# Ongoing work on MCWAL

Pierre Borie*

## Contents



**Abstract**

This informal document deals with the ongoing work and thinking on a algorithm for constrained nonlinear least squares. The current algorithm name is MCWAL for Moindres Carrés With Augmented Lagrangian. Some sections about more general aspects about nonlinear programming might also appear here and there. The aim of this document is to reflect the thinking and understanding of different kinds of notions.

*University of Montreal, Department of Computer Science and Operations Research, Montreal, QC, Canada

# 1 Introduction

We consider least squares problems subject to both nonlinear and linear constraints of the form

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & \frac{1}{2}\|r(x)\|^2 \\ \text{s.t.} \quad & h(x) = 0 \\ & \langle c_i, x\rangle = b_i, \quad i = 1,\dots,m \\ & l \le x \le u, \end{aligned} \tag{1.1}$$

where $r\colon \mathbb{R}^n \to \mathbb{R}^d$ and $h\colon \mathbb{R}^n \to \mathbb{R}^t$ are assumed to be nonlinear, potentially non convex, continuously differentiable functions, $\langle\cdot,\cdot\rangle$ is the canonical inner product and $\|\cdot\|$ its induced euclidean norm, $c_i$ are $m$ independent vectors of $\mathbb{R}^n$, $(m \le n)$, $b = (b_1,\dots,b_m)^T \in \mathbb{R}^m$ and $\ell$ and $u$ are vectors in $\mathbb{R}^n$. Without loss of generality, some components of the latter two vectors can be set to $\pm\infty$ for unbounded parameters. In the context of least squares problems, components $r_i$ of the function $r$ are often denoted as the residuals.

We will also refer to the linear constraints using the following set notation

$$\mathcal{X} = \{x \in \mathbb{R}^n \mid Cx = b,\ l \le x \le u\}, \tag{1.2}$$

where $C$ is the matrix whose columns are the vectors $c_i$. By linear independence of those vectors, $C$ is a full rank matrix. The set $\mathcal{X}$ is thus convex.

# Notations

When considering iterative methods for solving problem (1.1), $k$ will, if not mentioned otherwise, refer to the iteration number. In order to simplify notations, quantities relative to a given iteration will be noted with the iteration number as an index, such as $x_k$ for the iterate, $f_k$ for the evaluation $f(x_k)$ etc.

Symbol := shall be used to state the definition of a numerical object (function, vector etc.)

For a subset $C$ of a metric space $E$, when considering a sequence $(x_k)_k \in E^{\mathbb{N}}$ converging to $x^* \in C$ when $k \to +\infty$, we will write $x_k \to_E x^*$ if, above a certain index, all $x_k$ are in $C$.

The set of linear combinations of a set of vectors $(v_1,\dots,v_n)$ is written $\rangle\ span(v_1,\dots,v_n)$.

$\mathcal{M}_{m,n}(\mathbb{K})$ denotes the set of matrices with $m$ rows and $n$ columns whose coefficients are in the field $\mathbb{K} = \mathbb{R}$ or $\mathbb{C}$. In practice, it will always be $\mathbb{R}$ and would be surprising to consider the complex case but one never knows. When $m = n$, i.e. for square matrices, we will write $\mathcal{M}_n(\mathbb{K})$. The adjoint of a matrix $A \in \mathcal{M}_{m,n}(\mathbb{K})$ is written $A^*$. In the real case, it corresponds to its transpose, in the complex case, it corresponds to the tranpose if its conjugate.

We also introduce notations for special subsets of matrices. $GL_n(\mathbb{K})$, also known as the linear group of order $n$, denotes the set of invertible matrices (or "non-singular"...). $O(n)$ is the orthogonal group, i.e. the set of matrices $Q$ in $\mathcal{M}_n(\mathbb{R})$ such that $Q^TQ = I_n$. $\mathcal{S}_n(\mathbb{K})$ denotes the set of matrices equal to their adjoint. In the real case, such matrices are called symmetric, in the complex case, they are called hermitian.

The condition number of a matrix $A \in \mathcal{M}_n(\mathbb{R})$, given by $\|A\|\ \|A^{-1}\|$ will be noted $\kappa(A)$.

If $H$ is a symmetric matrix, i.e. $H^T = H$, we will write $H \succ 0$, resp. $H \succeq 0$, when $H$ is positive definite, resp. semi-definite positive. In the case $H \succ 0$, the bilinear mapping $\langle\cdot,\cdot\rangle_H : (x,y) \mapsto \langle x, Hy\rangle$ forms an inner product of induced norm $\|\cdot\|_H : x \mapsto \sqrt{\langle x, Hx\rangle}$.

# 2 Optimality conditions for nonlinear programming

In this section, we describe optimality conditions for nonlinear programs more general than (1.1) from different views.

## 2.1 An algebraic view

We consider the general mathematical program[1]

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & h(x) = 0 \\ & l \leq x \leq u, \end{aligned} \tag{2.1}$$

with the same assumptions on of differentiability of functions $f$ and $h$. We introduce the Lagrangian associated to problem (2.1):

$$\ell(x, \lambda) = f(x) + \langle \lambda, h(x) \rangle \,, \tag{2.2}$$

where $\lambda$ is the vector of Lagrange multipliers. Algorithms discussed aim to find local minimum of problem (2.1), i.e. feasible and of minimal value in s neighborhood. Under suitable assumptions, one can establish necessary and even sufficient conditions for local optimality. One of the most important assumptions belongs to the familty of constraints qualifications. The following is the one we will employ.

