradient** of a function $f : \mathbb{R}^n \longrightarrow (-\infty, +\infty]$ at the point $x \in \mathbb{R}^n$ is a vector $x^* \in \mathbb{R}^n$ such that*

$$(x^*)^T(y - x) + f(x) \leq f(y), \quad \forall y \in \mathbb{R}^n.$$

*The set of all subgradients of f at x is called **subdifferential** of f at x and it is denoted by $\partial f(x)$.*

The subdifferential concept is a generalisation of the classical gradient, as we see in the following result

Proposition 3.1.1. *Let $f : \mathbb{R}^n \longrightarrow (-\infty, +\infty]$ be a convex function that is differentiable at the point $x \in dom(f)$. Then,*

$$\partial f(x) = \{\nabla f(x)\}.$$

Now we will apply these concepts to calculate subdifferentials related to our objective function.

- The function $f : \mathbb{R} \longrightarrow \mathbb{R}$, defined by $f(x) = \max\{0, x\}$

  1. If $x \in (0, \infty)$, then $f(x) = x$, and using the above proposition,

  $$\partial f(x) = \{f'(x)\} = \{1\}.$$

  2. If $x \in (-\infty, 0)$, then $f(x) = 0$, and using the same arguments as before,

  $$\partial f(x) = \{f'(x)\} = \{0\}.$$

  3. If $x = 0$, f is not differentiable. At this point, using Definition 3.1.3, we obtain the following equivalences:

  $$x^* \in \partial f(0) \Longleftrightarrow (x^*)^T y \leq \max\{0, y\},\ \forall y \in \mathbb{R} \Longleftrightarrow \begin{cases} (x^*)^T y \leq y & \text{if } y > 0 \\ (x^*)^T y \leq 0 & \text{if } y \leq 0 \end{cases}.$$

  So, in this case, $\partial f(0) = [0, 1]$.

- The function $f : \mathbb{R} \longrightarrow \mathbb{R}$ defined by $f(x) = |x|$.

  1. If $x \in (0, \infty)$, then $f(x) = x$, and as it is diferentiable and convex,
  $$\partial f(x) = \{f'(x)\} = \{1\}$$
  2. If $x \in (-\infty, 0)$, then $f(x) = -x$, and as in the previos case,
  $$\partial f(x) = \{f'(x)\} = \{-1\}$$
  3. If $x = 0$, at this point f is not differentiable. The following equivalences state $\partial f(0) = [-1, 1]$:
  $$x^* \in \partial f(0) \Longleftrightarrow (x^*)^T y \leq |y|,\ \forall y \in \mathbb{R} \Longleftrightarrow \begin{cases} (x^*)^T y \leq y & \text{if } y > 0 \\ (x^*)^T y \leq -y & \text{if } y \leq 0 \end{cases}.$$

Summarizing, for the function $f(x) = \max\{0, x\}$ the subdifferential $\partial f(x) \subset [0, 1]$ and for $f(x) = |x|$, $\partial f(x) \subset [-1, 1]$. Now, using these subgradients, we will get the optimality conditions for the generalized quadratic programming.

3.2 Optimality conditions with subdifferential calculus

Now we state the optimality conditions for problem (GGQP) (see [13], for instance). In the following result we consider first order optimality conditions.

Theorem 3.2.1. *Given the problem (GGQP), be $\overline{x} \in \mathbb{R}^n$ a feasible point.*

If $\overline{x}$ is a local solution of (GGQP) problem, then there are two Lagrange multiplier vectors, $\overline{\mu} \in \mathbb{R}^{n_{IF}+n_{DF}}$ and $\overline{\lambda} \in \mathbb{R}^{n_I+n_D}$, such that:

$$H\overline{x} + p + \sum_{j=1}^{n_{IF}+n_{DF}} \overline{\mu}_j \overline{a_j} + \sum_{j=1}^{n_I+n_D} \overline{\lambda}_j a_j = 0; \tag{3.3}$$

