# Computational Matehmatics for Learning and Data Analysis

2019 / 2020

**Poggiali Alessandro, Berti Stefano**


## Abstract

**(P)** is the linear least squares problem

$$\min_{x\in\mathbb{R}^n} \|\hat{A}x - b\|$$

where $\hat{A}$ is the matrix obtained by augmenting the (tall thin) matrix $A$ from the ML-cup dataset by prof. Micheli with a few functions of your choice of the features of the dataset, and $b$ is one of the corresponding output vectors. For instance, if $A$ contains columns $[a1, a2]$, you may add functions such as $log(a1), a1^2, a1 * a2, \ldots$

**(A1)** is an algorithm of che class of Conjugate Gradient methods [references: J. Nocedal, S. Wright, Numerical Optimization].

**(A2)** is thin QR factorization with Householder reflectors [Trefethen, Bau, Numerical Linear Algebra, Lecture 10], in the variant where one does not form the matrix $Q$, but stores the Householder vectors $u_k$ and uses them to perform (implicitly) products with $Q$ and $Q^T$.

No off-the-shelf solvers allowed. In particular you must implement yourself the thin QR factorization, and the computational cost of your implementation should scale linearly with the largest dimension of the matrix $A$.


# 1 Setting the stage

## 1.1 Linear Square Problem

Our problem is: how close can we get to

$$\min_{x\in\mathbb{R}^n} \|Ax - b\|$$

where

- $A$ is a *tall thin matrix* (so it is a matrix of the form $A \in M(m, n, R)$ where $m \gg n$)
- $b$ is a vector of real number
- $\|.\|$ is the *2-norm* or *Euclidean Norm*: $\|x\| := \sqrt{\sum_{i=1}^{n} x_i}$

and so which is the closest vector to $b$ inside the hyperplane $Im(A)$.

## 1.2 QR factorization with Householder Reflectors

Every matrix $A \in \mathbb{R}^{m\times n}$ can be written as $A = QR$, where $Q \in \mathbb{R}^{m\times m}$ is orthogonal and $R \in \mathbb{R}^{m\times n}$ is upper traingular (i.e. $i > j \Rightarrow R_{ij} = 0$). To obtain this form of factorization, we use the *Householder reflectors*. *Housheoulder reflectors* (written as $H$) have the form $H = I - \frac{2}{u^T u}uu^T$. $H$ can also be written as $I - 2vv^T$ where $v = \frac{u}{\|u\|}$. $H$ is a matrix that if multiplied with a vector perform a symmetry w.r.t. the plane orthogonal to $u$. By setting this $u = x - y$, we are projecting $x$ in $y$, and if $y$ has the form $e_1\|x\|$ (because $H$ is orthogonal and must preserve the norm), we are projecting $x$ in the first coordinate. But if $\|x\| \sim x_1$, this could lead to **cancellation problem**, so it is a good pratice to choose the norm of $x$ accordingly with the sign of the first entry, so $+\|x\|$ if $x_1$ is negative and $-\|x\|$ if $x_1$ is positive, as shown in Algorithm 1. We also have a way to perform **fast product Housheoulder-vector** by re-arranging parenthesis: we can rewrite $Hx = (I - 2vv^T)x = x - 2v(v^T x)$ to reduce the computational complexity from $O(m^2)$ to $O(m)$.

Algorithm 2 give us the QR factorization of the matrix $A$ using the *Househoulder reflectors*. In this Algorithm we use some techniques that allow us to have a more efficient implementation than the trivial one. In particular, we exploit **fast product Householder-vector** to upgrade matrix $A$ in order to avoid matrix-matrix multiplications. Moreover, we put manually the zeros in the right place and if the matrix is square we can stop the algorithm one step before the full size, because last iteration can only change the sign of last entry. Finally, instead of perform a lot of matrix-vector multiplication we accumulate this vector multipling them toghether. In this way we have the same number of operations, but on a computer this way is more efficient because computer manages well block operations.