**Definition 1.** ***(LICQ)*** *The LICQ holds at $x^*$ if the gradients of the equality constraints evaluated at $x^*$ are linearly independent. In other terms, the matrix $A(x)$ is full rank.*

Using this and standard differentiability assumptions, one can state first-order necessary optimality conditions, also known as KKT conditions.

**Definition 2.** ***(KKT conditions)*** *A point $x^*$ satisfies the KKT conditions if $x^*$ is feasible and there exists multipliers $\lambda^*$ such that*

$$\nabla_x \ell(x^*, \lambda^*) = 0.$$

A point satisfying those conditions if also said to be a KKT point or a first order critical point for problem (2.1). Depending on the context, we would refer to a KKT point either by just wrting $x^*$ or the couple formed after $x^*$ and its associated Lagrange multiplier $\lambda^*$The necessary conditions follow.

**Theorem 3.** ***(First Order Necessary Conditions)[10, Theorem 12.1]*** *Let $x^*$ be a local solution of (2.1) at which the LICQ holds. Then $x^*$ is a KKT point.*

Sufficient optimality conditions can be established using second order information.

**Definition 4.** ***(Second Order Conditions)*** *A point $(x^*, \lambda^*)$ satisfies the second order conditions if it is a KKT point for (2.1) at which the LICQ holds and if the matrix $\nabla^2_{xx}\ell(x^*, \lambda^*)$ is positive definite on the null space of the constraints Jacobian, i.e.:*

$$\forall w \text{ verifying } \langle A(x^*), w \rangle = 0, \ \left\langle w, \nabla^2_{xx}\ell(x^*, \lambda^*)w \right\rangle > 0.$$

**Theorem 5.** ***(Second Order Sufficient Conditions) [10, Theorem 12.5]*** *If $x^*$ satisfies the second order conditions, then $x^*$ is a local minimum of (2.1).*

[1] Shall I write the KKT conditions w.r.t. this formulation and then consider multipliers for the bounds?

## 2.2 A geometric view

We now consider the general program

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & x \in \mathcal{C}, \end{aligned} \tag{2.3}$$

where the feasible set $\mathcal{C}$ is a (of course non empty), subset of $\mathbb{R}^n$. The idea is to describe optimality conditions using inclusions of certain vectors to sets and cones. General results can be stated without any further assumptions on $\mathcal{C}$. Yet, we will give important results about projections that require to assume $\mathcal{C}$ is convex. As we shall see in section 5, our method will imply solving quadratic programs subject to linear, hence convex, constraints. Most of the content exposed is from [10, Chapter 12].

We start by defining two important sets, that are actually cones. We remind that a set $K$ is a cone if for all $x \in K$ and $\alpha > 0$, $\alpha x \in K$. A cone is pointed if it contains 0.

**Definition 6.** ***Tangent Vector*** *Let $x \in \mathcal{C}$. A vector $v \in \mathbb{R}^n$ is said to be a tangent vector to $\mathcal{C}$ at $x$ if there are sequences $(x_k)_k \to_{\mathcal{C}} x$ and $(t_k)_k \searrow 0^+$ such that*

$$\lim_{k\to\infty} \frac{x_k - x}{t_k} = v.$$

**Definition 7.** *The **tangent cone** is the set of all tangents vectors to $\mathcal{C}$ at $x$ and is denoted by $T_{\mathcal{C}}(x)$.*

For a better understanding, the limit in the definition can be written as the relation

$$x_k - x = t_k v + o(1).$$

Intuitively, tangent vectors are all the directions going from $x$ that stay in $\mathcal{C}$, up to a scaling by a very small positive factor. In the context of constrained optimization, they characterize directions in which one can step away from $x$ while remaining feasible.

**Remark**

Rigorously, it is a $o(t_k)$ but since $t_k \to 0$, a $o(t_k)$ is also a $o(1)$. Our intent is to highlight the visualization of a tangent vector as a scaled $x_k - x$ for $(x_k)_k \to x$.

**Definition 8.** *The **normal cone** to $\mathcal{C}$ at $x$ is the set defined by*

$$N_{\mathcal{C}}(x) := \{w \in \mathbb{R}^n \mid \forall v \in T_{\mathcal{C}}(x), \langle v, w\rangle \leq 0\}.$$

The normal cone is merely the orthogonal complement of the tangent cone and is involved in the following first order optimality condition.

**Theorem 9.** *If $x^*$ is a local minimum of $f$, then*

$$-\nabla f(x^*) \in N_{\mathcal{C}}(x^*).$$

This theorem reflects the intuitive fact that a local minimum is a point from which the objective function cannot be reduced while remaining feasible. While this result is simple is elegant, its use is limited by the difficulty to express both tangent and normal cones without any other assumptions. For instance, with the LICQ, the tangent cone can be expressed as an intersection of hyperplanes depending on the constraints gradients and combining theorem 9 with a Farkas-like lemma gives the KKT conditions. Similar consequences occur when the feasible set is convex, which we will assume from now on.

We first remind the definition a convex set.

**Definition 10.** *A set $\mathcal{C} \subseteq \mathbb{R}^n$ is **convex** if $\forall (x, y) \in \mathcal{C}^2$ and $\alpha \in [0, 1]$, one has $\alpha x + (1-\alpha)y \in \mathcal{C}$. In other words, the line segment between two points remains in $\mathcal{C}$, for all points of $\mathcal{C}$.*

In this setting, the normal cone can be described more simply.