$$\overline{\mu}_j \in \begin{cases} \{1\} & \text{if } \overline{a}_j^T \overline{x} - \overline{b}_j > 0, \\ [-1, 1] & \text{if } \overline{a}_j^T \overline{x} - \overline{b}_j = 0, \quad j = 1 \ldots, n_{IF}; \\ \{-1\} & \text{if } \overline{a}_j^T \overline{x} - \overline{b}_j < 0, \end{cases} \tag{3.4}$$

$$\overline{\mu}_j \in \begin{cases} \{1\} & \text{if } \overline{a}_j^T \overline{x} - \overline{b}_j > 0, \\ [0, 1] & \text{if } \overline{a}_j^T \overline{x} - \overline{b}_j = 0, \quad j = n_{IF} + 1, \ldots, n_{IF} + n_{DF}; \\ \{0\} & \text{if } \overline{a}_j^T \overline{x} - \overline{b}_j < 0, \end{cases} \tag{3.5}$$

$$\overline{\lambda}_j (a_j^T \overline{x} - b_j) = 0, \quad j = n_I + 1, \ldots, n_I + n_D; \tag{3.6}$$

$$\overline{\lambda_j} \geq 0, \quad j = n_I + 1, \ldots, n_I + n_D. \tag{3.7}$$

If H is positive semidefinite the reciprocal is also true.

Now we present a sufficient second order optimality condition for the nonconvex case:

Theorem 3.2.2. *Let $\overline{x} \in \mathbb{R}^n$ be a feasible point for the (GGQP) problem, $\overline{\lambda} \in \mathbb{R}^{n_I+n_D}$ and $\overline{\mu} \in \mathbb{R}^{n_{IF}+n_{DF}}$ satisfying the first-order optimality conditions* (3.3), (3.4), (3.5), (3.6) *and* (3.7)*; the index sets*

$$I = \{j : 1 \leq j \leq n_I\} \cup \{j : j > n_I \text{ and } \overline{\lambda}_j > 0\}$$

$$\text{and } \overline{I} = \{j : 1 \leq j \leq n_{IF} \text{ and } -1 < \overline{\mu}_j < 1\} \cup \{j : j > n_{IF} \text{ and } 0 < \overline{\mu}_j < 1\},$$

m=card(I)+card($\overline{I}$), $\overline{A}_I = (a_j)_{j\in I}$, $\overline{A}_{\overline{I}} = (a_j)_{j\in\overline{I}}$ and $\overline{Z} \in \mathbb{R}^{n\times(n-m)}$ with full rank and such that $\overline{A}_I^T\overline{Z} = \overline{A}_{\overline{I}}^T\overline{Z} = 0$.

Then, if m=n or $\overline{Z}^T H\overline{Z}$ is positive semidefinite, then $\overline{x}$ is a local minimum.

Observation 3.2.1. *In the Theorem 3.2.2 if m=n or $\overline{Z}^T H\overline{Z}$ is semi-positive definite, then $\overline{x}$ is a strictly local minimum.*

3.3 An algorithm for GGQP

This algorithm follows the same structure as the algorithm for the problem (GQP), so, instead of repeating the whole algorithm we only will explain the differences. These changes affect mainly in the calculation of the gradient and the descent direction, adapting also the stop conditions to the new optimality conditions theorem.

Now the working set $\mathcal{W}_k$ is formed by both constraints belonging to the feasible set of (GGQP) as well as those that are in the objective function, whose associated matrix are A_k and $\overline{A}_k$. We denote as J_k the index set of the constraints in the objective function such that they are not in the working set and as I_k the set formed by the active constraints which belong to the feasible set of (GGQP).

Based on the subdifferentials calculated in the final part of Section 3.1, we replace $\nabla f(x)$ by

$$g^{(k)} = Hx^{(k)} + p + \alpha \sum_{j=1}^{n_{IF}+n_{DF}} \gamma_j^{(k)} \overline{a}_j$$

where

$$\gamma_j^{(k)} = \begin{cases} -1 & \text{if } \overline{a}_j^T x^{(k)} < \overline{b}_j \text{ and } j = 1, \ldots, n_{IF}, \\ +1 & \text{if } \overline{a}_j^T x^{(k)} > \overline{b}_j, \\ 0 & \text{in other case.} \end{cases}$$