**Algorithm 1** Householder Reflectors

```
1: function [v, s] ← householder_vector(x)
2:     s ← -sign(x[0]) * norm(x)
3:     v ← x
4:     v[0] ← v[0] − s
5:     v ← v/norm(v)
```

**How can we use QR to solve LS?**

If $A$ is a *tall thin matrix* ($m \gg n$), we write the matrix $A$ as $QR$, where $Q = [Q_1, Q_2]$ and $R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$ and since orthogonal matrices preserve norms, we can do $\|Ax - b\| = \|Q^T(Ax - b)\| = \|Q^T QRx - Q^T b\| = \|Rx - Q^T b\| = \left\| \begin{bmatrix} R_1 \\ 0 \end{bmatrix} x - \begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix} b \right\| = \left\| \begin{bmatrix} R_1 x - Q_1^T b \\ Q_2^T b \end{bmatrix} \right\|$ so our problem can be summarized as $\|Ax-b\| = \left\| \begin{bmatrix} R_1 x - Q_1^T b \\ Q_2^T b \end{bmatrix} \right\|$ and now I can chose $x$ s.t. $R_1 x - Q_1^T b = 0$ (just take

**Algorithm 2** QR factorization with Householder Reflectors

**function** $[Q, R] \leftarrow myqr(A)$
*top*:
    *[m, n]* $\leftarrow$ *size(A)*
    $V \leftarrow$ *emptyList*
    $Q \leftarrow I_{m \times m}$
*loop*:
    *for j = 1:min(m-1, n)*:
        *[v, s]* $\leftarrow$ *householder-vector(A(j:end, j))*
        *A(j, j)* $\leftarrow$ *s*
        *A(j+1:end, j)* $\leftarrow$ *0*
        *A(j:end, j:end)* $\leftarrow$ *A(J.end, j:end) - 2*v*(v'*A(j:end, j+1:end))*
        $V \leftarrow$ *V.insert(v)*
*end*:
    $R = A$
    *for i = 1:min(m-1, n)*
        *Q(i:end, i:end) = Q(i:end, i:end)*[*$I_{min(m-i,n)-i}$ *- 2*$v_i * v_i^T$*]*

$x = R_1^{-1} Q_1^T b$ if $R_1$ is invertible), and in this way we have $\min_{x \in \mathbb{R}^n} \|Ax - b\| = \|Q_2^T b\|$ and $x = \text{argmin}_{x \in \mathbb{R}^n} \|Ax - b\| = x = R_1^{-1} Q_1^T b$.

How many solutions do we have? We know that $\min_{x \in \mathbb{R}^n} \|Ax - b\|$ has a unique solution $\iff$ $A$ has full column rank $\iff$ $A^T A \succ 0$ $\iff$ 0 is not an eigenvalue of $A^T A$ $\iff$ $A^T A$ is invertible.

We can add [$\iff$ $R_1$ is invertible] because $A^T A = (QR)^T QR = R^T Q^T QR = R^T R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}^T \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = \begin{bmatrix} R_1^T & 0 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = R_1^T R_1$ and, since $A^T A$ is a square matrix that is the product of the two square matrices $R_1^T$ and $R_1$, then $A^T A$ is invertible $\iff$ both $R_1$ and $R_1^T$ are invertible (this can be seen looking at the determinant $[det(A^T A) = det(R_1^T) det(R_1)]$). Finally, we got $A^T A = R_1^T R_1$ is invertible $\iff$ $R_1$ is invertible. We can easily check if $R_1$ is invertible because it is upper triangular, so we just look at its elements on the diagonal: if they are all $\neq 0$, it is invertible.

## 1.3 Conjugate Gradient Method

If $A^T A$ is positive definite, then we can find a solution to $Ax = b$ by minimizing the (strictly convex) function $f(x) = \frac{1}{2} x^T Ax - b^T x$ (that is equal to set the gradient = 0). Doing *Conjugate gradient* on this problem, can be interpreted as *Krylov subspace method*. We remember that

- **A-orthogonal**: $x, y \in R^m$ are A-orthogonal if $x^T Ay = 0$
- **A-norm**: the A-norm of $x \in R^m$ is $\|x\|_A := (x^t Ax)^{\frac{1}{2}}$. Note that if A is *positive definite*, then the A-norm is $\geq 0$ (it is 0 only for the 0 vector)

For conjugate gradient, we use three ingredients: the current interate $x_k$, the residual $r_k = b - Ax_k = -\Delta f(x_k)$ and the search direction $d_k$.