**Proposition 11.** *Let $\mathcal{C}$ a non empty convex set. Then for all $x \in \mathcal{C}$:*

$$N_{\mathcal{C}}(x) = \{v \mid \forall y \in \mathcal{C}, \langle v, x - y\rangle \geq 0\}.$$

*For $x \notin \mathcal{C}$, $N_{\mathcal{C}}(x) = \emptyset$.*

Then, if $x^*$ is a local minima of (2.3), the inclusion in theorem 9 becomes

$$\forall x \in \mathcal{C}, \langle \nabla f(x^*), x - x^*\rangle \geq 0. \tag{2.4}$$

**Remarks on** (2.4)

1. This condition if sufficient when the function $f$ is convex
2. It also reflects the very visual fact that in $\mathbb{R}^2$, the line tangent to a local minima is below the curve

So we now have a more simple way to characterize a minimum but it is still not suited to be used in algorithms. Obviously, one cannot check the sign of an infinite number of inner product but more fundamentally, algorithm must be able to manipulate points that are likely to be outside of the feasible set. Using projections is the key.

**Definition 12.** *Let $\mathcal{C} \subseteq \mathbb{R}^n$. The **projection operator** is the set-value mapping $P_{\mathcal{C}} \colon \mathbb{R}^n \rightrightarrows \mathcal{C}$ defined by*

$$P_C(\bar{x}) = \operatorname*{argmin}_{y \in \mathcal{C}} \frac{1}{2}\|\bar{x} - y\|^2.$$

For this definition, we introduce the notion of set-value mapping [11] to remain general but in the convex case, it is single valued and well defined if the concerned set is closed.

**Theorem 13.** *Let $\mathcal{C}$ be a close convex set and $x \in \mathbb{R}^n$. Then*

$$\bar{x} = P_{\mathcal{C}}(x) \iff \forall y \in \mathcal{C}, \langle \bar{x} - x, y - \bar{x} \geq 0.\rangle$$

Since the norm is convex, this theorem is simply derived from the first order necessary optimality conditions. Also the inequality kind of looks like the inequality in proposition (11), which motivates the following theorem.

**Theorem 14.** *Let $\mathcal{C}$ be a close convex set, $x \in \mathbb{R}^n$ and $z \in \mathcal{C}$. Then*

$$z = P_{\mathcal{C}}(x) \iff (x - z) \in N_{\mathcal{C}}(z).$$

From this result, one deduces the following useful corollary.

**Corollary 15.** *Let $\mathcal{C}$ be a close convex set, $x \in \mathcal{C}$ and $z \in \mathbb{R}^n$. If $z \in N_{\mathcal{C}}(x)$, then $\forall t \geq 0$, $P_{\mathcal{C}}(x + tz) = x$.*

Applying the above corollary to opposite of the gradient of a local minimum, thus verifying theorem 9, gives the next theorem.

**Theorem 16.** *Consider the problem* (2.3) *where the non empty feasible set* $\mathcal{C}$ *is closed and convex. Then* $x^*$ *is a local minimum if and only if for all* $t \geq 0$, $P_{\mathcal{C}}\left(x^* - t\nabla f(x^*)\right) = x^*$.

There is our desired result that we shall use as a termination criteria, up to a small tolerance. In practice, we will work with the Cauchy arc defined as

$$p(t, x) := P_{\mathcal{C}}(x - t\nabla f(x)), \tag{2.5}$$

that is the path alongside the steepest descent direction. The Cauchy point is the point minimizing the objective function on the Cauchy arc, i.e.:

$$x^C := x - t^C \nabla f(x) \text{ where } t^C = \underset{t \geq 0}{\operatorname{argmin}} f\left(p(t, x)\right). \tag{2.6}$$

If an algorithm uses the Cauchy point as the new iterate, then the algorithm would converge. Moreover, if each iterate reduces the objective function as much as a fraction of the Cauchy point, the algorithm would also converge. This pretty much sets the foundation, and intuition, for the theory of convergence for algorithms producing inexact solutions of the sub problems, especially for trust regions methods [2]. In a less formal way, it translates into a theoretical notion the practical aspect that computers are by nature inexact, because of floating-point arithmetic, and thus we are more interested into proving that "not exact but good enough" solutions will do the job. An algorithm that does not converge if solutions are not exact might be nice on paper but will not be very useful in practice.

# 3 The conjugate gradient method

In this section, we describe a very popular iterative method in optimization, the conjugate gradient (CG) method. We will try to give a comprehensive review and intuitive view of the method and its main characteristics.

## 3.1 The main algorithm

Let $H \in \mathcal{S}_n(\mathbb{R})$ such that $H \succ 0$, $c \in \mathbb{R}^n$ and consider the quadratic program (QP)

$$\min_x \; q(x) = \frac{1}{2}\langle x, Hx\rangle - \langle c, x\rangle. \tag{3.1}$$

This formulation will often occur in our case when we will formulate sub-problems from a quadratic model of the objective function.

By positive-definiteness of $H$, program (3.1) is convex and thus has a unique minimizer $x^*$ verifying $\nabla q(x^*) = 0$, i.e.

$$Hx^* = c. \tag{3.2}$$

Solving a convex QP is then equivalent to solving a symmetric linear system. The question is now how to solve it? A first approach could be to factorize $H$ to transform system (3.2) into linear systems easier to solve, by using the Cholesky factorization or the QR decomposition for instance. See [5] for details on how to compute them. Using such direct methods can still be prohibitive in computations and storage[2]. The conjugate gradient method has the advantage to be iterative, to only require matrix-vector products and to converge in, at most, $n$ iterations

[2]Expand this paragraph to tell how exact methods can have issues

(modulo zero precision arithmetic). Before describing the principle of the method, we will introduce the notion of conjugate vectors w.r.t. a symmetric matrix.