Now in the Step 2, to know if we get a solution for the problem (GGQP), we have to check if the computed Lagrange multipliers associated to the current working set meet

the conditions of Theorem 3.2.1 and, furthermore, they do not take extreme values. If not, we removed the multiplier that most violates the corresppponding condition:

$$s = \min_{j \in \mathcal{W}_k, j > n_I} \{s_j^{(k)}\}$$

where

$$s^{(k)} = \begin{cases} \lambda_j^{(k)}, \\ 1 - |\mu_j^{(k)}|, \\ \min\{\mu_j^{(k)}, 1 - \mu_j^{(k)}\}, \end{cases}$$

for constraints belonging to the feasible set, associated with relaxed equality constraints and associated with relaxed inequality constraints, respectively.

As we did in Section 2.5 we update the QR factors and we computed the decomposition of the new reduced Hessian. But now, if we removed from the working set a constraint l associated to the objective function we have to update the gradient as follows:

$$\hat{g}^{(k)} = g^{(k)} + \sigma^{(k)} \overline{a}_l \tag{3.8}$$

being

$$\sigma^{(k)} = \begin{cases} 1 & \text{if } \mu_l^{(k)} > 1 \\ 0 & \text{if } \mu_l^{(k)} < 0 \text{ and } l > n_{IF} \\ -1 & \text{if } \mu_l^{(k)} < -1 \text{ and } l \leq n_{IF} \end{cases}.$$

Now, we calculate the descent direction $d^{(k)}$ with the new gradient. We have to check that $d^{(k)}$ verifies that the sign of $\overline{a}_l^T d^{(k)}$ is consistent with the election of the value for σ_k. As we have removed a constraint from the working set, we have the following equality:

$$g^{(k)} + \overline{A}_k \mu_j^{(k)} \overline{a}_j + A_k \lambda_j^{(k)} a_j = 0,$$

from which

$$-\left(\overline{A}_k \mu_j^{(k)} + A_k \lambda_j^{(k)}\right)^T d^{(k)} = (g^{(k)})^T d^{(k)}$$

and as we have that $\overline{a}_j^T d^{(k)} = 0$ and $a_j^T d^{(k)} = 0$ for all $j \in \mathcal{W}_k \setminus \{l\}$, then

$$-\mu_k^{(k)} \overline{c}_l^T d^{(k)} = (g^{(k)})^T d^{(k)},$$

and adding to the both sides $\sigma a_l^T d^{(k)}$ and using (3.8)

$$\overline{a}_l^T d^{(k)} = \frac{\hat{g}^T d^{(k)}}{(-\mu_j^{(k)} + \sigma^{(k)})}.$$

Now, as we know that $d^{(k)}$ is a descent direction, we have $\hat{g}^T d^{(k)} < 0$, and depending on the value of $\sigma^{(k)}$, we have

$$\overline{a}_l^T d^{(k)} \begin{cases} > 0 & \text{if } \sigma^{(k)} = 1, \\ < 0 & \text{if } \sigma^{(k)} = -1 \text{ or } 0, \end{cases}$$

and $d^{(k)}$ is consistent with the $\sigma^{(k)}$.

We now focus in the calculation of the steplength associated to the descent direction. We will consider three cases: positive, null or negative curvature direction. The general scheme for all the cases is the following:

For the active constraints of the feasible set of (GGQP), the steplength expression is

$$\rho_{strict} = \min_{j \notin I_k, a_j^T d^{(k)} > 0} \frac{b_j - a_j^T x^{(k)}}{a_j^T d^{(k)}}. \tag{3.9}$$

Now we define the sets:

$$V_{k,1} = \{j \in J_k : \overline{a}_j^T x^{(k)} - \overline{b}_j > 0, \overline{a}_j^T d^{(k)} < 0\};$$

$$V_{k,2} = \{j \in J_k : \overline{a}_j^T x^{(k)} - \overline{b}_j < 0, \overline{a}_j^T d^{(k)} > 0\}.$$

For each index of these sets we have a steplength where the associated constraint becomes active. We consider the minimum of these steplenghts:

$$\rho_{obj} = \min_{j \in (V_{k,1} \cup V_{k,2})} \frac{\overline{b}_j - \overline{a}_j^T x^{(k)}}{\overline{a}_j^T d^{(k)}}$$

- If $d^{(k)}$ is a positive curvature direction, then the steplength is calculated by:

  $$\rho_k = \min\{1, \rho_{strict}, \rho_{obj}\}$$

  If $\rho_k < 1$, then it is associted with a constraint and as we do in Section 2.5, we add it to the working set, $\mathcal{W}_k$, modify the QR factors and compute the new factors for the new reduced Hessian.