**Algorithm 3** Conjugate Gradient

```
1:  function x* ← mycg(A, b)
2:  top:
3:      x_0 ← 0
4:      r_0 ← d_0 ← b
5:  loop:
6:      for k = 1:n
7:          α_k = (r_{k-1}^T r_{k-1})/(d_{k-1}^T A d_{k-1})
8:          x_k = x_{k-1} + α_k A d_{k-1}
9:          β_k = (r_k^T r_k)/(r_{k-1}^T r_{k-1})
10:         d_k = r_k + β_k d_{k-1}
11: end:
12:     x_* = x_k
```

Initially, to keep things simple, we choose as our first direction the prior $b$, because we start with $x_0 = 0$. At each iteration, we need to chose $\alpha_k$, that is our step length, in such a way that it minimizes the residual along the search direction. Then we can update our $x_k$ and, consequently, the residual $r_k$. To update the residual, we use the fact that $b - Ax_k = [x_k = x_{k-1} + \alpha_k d_{k-1}] = b - A(x_{k-1} + \alpha_k d_{k-1}) = [b - Ax_{k-1} = r_{k-1}] = r_{k-1} - \alpha_k A d_{k-1}$. The trick about conjugate gradient is that at each step we generate a new search direction, which is not exactly the residual, but is the residual modified to be A-orthogonal to the previous search direction. In this way, $\beta_k$ is a scalar that is chosen s.t. $d_k$ is A-orthogonal to $d_{k-1}$ ($d_{k-1} A d_k = 0$).

# 2 What to expect from the algorithms

## 2.1 QR with HouseHolder

We know that this algorithm is bandwith heavy and not parallelizable, as every reflection that produces a new zero element changes the entirety of both Q and R matrices. We are going to use a variant of the QR factorization, in which we do not form the matrix Q, but we store the Householder vectors $u_k$ and we use them to perform the implicit product with $Q$ and $Q^T$, in this way we need just to store those vectors which size is $O(mn)$ instead of $O(m^2)$ and if $m \gg n$, this has a big impact: this could lead to have a cost for thin QR equal to $O(mn^2)$, so it scales lineary with the largest dimension of the matrix and quadratically with the other one. The computational cost for thin QR factorization is $2mn^2 - \frac{2}{3}n^3 + O(mn)$ that represents the two cases: if $m = n$, we have that the cost is $\frac{4}{3}n^3$, if $m \gg n$ the cost scales like $2mn^2$. LU/Gaussian elimination has computational cost of $\frac{2}{3}n^3$.

### 2.2 Conjugate Gradient

We don't need to remember the history of each step, we just need to work on three variables $x, r$ and $d$. The cost is $n$ matrix-vector products (for $Ad_{k-1}$, that appears two times, but we can compute it and store it) plus $O(nm)$, which is the same cost for Krylov method for asymmetric matrices. In fact, this algorithm is Krylov type algorithm. A **Krylov subspace** of dimension $k$ $K_k(A, b)$ is the span made of our iterations, and this is s.t. $K_k(A, b) = span(x_1, x_2, \ldots, x_k) = span(d_0, d_1, \ldots, d_{k-1}) = span(r_0, r_1, \ldots, r_{k-1})$ where $span(v_1, \ldots, v_n) := \{\alpha_1 v_1 + \ldots + \alpha_n v_n \,|\, \alpha_1, \ldots, \alpha_n \in \mathbb{R}\}$. So we build a **orthogonal basis** (not orthonormal) $c_i, \ldots, c_k$, where necessarily $c_i^T c_j = 0 \; \forall i \neq j$. In our algorithm, we have that $r_i^T r_k = d_i^T A d_k = 0 \; \forall i > k$, and $r_k^T r_k \neq 0$ because they are not orthonormal (but if we want, we can rescale it and made them orthonormal). The $r_i$ are orthogonal, while the $d_i$ are A-orthonormal. So in CG, at each iteration, we compute $x_k$ s.t. $v^T r_K = 0 \; \forall$ vectors in $K_k(A, b) = span(r_0, r_1, \ldots, r_{k-1})$ in a way that $r_i^T r_k = 0 \; i < k \iff Q_k^T(b - Ax_k) = 0$, because columns of $Q_k$ are $q_k = \frac{r_k}{\|r_k\|}$ The convergence of the Conjugate Gradient Method depends on the maximum eigenvalue $\lambda_{max}$ and the minimum eigenvalue $\lambda_{min}$, and CG converges with rate
$\|x - x_k\| \leq \left\| \frac{\sqrt{\lambda_{max}} - \sqrt{\lambda_{min}}}{\sqrt{\lambda_{max}} + \sqrt{\lambda_{min}}} \right\|^k \leq \|x - x_0\|$

# 3 Input data

Our input data is the matrix A, which is the tall thin matrx used as input in the *ML-cup* competition. Its shape is $(1765, 23)$, we have to add few functions of our choice of the features of the dataset, and we decided to add the following functions:

- logarithm of the absolute value of the first column
- product of column 2, 3, 4
- fifth column to square