**Definition 17.** *Let $H \succ 0$. Two vectors $u$ and $v$ are said to be $H$-conjugate if $\langle u, Hv\rangle = 0$.*

Visually, two $H$-conjugate vectors are orthogonal in the space obtained obtained after applying $H$ to $\mathbb{R}^n$. This definition naturally extends to a set of vectors $(p_i)_{i\in I}$ who are said to be $H$-conjugate if $\langle p_i, Hp_j\rangle = 0$ for all $(i,j) \in I^2$ such that $i \neq j$. If the matrix $H$ is implicit, we will just say "conjugate".

An interesting fact is that $k \leq n$ conjugate vectors are also linearly independent. Thus, a set of $n$ conjugate vectors form a basis of $\mathbb{R}^n$. Let's say that we initiate our iterative process from a given $x_0$ from which we seek to compute our solution $x^*$. If $(p_0, \ldots, p_{n-1})$ is a set of conjugate vectors, then there are scalars $(\alpha_0, \ldots, \alpha_{n-1})$ such that

$$x^* - x_0 = \sum_{i=0}^{n-1} \alpha_i p_i.$$

Using the conjugacy property, one can find that the scalars are actually determined by the relation

$$\alpha_k = \frac{\langle p_k, H(x^* - x_0)\rangle}{\langle p_k, Hp_k\rangle}.$$

The problem is that this relation depends on the solution that we are actually looking for and do not have access to[3]. However, this issue can be solved by introducing, for $k \geq 1$, the intermediates iterates

$$x_k = x_0 + \sum_{i=0}^{k-1} \alpha_i p_i.$$

The scalars can now be expressed

$$\alpha_k = -\frac{\langle p_k, r_k\rangle}{\langle p_k, Hp_k\rangle}, \tag{3.3}$$

where $r_k := Hx_k - c$ is the $k$-th residual. By the iterative relation $x_k = x_{k-1} + \alpha_{k-1}p_{k-1}$, one also has $r_k = r_{k-1} + \alpha_{k-1}Hp_{k-1}$.

The latter relation can also be derived from performing a linesearch along the direction $p_k$ by computing $\alpha_k = \operatorname{argmin}_\alpha q(x_k + \alpha p_k)$. Since $q$ is quadratic, there is an explicit formula that also gives the relation (3.3). Yet, we prefer to view this process as an iterative decomposition of the vector $x^* - x_0$ onto a basis of conjugate vectors. For instance, it is trivial that this process converges in $n$ iterations because each iteration consists in computing a coefficient of the decomposition.

In any case, the coefficients can now be computed recursively, provided we have access to conjugate directions, which is not a trivial question. If one knows its linear algebra classics, since $H$ is symmetric, by the spectral theorem, there is an orthonormal basis of eigen vectors into which $H$ is diagonal. With such vectors, the conjugacy property is clearly satisfied so this basis could do the job. The problem is that diagonalizing a matrix is a very expensive process on its own that could even exceed the cost of a relevant factorization of $H$.

[3] if we did, why solving the problem?

Also note that having a base diagonalizing $H$ is stronger than what we actually need. Indeed, using matrix notations, let $P$ be the matrix whose columns are simply conjugate vectors $p_1, \ldots, p_n$ and $G \in O(n)$ such that $H = GDG^T$ for some diagonal matrix $D$. Then, because $G^TG = I_n$, one has $G^THG = D$, whereas definition 17 simply implies that $P^THP$ is diagonal. The latter is weaker, because there is absolutely no reasons for $P$ to be in $O(n)$. Of course, $P$ is invertible but its inverse is not necessarily its transpose.

Another method would be to apply the Gram-Schmidt orthonormalization process to a set of given vectors. It would work but would still require storing all the vectors, which is less suited for large-scale applications.

Fortunately, there is a way to produce each direction iteratively in such a way that only accessible information about the problem and the previous basis vector are required to compute a new basis vector. The initial information will consist of the residuals $r_k$. Let $x_0$ be our starting point, $r_0 = Hx_0 - c$. For $k \geq 1$, every new basis vector is defined by

$$p_k = -r_k + \beta_{k-1} p_{k-1}$$

where $\beta_{k-1}$ is a conjugacy parameter whose expression can be derived from imposing $\langle p_{k-1}, Hp_k \rangle = 0$ and injecting the above expression:

$$\beta_{k-1} = \frac{\langle p_{k-1}, Hr_k \rangle}{\langle p_{k-1}, Hp_{k-1} \rangle}.$$

Combined with the recursion previously described, this process respects our conditions to only require accessible information about the previous iteration and ensures the following properties:

- $\forall i < j,\ \langle r_i, r_j \rangle = 0$
- $\forall i \neq j,\ \langle p_i, Hp_j \rangle = 0$ ($H$-conjugacy).
- $\forall k,\ span(r_0, \ldots, r_k) = span(p_0, \ldots, p_k) = span(r_0, Hr_0, \ldots, H^{k-1}r_0)$

The space $span(r_0, Hr_0, \ldots, H^{k-1}r_0)$ is the Krylov subspace of degree $k$, generally written $\mathcal{K}(H, r_0, k)$, is a the foundation of a category of iterative methods called Krylov methods, of which the CG method is a special instance.

Before describing the an algorithm, some relations can be modified such that the computations are even more efficient. First, with the conjugacy recursion, one has $\langle p_k, r_k \rangle = -\|r_k\|^2$, leading to

$$\alpha_k = \frac{\|r_k\|^2}{\langle p_k, Hp_k \rangle}.$$

Similarly, the conjugacy parameter is also equal to

$$\beta_k = \frac{\|r_{k+1}\|^2}{\|r_k\|^2}.$$

The CG method is stated in algorithm 1.