  In the other hand, if $\rho_k = 1$ and it is not associated wih any constraint, we take $x^{(k+1)} = x^{(k)} + d^{(k)}$ and the next iteration starts.

- If we have computed a null curvature direction and the index set used in (3.9) is empty, then it is possible that the objective function is not bounded below in this direction. To check this, assuming that the steplengths ρ_j, where $j \in (V_{k,1} \cup V_{k,2})$, are ordered from smallest to largest, being s the number of elements of the set, we use the following algorithm:

  Be $\delta = (g^{(k)})^T d^{(k)}$ and j=1.

  1. *If $\delta > 0$, then STOP.*
  2. *In other case*
     - *If j=s, then the problem is not bounded below. STOP.*
     - *In other case, j=j+1 and $\delta = \delta + \nu_j \overline{a}_j^T d^{(k)}$, where*
       - *$\nu_j = -2 \quad if \quad \overline{a}_j^T x^{(k)} > \overline{b}_j$ and $j = 1, \ldots, n_{IF}$.*

$*$ $\nu_j = -1 \quad if \quad \overline{a}_j^T x^{(k)} > \overline{b}_j$ *and* $j > n_{IF}$.

$*$ $\nu_j = +1 \quad if \quad \overline{a}_j^T x^{(k)} < \overline{b}_j$ *and* $j > n_{IF}$.

$*$ $\nu_j = +2 \quad if \quad \overline{a}_j^T x^{(k)} < \overline{b}_j$ *and* $j = 1, \ldots, n_{IF}$.

Go to Step 1.

If in (3.9) the set I_k is not empty or the previous algorithm stops in Step 1, the problem is bounded in the direction of $d^{(k)}$ and we take the final steplength

$$\rho_k = \min\{\rho_{estric}, \rho_{obj}\}. \tag{3.10}$$

- If $d^{(k)}$ is a negative curvature direction and the index set of (3.9) is empty the problem is not lower bounded. Indeed using (2.4) and $(d^{(k)})^T H d^{(k)} < 0$

$$f(x^{(k)} + \rho d^{(k)}) \longrightarrow -\infty \text{ when } \rho \longrightarrow +\infty$$

In other case we take the steplength as in (3.10).

3.4 Calculation of a feasible point

The active-set methods need to start at one initial feasible point. Since it is not always easy to find a feasible point, we have to use some strategy to achive one and there are several approaches for it. For instance, it is posible to calculate an initial feasible point, $x^{(0)}$, through the resolution of a particular case of a generalized quadratic problem. Given a (GQP) problem, we keep the equality constraints in the feasible set while the inequality constraints are included in the objective function in the following way:

$$(P_{aux}) \begin{cases} \text{Minimize } \widetilde{F}(x) = \displaystyle\sum_{j=n_I+1}^{n_I+n_D} (a_j^T x - b_j)_+ \\ x \in \mathbb{R}^n \\ \text{subject to} \\ \qquad\qquad a_j^T x = b_j \ , \quad j = 1, \ldots, n_I. \end{cases}$$

Proposition 3.4.1. *If there are feasible points for the problem (GQP), then there exists at least one solution. Moreover, if $x^{(0)}$ is a solution of (P_{aux}), then $x^{(0)}$ is a feasible point for (GQP).*

Proof. Let us assume that $\hat{x}$ is a feasible point for (GQP), then it is also a feasible point for (P_{aux}), since $a_j^T \hat{x} = b_j$ for $j = 1, \ldots, n_I$. Furthermore, we have that $a_j^T \hat{x} - b_j \leq 0$, for $j = n_I + 1, \ldots, n_I + n_D$ and, then $\widetilde{F}(\hat{x}) = 0 \leq \widetilde{F}(x)$ for all x feasible for (P_{aux}). Then, $\hat{x}$ is a global solution for (P_{aux}).