One can observe that each iteration only requires one matrix-vector product and inner products, making this method very efficient for large scale applications. To expand an earlier remark, the CG method belongs to the class of Krylov methods who consist in an iterative process for which, starting from $x_0$, each iterate is a minimizer of the objective function over

**Algorithm 1** The conjugate gradient method

**Require:** Starting point $x_0$
  $r_0 \leftarrow Hx_0 - c$ and $p_0 \leftarrow -r_0$
  **for** $k = 0, \dots, n-2$ **or** until convergence **do**
    $\alpha_k \leftarrow \frac{\|r_k\|^2}{\|p_k\|_H^2}$
    $x_{k+1} \leftarrow x_k + \alpha_k p_k$
    $r_{k+1} \leftarrow r_k + \alpha_k H p_k$
    $\beta_k = \frac{\|r_{k+1}\|^2}{\|r_k\|^2}$
    $p_{k+1} \leftarrow -r_{k+1} + \beta_k p_k$
  **end for return** $x_k$

a subspace of the form $\mathcal{K}(H, r_0, k)$, also called a Krylov subspace. We have only seen an application for square symmetric systems but the theory extend to rectangular and indefinite systems.[4]

As we already said, the CG method converges in at most $n$ iterations, but the efficiency of the method is highly related to the eigenvalues distributions of the matrix.

**Proposition 18.** *Let $H \in \mathcal{S}_n^{++}(\mathbb{R})$ such that $H$ has $r < n$ distinct eigenvalues, then algorithm 1 applied to problem* (3.1) *converges in at most $r$ iterations.*

In a similar fashion, if matrix $H$ has clustered eigenvalues, then the CG method will give a very good solution in a few iterations (as much as they are clusters).

A more numerical result [2, Theorem 5.1.7] states that, when applying algorithm 1, the norm decrease of the error $\epsilon_k := x_k - x_*$ satisfies the inequality

$$\frac{\|\epsilon_k\|}{\|\epsilon_0\|_H} \leq 2\left(\frac{\sqrt{\kappa(H)} - 1}{\sqrt{\kappa(H)} + 1}\right). \tag{3.4}$$

Then, if we could solve another linear system equivalent to (3.2) where the matrix has clustered eigenvalues, and thus a condition number closer to 1, the efficiency of the CG method would be improved. This is the principle of preconditioning, that we will now discuss about.

## 3.2 Preconditioning

The idea of preconditioning is to apply the CG method to a modified version of system (3.2) after applying a change of variables of the form $x = R^{-1}x$, with $R \in GL_n(\mathbb{R})$ is a *preconditioner*. Rewriting the system w.r.t. $\bar{x}$ gives

$$\bar{H}\bar{x} = \bar{c}, \tag{3.5}$$

where $\bar{H} := R^{-T}HR^T$ and $\bar{c} := R^{-T}c$. This implies that the preconditioner should be chosen such that the transformed Hessian $\bar{H}$ has a better eigenvalue distribution than $H$.

One could directly apply algorithm 1 to system (3.5), to get the solution $\bar{x}^*$ and then back transform to obtain the solution of the original problem $x^* = R^{-1}\bar{x}^*$. On paper this works but it has the disadvantage to require an explicit computation of $\bar{H}$, which we do not want in practice. Forming this matrix matrix could, for instance, ruin the sparsity pattern of the

[4] TODO: add a reference to some Krylov methods

original matrix $H$. We would prefer to formulate an method where iterates are manipulated implicitly by using their expressions w.r.t. the original variables.

Insteas of the matrix $R$, we shall use the matrix $M = R^T R$ that naturally appears when applying CG to system (3.5). In general, this is the matrix we refer to as the preconditioner, by implicitly imposing it has the suitable format and properties. The Preconditioned Conjugate Gradient (PCG) method is outlined in algorithm 2.

**Algorithm 2** The preconditioned conjugate gradient method

**Require:** Starting point $x_0$
  $r_0 \leftarrow Hx_0 - c$, $v_0 \leftarrow M^{-1}$ $p_0 \leftarrow -v_0$
  **for** $k = 0, \dots, n-2$ **or** until convergence **do**
    $\alpha_k \leftarrow \frac{\langle r_k, v_k \rangle}{\|p_k\|_H^2}$
    $x_{k+1} \leftarrow x_k + \alpha_k p_k$
    $r_{k+1} \leftarrow r_k + \alpha_k H p_k$
    $v_{k+1} \leftarrow M^{-1} r_{k+1}$
    $\beta_k = \frac{\langle r_{k+1}, v_{k+1} \rangle}{\langle r_k, v_k \rangle}$
    $p_{k+1} \leftarrow -v_{k+1} + \beta_k p_k$
  **end for return** $x_k$

Compared to the vanilla CG, the PCG method comes with the extra cost of solving, at each iteration, a linear system of the form $Mv_k = r_k$. The "ease" with which this system can be solved should therefore be taken into account when choosing a preconditioner. The art of finding a good preconditioner lies in this compromise between the gain in speed of convergence and the extra cost of solving a linear system.

Since matrices $M^{-1}H$ and $R^{-T}HR^{-1}$ have the same spectrum, the efficiency of the method depends on how $M$ clusters the eigenvalues of $H$, as it is shown by the following convergence results.

**Corollary 19.** *The error $\epsilon_k$ of the iterates generated by algorithm 2 satisfy the inequality*

$$\frac{\|\epsilon_k\|_H}{\|\epsilon_0\|_H} \leq 2\left(\frac{\sqrt{\kappa(M^{-1}H)}-1}{\sqrt{\kappa(M^{-1}H)}+1}\right). \tag{3.6}$$

*Furthermore, if $M^{-1}$ has $r$ distinct eigenvalues, then algorithm 2 applied to problem* (3.1) *converges in at most $r$ iterations.*

Multiple choices of preconditioners with good practical performance can be done, although there is no preconditioner that performs well on any problem. As always in optimization, it is important to exploit the structure of a problem to make more efficient algorithms, and preconditioners make no exception. However, the most popular and versatile choice seems to be the Incomplete Cholesky Factorization [5]. Remind that the Cholesky factorization consists in forming $H = L^T L$ for $H \succ 0$, with $L$ lower triangular. WHen $H \succeq 0$, or even indefinite, the incomplete version consists into the decomposition

$$A = L^T L + R$$

where $L$ is lower triangular and $R$ is a matrix where the coefficients of $H$ that mess up the computation of the pivots of the Cholesky factorization are thrown into, broadly speaking.

In [8], proposed an algorithm for a sparse version of this factorization. They use it in the TRON solver [7] and it gives stellar performances.

We will likely use it in our algorithms so stay tuned.

### 3.3 The projected conjugate gradient

We continue our exploration of the CG method and now tackle the linearly constrained case:

$$\begin{aligned} \min_x \quad & \frac{1}{2} \langle x, Hx \rangle - \langle c, x \rangle \\ \text{s.t.} \quad & Ax = b. \end{aligned} \tag{3.7}$$

## 4 Generalities on least squares

Rewriting the objective function of problem (1.1) as $f : x \mapsto \frac{1}{2}\|r(x)\|^2$, one has:

$$\nabla f(x) = J(x)^T r(x) \tag{4.1a}$$

$$\nabla^2 f(x) = J(x)^T J(x) + S(x), \tag{4.1b}$$

where $J(x) = \left[\frac{\partial r_i}{\partial x_j}\right]_{(i,j)}$ is the Jacobian matrix of the residuals and the second component of the Hessian $S(x) = \sum_{i=1}^{d} r_i(x)\nabla^2 r_i(x)$. The latter is expensive in both computational time and storage, since it requires $d$ computations of $n \times n$ matrices. Hence, this component of the Hessian is the one to be approximated.

The Jacobian matrix of constraints function $h$ is noted $A$.

We now address is a quick review of the three most popular classes of approximations. For a comprehensive review of these methods, we refer the reader to the chapter 10 of [3].

### 4.1 Gauss-Newton method

Originally used by Gauss the prince himself, **Gauss–Newton** (GN) approximation sets $S(x)$ to the zero matrix. It is the cheapest to compute, since the Jacobian is necessary to evaluate the gradient and works well in practice for zero residuals problems [3]. When solving problem (1.1) using an iterative method, this approximation amounts to linearizing the residuals function within the norm. Indeed, approximating $\nabla f^2(x)$ by $J(x)^T J(x)$ in a quadratic model $\mathcal{Q}$ of $f$ around $x$ gives

$$\mathcal{Q}^{GN}(p) = \frac{1}{2} p^T \nabla f^2(x) p + \nabla f(x)^T p = \frac{1}{2} \left\| J(x)p + r(x) \right\|^2, \tag{4.2}$$

which corresponds to injecting the linearization $r(x + p) \approx J(x)p + r(x)$ in the squared norm.

### 4.2 Levenberg-Marquardt

Next, we describe the **Levengerg–Marquardt** (LM) method, respectively named after the first author to publish it [6] and the author of its best-known rediscovery [9]. As noted by

Marquardt in his paper, the two authors started from a different line of reasoning but came to the same conclusion. In this method, matrix $S(x)$ is set to a multiple of the identity matrix $\sigma I$ where $\sigma$ is a positive scalar. The latter is called regularization parameter, because setting $\nabla f^2(x)$ to $J(x)^T J(x)$ leads to the quadratic model

$$\mathcal{Q}^{LM}(p) = \frac{1}{2}\|J(x)p + r(x)\|^2 + \frac{\sigma}{2}\|p\|^2. \tag{4.3}$$

In other words, one can see very schematically that

$$\text{LM model} = \text{GN model} + \text{regularization term}.$$

In practice, the LM approximation works well on zero and small residuals problems and tends to be more robust than the GN approximation. It is still cheap to compute but only requires updating the regularization parameter throughout the iterative process. This method is often referred as the early stage of the trust region methods [2]. For the link with regularization methods, consider the trust region problem

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & \|x\| \leq \Delta. \end{aligned}$$

By applying KKT conditions and assuming that there is a minimizer $x^*$ lying on the trust region, i.e. $\|x^*\| = \Delta$, then (under strict complementarity), there is a strictly positive scalar (the multiplier) $\lambda$ such that

$$\nabla f(x^*) + \frac{2\lambda}{\Delta} x^* = 0.$$

As a consequence, $x^*$ is a critical point oh the unconstrained regularized problem

$$\min_x \quad f(x) + \frac{\delta}{2}\|x\|^2,$$

where $\delta = \frac{2\lambda}{\Delta}$.
Some references for LM methods: [1]

## 4.3 Quasi-Newton

Finally, one can compute an approximation of $\nabla^2 f(x)$ in a similar pattern as in **quasi-Newton** methods [10, Chapter 6] but targeted on the second order components. It has a higher computational cost than the previous two but is more accurate on large residuals problems. In [4], the authors exploit this approach in a adaptative scheme, where an estimation of the curvature is used to decide whether or not the quasi-Newton approximation is worth to use compared to the Gauss-Newton one.

It is important to bear in mind that choosing between a "cheap" approximation and a quasi-Newton type one implies making compromises. Depending on the initialization, the quasi-Newton approximation will take some iterations to be good and the accuracy will not be there when most needed, i.e. at the starting point potentially far from the solution, and will match the Gauss-Newton close to the solution on small residuals problems. In other words, the quasi-Newton is not accurate enough when most needed and very accurate when a way cheaper alternative does the same job.