Finally, If $x^{(0)}$ is a solution for (P_{aux}), using the convexity of the problem, it is a global solution and, then $\widetilde{F}(x^{(0)}) = 0$ and therefore $x^{(0)}$ is a feasible point for (GQP). □

This procedure to find an inital feasible point has some advantages to be used as a first step for the GQP programming algorithm. In fact, it will provide an associated working set and its corresponding QR factorization.

In the Table 3.1 we present some feasible initial vectors calculated by this procedure for some of the problems of Section 2.5.1.

Example	n_I	n_{DF}	iter	Starting point	feasible point
1	0	23	8	$(0,\ldots,0)^T$	$(1,2,3,4,5,6,7,8)^T$
2	1	1	2	$(1,0,0,-1)^T$	$(0.90909,0.36364,-0.18182,-0.909090)^T$
3	0	7	3	$(0,0,0,0,0)^T$	$(0,0,0,5,-5)^T$
4	0	2	2	$(20,\ldots,20)^T$	$(10,\ldots,10)^T$
5	0	200	3	$(2,0,\ldots,0,2)^T$	$x_1 = x_{100} = 1, x_{82} = -1, x_i = 0$ otherwise

Table 3.1: Computing feasible points with (P_{aux}).

Bibliography

[1] http://qplib.zib.de/instances.html. Last visited: 19/05/2021.

[2] https://es.mathworks.com/help/optim/ug/quadprog.html. Last visited: 18/05/2021.

[3] https://runebook.dev/es/docs/octave/quadratic-programming. Last visited: 31/05/2021.

[4] E. Balder. Notes of Ph.D. course "Convex Analysis for Optimization". https://www.researchgate.net/publication/262562854_On_subdifferential_calculus, September,2008.

[5] J. Bunch and L. Kaufman. A computational method for the indefinite quadratic programming problem. *Linear Algebra Appl.*, 34:341–370, 1980.

[6] E. Casas and C. Pola. An algorithm for indefinite quadratic programming based on a partial cholesky factorization. *RAIRO - Operations Research - Recherche Opérationnelle*, 27(4):401–426, 1993.

[7] L. Contesse. Une caractérisation complète des minima locaux en programmation quadratique. *Numer. Math.*, 34(3):315–332, Sept. 1980.

[8] B. N. Datta. *Numerical Linear Algebra and Applications, Second Edition.* Society for Industrial and Applied Mathematics, USA, 2nd edition, 2010.

[9] P. Gill, W. Murray, M. Saunders, and M. Wright. Inertia-controlling methods for general quadratic programming. *SIAM Review*, 33(1):1–36, 1991.

[10] P. Gill and W.Murray. Numerically stable methods for quadratic programming. *Math. Program.*, 14:349–372, 1978.

[11] C. Laguillo. Técnica de optimización para la recuperación de imágenes. TFG para obtener el Grado en Matemáticas en la Universidad de Cantabria, 2015.

[12] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer, New York, NY, USA, second edition, 2006.

[13] C. Pola. *Algoritmos numéricos para la resolución de problemas de optimización con restricciones.* PhD thesis, Dpt. Matemáticas, Estadística y Computación, Universidad de Cantabria, 1992.

Appendices

Appendix A

Background material

A.1 Some definitions

We consider the following problem:

$$(P)\begin{cases}\min F(x)\\ x \in K \subset \mathbb{R}^n\end{cases}.$$

where

$$K = \{x \in \mathbb{R}^n : h_i(x) = 0, i = 1, \ldots, n_I;\ \ g_j(x) \leq 0, j = 1, \ldots, n_D\}.$$