References for quasi-Newton specialized to least-squares: [4, 12].

# 5 Augmented Lagrangian reformulation

In this section, we introduce the framework of Augmented Lagrangian-based algorithms and describe an application in the least-squares setting of problem (1.1).

## 5.1 Generalities

In order to remain general, we temporarily consider the mathematical program (2.1) and shall comeback to our beloved least-squares shortly after. We introduce the Augmented Lagrangian (AL) function associated to program (2.1):

$$\Phi_A(x, \lambda, \mu) := f(x) + \langle \lambda, h(x) \rangle + \frac{\mu}{2} \|h(x)\|^2, \tag{5.1}$$

where $\lambda \in \mathbb{R}^m$ is the vector of Lagrange multipliers and $\mu > 0$ is the penalty parameter.

Function (5.1) is nothing than the Lagrangian (2.2) with a quadratic penalty term, hence the adjective *Augmented*.Assume we fixed $\lambda$ and $\mu$, the problem of interest is now the bound constrained program

$$\begin{aligned} \min_x \quad & \Phi_A(x, \lambda, \mu) \\ \text{s.t.} \quad & l \leq x \leq u. \end{aligned} \tag{5.2}$$

AL-based algorithms fall into the class of penalty methods that generally enable one to use iterative methods for unconstrained optimization while steel achieving feasibility. In our case, moving the nonlinear constraints into the objective simplifies the set of constraints since only bounds constraints are left.

One of the pros of AL methods is that they come naturally with an update formula for the multipliers. Let $(x_k, \lambda_k)$ be the current primal-dual iterate, then one can choose $\lambda_{k+1}$ as

$$\lambda_{k+1} = \lambda_k + \mu_k h(x_k). \tag{5.3}$$

This relation is merely derived from $\nabla_x \Phi_A = 0$ by identifying the right hand side of equation (5.3) as multipliers satisfying the KKT condition $\nabla_x \ell = 0$.

The procedure of an AL method is relatively and relies on solving successively problem (5.2) with respect to the primal variable until a first order critical point of the original problem (2.1) is found. At every iteration, the penalty parameter is increased and the multipliers are updated by formula (5.3). This general pattern is outlined in algorithm 3 and the main steps of the outer iteration are given in algorithm 4.

**TODO How to compute the approximate minimizer? Projected conjugate gradient!**

## 5.2 Application to structured least-squares

We now consider program (1.1), for which the AL function is given by

$$\Phi_A(x, \lambda, \mu) := \frac{1}{2} \|r(x)\|^2 + \langle \lambda, h(x) \rangle + \frac{\mu}{2} \|h(x)\|^2, \tag{5.4}$$

Contrary to formulation (5.1), we keep the linear equality constraints as is and penalize the violation of the nonlinear constraints. Although the computation of the projection onto the

**Algorithm 3** Basic AL algorithm for solving (2.1)

**Require:** Starting point $(x_0^s, \lambda_0)$, parameter $\mu_0$ parameter $\mu_0$ and tolerances $\omega_*, \omega_0, \eta_*, \eta_0$.
**repeat**
Compute an approximate solution $x_k$ of (5.2) starting from $x_k^s$ with tolerance $\omega_k$.
**if** $\|h(x_k)\| < \eta_k$ **then**
Update iterate and increase penalty parameter.
**else**
Restart minimization of (5.2) with a higher penalty parameter.
**end if**
Update tolerances
**until** $\|h(x_k)\| < \eta_*$ **and** $\|\nabla_x \ell(x_k, \lambda_k)\| < \omega_*$
Return current approximate solution.

**Algorithm 4** Outer iteration of basic AL algorithm with trust region

**Step 1: Inner Iteration** Starting from $x_0^s$, approximately solve $\min_x \Phi_A(x, y_k, \mu_k)$ by computing $x_k$ such that $\|x_k - P(x_k - \nabla_k \Phi_A)\| \leq \omega_k$ by a trust region process.

**If** $\|h(x_k)\| \leq \eta_k$, execute **Step 2**.

Otherwise, excecute **Step 3**.

**Step 2: Iterate Update** Update $y_{k+1}$ by formula (5.3) and set $x_{k+1}^s \leftarrow x_k$.

Choose new tolreances $\omega_{k+1}, \eta_{k+1}$ and new penalty parameter $\mu_{k+1}$.

Increment $k$ and go back to **Step 1**.

**Step 3: Adjustment of the Penalty Parameter** Choose $\mu_{k+1}$ significantly greater than $\mu_k$.

Leave the iterate unchanged: $\left(x_{k+1}^s, \lambda_{k+1}\right) \leftarrow (x_k^s, \lambda_k)$.

Go back to **Step 1**.

set $\mathcal{X}$ shall differ, the framework remains the same, One has the following expression of the gradient:

$$\nabla_x \Phi_A(x, \lambda, \mu) = J(x)^T r(x) + A^T \pi(x, \lambda, \mu), \tag{5.5}$$

with $\pi(x, \lambda, \mu) := \lambda + \mu h(x)$ is the first-order estimates of the Lagrange multipliers.

The Hessian is given by

$$\nabla^2_{xx} \Phi_A(x, \lambda, \mu) = J(x)^T J(x) + \mu A(x)^T A(x) + S(x) + \sum_{i=1}^{d} \nabla^2 h_i(x) \pi(x, \lambda, \mu). \tag{5.6}$$

For fixed $\lambda$ and $\mu$, reformulating problem (1.1) with function (5.1) gives the linearly constrained problem

$$\begin{aligned} \min_x \quad & \Phi_A(x, \lambda, \mu) \\ \text{s.t.} \quad & x \in \mathcal{X} \end{aligned} \tag{5.7}$$

As for any other AL based algorithm, the idea behind MCWAL is to solve by an iterative method problem (5.7) until a first order critical point of problem (1.1) is found. At every iteration, a the new iterate will be computed after (approximately) solving a trust region subproblem formed after a quadratic model of the AL around the current iterate.

## 5.3 Subproblem

Given a primal-dual iterate $(x_k, \lambda_k)$ and a penalty parameter $\mu_k$, we consider a quadratic model of the AL around $x_k$:

$$\mathcal{Q}_k(p) = \frac{1}{2} \langle p, H_k p \rangle + \langle g_k, p \rangle , \tag{5.8}$$

where $H_k := \nabla^2_{xx} \Phi_A(x_k, \lambda_k, \mu_k)$ or an approximation of it and $g_k := \nabla_x \Phi_A(x_k, \lambda_k, \mu_k)$.

Vector $p$ denotes the unknown of the subproblem whose (approximate) solution $p_k$ shall be used to compute the new iterate $x_{k+1} = x_k + p_k$.

For the linear constraints, we want $x_k + p \in \mathcal{X}$ which will be provided if:

- $Cp = 0$ (provided that $Cx_0 = b$)
- $x_k - l \leq p \leq u - x_k$

The above conditions shall be written $p \in \mathcal{X}_k$ where $\mathcal{X}_k := \{p \mid Cp = 0,\ x_k - l \leq p \leq u - x_k\}$.

As part of our method, we will also add a trust region constraint of the form $\|p\|_k \leq \Delta_k$ for a radius $\Delta_k$ and a norm $\|\cdot\|_k$. Index $k$ in the latter means that the norm might depend on the iteration. A priori, we would use the euclidean norm.

The subproblem of an outer iteration is then given by

$$\begin{aligned} \min_{p \in \mathcal{X}_k} \quad & \mathcal{Q}_k(p) \\ \text{s.t.} \quad & \|p\|_k \leq \Delta_k. \end{aligned} \tag{5.9}$$

A first sketch of the MCWAL procedure is drawn in algorithm 5.

**Algorithm 5** Sketch of MCWAL

**Require:** $x_0 \in \mathcal{X}$, $\lambda_0$, $\mu_0, \tau_0$ and constants $\eta_s$

**while not optimal**[5] **do**

Evaluate $H_k$ and $g_k$

Compute a solution $p_k$ of subproblem (5.9)

Compute ratio ${\rho_k}^6$

**if** $\rho_k \geq \eta_s$ **then** ▷ Good step

$x_{k+1} \leftarrow x_k + p_k$

Choose $\Delta_{k+1} > \Delta_k$

**if** $\|h(x_k)\| \leq \tau_k$ **then**

$y_{k+1} \leftarrow \pi(x_k, y_k, \mu_k)$

Choose $\mu_{k+1} > \mu_k$ and $\tau_{k+1} < \tau_k$

**else**

$y_{k+1} \leftarrow y_k$

Choose $\mu_{k+1} < \mu_k$

**end if**

**else** ▷ Bad step

$x_{k+1} \leftarrow x_k$

$y_{k+1} \leftarrow y_k$

$\mu_{k+1} \leftarrow \mu_k$

Choose $\Delta_{k+1} < \Delta_k$

**end if**

**end while**

# References


[1] S. Bellavia, S. Gratton, and E. Riccietti. A Levenberg-Marquardt method for large nonlinear least-squares problems with dynamic accuracy in functions and gradients. *Numerische Mathematik*, 140:791–825, 2018. doi: 10.1007/s00211-018-0977-z.

[2] A.R. Conn, N.I.M Gould, and Ph.L. Toint. *Trust Region Methods*. SIAM: Society of Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000. doi: 10.1137/1.9780898719857.

[3] J.E. Dennis Jr and R.B. Schnabel. *Numerical Methods for Unconstrained optimization and Nonlinear Equations*. Classics in Applied Mathematics. SIAM, Philadelphia, PA, USA, 1996. doi: 10.1137/1.9781611971200.

[4] J.E. Dennis Jr, D.M. Gay, and R.E. Walsh. An adaptive nonlinear least-squares algorithm. *ACM Transactions on Mathematical Software*, 7(3):348–368, 1981. doi: 10.1145/355958.355965.

[5] G.H. Golub and C.F. van Loan. *Matrix Computations*. JHU Press, Maryland, MD, USA, 4th edition, 2013. doi: 10.56021/9781421407944.

[6] K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*, 2:164–168, 1944.

[7] C.-J. Lin and J.J Moré. Newton's method for large bound-constrained optimization problems. *SIAM Journal on Optimization*, 9(4):1100–1127, 1999. doi: 10.1137/S1052623498345075.

[8] C.-J. Lin and J.J Moré. Incomplete Cholesky factorizations with limited memory. *SIAM Journal on Scientific Computing*, 21(1):24–45, 1999. doi: 10.1137/S1064827597327334.

[9] D.W. Marquardt. An algorithm for least squares estimation of non-linear parameters. *SIAM Journal*, 11:431–441, 1963.

[10] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer series in Operation Reasearch and Financial Engineering. Springer, New York, NY, USA, seconde edition, 2006. doi: 10.1007/978-0-387-40065-5.

[11] R.T. Rockafellar and R.J.B. Wets. *Variational Analysis*. Springer Berlin, Heidelberg, 1st edition, 1998. doi: 10.1007/978-3-642-02431-3.

[12] H. Yabe and T. Takahashi. Factorized quasi-Newton methods for nonlinear least squares problems. *Mathematical Programming*, 51:75–100, 1991.