Definition A.1.1. *A vector $x \in \mathbb{R}^n$ is a **global minimum** for the problem (P) if it verifies that*

$$F(x) \leq f(y), \quad \textit{for all } y \in K.$$

Definition A.1.2. *A vector x is a **local minimum** (strict) for a problem (P) if there is an open neighborhood U of x such that*

$$F(x) \leq f(y), \quad \textit{for all } y \in K \cap U,$$

$$(F(x) < f(y), \quad \textit{for all } y \in K \cap U.$$

Definition A.1.3. *A **convex quadratic problem** is a linear constrained optimization problem where the Hessian of the objective function is a symmetric positive definite matrix.*

Definition A.1.4. *A point x is a **feasible point** for an optimization problem (P) if it satisfies all the constraints of the problem.*

Definition A.1.5. *A problem with **linear constraints** verifies that K is as follows:*

$$K = \{x \in \mathbb{R}^n : a_j^T x = b_j, j = 1, \ldots, n_I;\ \ a_j^T x \leq b_j, j = n_I + 1, \ldots, n_I + n_D\}$$

Definition A.1.6. *The **bound constraints** are upper or lower limits which restrict the components of the solution x.*

Definition A.1.7. *A **stationary point** is a point x that meets $Z_k^T \nabla F(x^{(k)}) = 0$.*

Theorem A.1.1. *(**Cholesky decomposition**) Be $A \in \mathbb{R}^{n\times n}$, then the following statements are equivalent:*

- *A is a symmetric positive definite matrix.*
- *There exist a unique lower triangular matrix $L \in \mathbb{R}^{n\times n}$ with positive diagonal entries such that*

$$A = LL^T.$$

A.2 Optimality conditions for quadratic programming

First-order Optimality conditions

Theorem A.2.1. *Given the following problem*

$$(GQP)\begin{cases} \text{Minimize } F(x) = \dfrac{1}{2}x^T Hx + p^T x \\ \text{subject to } x \in \mathbb{R}^n \\ \qquad a_j^T x = b_j \ , \quad j = 1, \ldots, n_I; \\ \qquad a_j^T x \leq b_j \ , \quad j = n_I + 1, \ldots, n_I + n_D \ ; \end{cases}$$

be $\overline{x}$ a feasible point. If $\overline{x}$ is a solution of (GQP), then there exists a Lagrange multiplier vector $\lambda \in \mathbb{R}^{n_I+n_D}$ such that

$$\nabla f(\overline{x}) + \sum_{j=1}^{n_I+n_D} a_j^T \lambda_j = 0; \tag{A.1}$$

$$\lambda_j(a_j^T \overline{x} - b_j) = 0, \ \ j = n_I + 1, \ldots, n_I + n_D; \tag{A.2}$$

$$\lambda_j \geq 0, \ \ j = n_I + 1, \ldots, n_I + n_D; \tag{A.3}$$

Moreover if F is a convex function, the reciprocal is also true.

Necessary Second-order Optimality conditions

Theorem A.2.2. *Given the problem (GQP) from the previous Theorem, $\overline{x}$ a solution of (GQP) and*

$$\overline{J} = \{j \in \{1, \ldots, n_I + n_D\} : a_j^T \overline{x} - b_j = 0\},$$

then there are two vectors $\overline{\lambda} \in \mathbb{R}^{n_I}$ and $\overline{\mu} \in \mathbb{R}^{n_D}$ such that

$$H\overline{x} + p + \sum_{j=1}^{n_I} a_j \overline{\lambda}_j + \sum_{j=n_I+1}^{n_I+n_D} a_j \overline{\mu}_j = 0; \tag{A.4}$$

$$\overline{\mu}_j \geq 0, \overline{\mu}_j(a_j^T \overline{x} - b_j) = 0, \ \ j = n_I + 1, \ldots, n_I + n_D; \tag{A.5}$$

$$a_j^T \overline{x} - b_j = 0, \ \ j = 1, \ldots, n_I, \quad a_j^T \overline{x} - b_j \leq 0, \ \ j = n_I + 1, \ldots, n_I + n_D; \tag{A.6}$$

$$d^T H d \geq 0, \ \textit{for all } d \in \mathbb{R}^n \textit{ such that } a_j^T d = 0, \textit{ if } j \in \overline{J}. \tag{A.7}$$

A.3 Notations

We present in this section some notation that have been used in the text.

- $\mathbb{R}_+ = \{x \in \mathbb{R} : x > 0\}$.
- Be $x, y \in \mathbb{R}^n$, $x \geq y$ if and only if $x_i \geq y_i$, for all $i = 1, \ldots, n$.
- $\mathcal{W}_k = \{j : a_j^T x - b_j = 0 \text{ and } \{a_j\}_{j \in \mathcal{W}_k} \text{ linearly independent}\}$ is the working set.
- Id is the identity matrix

A.4 Code list

In this section we present all the MATLAB functions that we have programmed and a brief description about their funcionality.

- anrs02.m : Solve a system as $(P^T R^T RP)x = b$, where P is a permutation matrix and R is an upper triangular matrix.
- cholpar.m: Calculate the partial or total Cholesky factorization for a matrix as $PZ^T HZP^T$, where P is a permutation matrix,

  If the matrix $Z^T HZ$ is positive definite the Cholesky decomposition (LL^T) will be used, if it is positive semidefinite and singular (2.20) will be done and in indefinite case,the factorization (2.26) will be computed.
- qpdes.m: Calculate a descent direction for a quadratic problem.
- qpstep.m: Calculate the lengthstep along the descent direction.
- qr1.m: Modify the QR factors of the working set matrix A_k when a constraint is added to the working set.
- qr2.m: Modify the QR factors of the working set matrix A_k when a constraint is removed from the working set.
- quaprog.m: Solve a quadratic optimization problem.
- sollowtri.m: Solve a system as $Lx = y$, where L is a lower triangular matrix.
- solupptri.m: Solve a system as $Rx = y$, where R is an upper triangular matrix.
- aux003.m: Depending on an initial parameter it adds to the working set one active constraints associated to the objective function and it updates the QR factors or it simply updates the initial working set.

A.5 Qpdes code

Now we present the code of qpdes

```
function [d,id]=qpdes(Z,LT,g,a,w,ipvt,n,ng,ind,info,id)
%%%%%%%%%%%%
%Author:Carlos Crespo under the direction of Cecilia Pola
%Date: 28/1/2021
%Reference: desr03 FORTRAN function of Eduardo Casas and Cecilia Pola.
%
% Purpose : this program calculates a descent direction for a quadratic
% problem
%
%Input List:
%-LT: An nxn upper triangular matrix which is calculated in cholpar
%-Z: An ngxn matrix whose columns form a basis of the null space of
%    the matrix associated to the working set Ak'
%-a: An  ng vector which contains the coefficients of the last
%    constraint removed from the active set if id<-ng or id>ng.
%-w: A work vector of dimension 2*N. If info=10 in the first n
%    coordinates it has the reduced gradient
%-ipvt: A vector of dimension n containing ipvt vector of cholpar
%-n: Number of columns of the matrix Z
%-ng: Dimension of gradient vector
%-ind: A number that indicates which type of factorization has been
%      produced in cholpar
%-info: This variable indicates depending on its value:
%        *0: The projected gradient is any vector
%        *1: The projected gradient is a scalar multiple
%            of the n-th vector of the canonical basis
%        *10: The projected gradient is given
%-id: This variable indicates dependenig on its value
%       *<> 0:
%        *0: In other case
%
%Output List:
%-d: Vector containing the descent direction
%-id: Variable which indicates how the descent direction has been calculated
%     *0: negative or null curvature direction
%     *1: other case
%%%%%%%%%%%%
Z=Z(:,n:-1:1);d=zeros(ng,1);eps1=eps^.75;indmul=id;id=0;s=0;%
%In the case of ind>=0 the projected gradient is calculated (positive
% semidefinite case)
if ind>=0
    if info==0
        w(1:n)=-Z(:,1:n)'*g;
    elseif info==1
        wn=-Z(:,n)'*g;
    elseif info==10
        info=0;
        w(1:n)=-w(1:n);
    end
end
% Projected Hessian positive definite
if ind==0
    id=1;
    if info==0
        [w(1:n),ifallo]=ansr02(LT(1:n,1:n),w(1:n),ipvt);
        if ifallo==1
            return;
        end
    else
        y=zeros(n,1);y(n)=wn/LT(n,n);
        [w(1:n),ifallo]=solupptri(LT(1:n,1:n),y);
        if ifallo==1
            return;
        end
```

```
    end
    % Projected Hessian indefinite. In this case a direction of negative
    % curvature is calculated
elseif ind< -1 && ind>=-n
    m=-ind;m2=m-1;B=-LT(1:m2,m);
    [d(1:m2),ifail]=solupptri(LT(1:m2,1:m2),B);
    if ifail==0
        w(1:n)=0;
        for i=1:m2
            w(ipvt(i))=d(i);
        end
        w(ipvt(m))=1;
    else
        fprint('Error');
        return;
    end
elseif ind<-n
    m=-ind-n;m1=m+1;Hrs=LT(m,m1);B=-LT(1:m-1,m1)+Hrs*LT(1:m-1,m);
    if m>1
        [d(1:m-1),ifail]=solupptri(LT(1:m-1,1:m-1),B);
        if ifail==1
            return
        end
    end
    d(m)=-Hrs;
    for i=1:m
        w(ipvt(i))=d(i);
    end
    w(ipvt(m1))=1;
    % Hessian positive semidefinite case
elseif ind>0 && ind<n
    k=0;m=n-ind;
    if info==0
        w(n+1:2*n)=w(ipvt(1:n));
    else
        i=1;
        while n~=ipvt(i)
            i=i+1;
        end
    end
    % A base of the projected Hessian null space is computed
    for j=1:ind
        mj=m+j;
        [w(1:m),ifail]=solupptri(LT(1:m,1:m),LT(1:m,mj));
        if ifail==1
            return;
        end
        if info==0
            s=w(1:m)'*w(n+1:n+m)-w(mj+n);
        else
            if i==mj
                s=-wn;
            elseif i<=m
                s=w(i)*wn;
            else
                s=0;
            end
        end
        if abs(s)>eps1
            k=1;d(1:m)=d(1:m)+s*w(1:m);d(mj)=-s;
            for i=1:n
                w(ipvt(i))=d(i);
            end
        end
    end
    %If k=0 means that the projected gradient is orthogonal to the base of
    %the null space. A descent direction is computed in this case
    if k==0
        id=1;
```

```
134         if info==0
135             for i=1:n
136                 w(ipvt(i))=w(n+i);
137             end
138             [w(1:n),ifail]=ansr02(LT(1:m,1:m),w(1:n),ipvt(1:m));
139             if ifail==1
140                 return
141             end
142             w(ipvt(m+1:n))=0;
143         else
144             d(m)=-1;d(1:m-1)=0;
145             [d(1:m),ifail]=solupptri(LT(1:m,1:m),d(1:m));
146             if ifail==1
147                 return
148             end
149             d(m+1:n)=0;
150             for i=1:n
151                 w(ipvt(i))=d(i);
152             end
153         end
154     end
155 end
156 % The obtained direction is projected
157 if ind==n && info==1
158     if abs(wn)>eps1
159         d(1:ng)=wn*Z(1:ng,1);
160     else
161         id=1;
162         d(1:ng)=0;
163     end
164 elseif ind==-1
165     d(1:ng)=Z(1:ng,ipvt(1));
166 else
167     if ind==n
168         s=norm(w(1:n));
169         if s<=eps1
170             id=1;d(1:ng)=0;
171         end
172     end
173     if ind~=n || (ind==n && id==0)
174         d(1:ng)=Z(1:ng,:)*w(1:n);
175     end
176 end
177 % The direction calculated is affected with the appropriate sign to
178 % turn it a descent direction
179 if ind<0 || (id==1 && info ==1 && ind>0)
180     if id==1
181         s=d'*g;
182         if s>0
183             d(1:ng)=-d(1:ng);
184         else
185             s=-s;
186         end
187     else
188         if indmul==0
189             s=d'*g;
190         else
191             if indmul >ng || indmul <-ng
192                 s=d'*a;
193             else
194                 s=d(abs(indmul));
195             end
196         end
197         if (indmul>=0 && s>0)||(indmul<0 && s<0)
198             d(1:ng)=-d(1:ng);
199         end
200     end
201 end
202 end
```