Computational Mathematics for Learning and Data Analysis

2019 / 2020

Poggiali Alessandro, Berti Stefano

1 Setting the stage

1.1 The problem: Least Square

Our problem is to find an array x such that

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

holds, where

- A is a *tall thin matrix* (so it is a matrix $A \in M(m, n, R)$ 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 to find the closest vector to b inside the hyperplane $Im(A)$.

1.2 First algorithm: Conjugate Gradient Method

If a square matrix $A = A^T$ 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 $\nabla f(x) = Ax - b = 0$). However our matrix A is a *tall thin matrix*

and since in this case finding a solution to $Ax = b$ is equivalent to solving the normal equations

$$A^T Ax - A^T b = 0$$

then we can use *Conjugate gradient* method to minimize the function

$$f(x) = \frac{1}{2} x^T A^T Ax - x^T A^T b$$

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 ≥ 0 (it is 0 only for the 0 vector)

Let x_k be the parameter vector after the k-th iteration, then the residual vector is $r_k = b - Ax_k$ and the negative gradient $g_k = -\nabla f(x_k) = A^T b - A^T Ax = A^T r_k$. The step size α_k can be computed in a similar way as in the standard CG method, but here the denominator is $d_k^T A^T Ad_k = \|Ad_k\|^2$ and so it can be obtained without calculating $A^T A$.

To update the residual, we use the fact that $r_k = b - Ax_k = [x_k = x_{k-1} + \alpha_k d_{k-1}] = b - A(x_{k-1} + \alpha_k d_{k-1}) = [b - Ax_{k-1} = r_{k-1}] = r_{k-1} - \alpha_k Ad_{k-1}$. 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 $A^T A$-orthogonal to the previous search direction. In this way, β_k is a scalar that is chosen s.t. d_k is $A^T A$-orthogonal to d_{k-1}.

1.3 Second algorithm: QR factorization with Householder Reflectors

To solve the Least Square Problem, we use the *thin QR factorization* with all the optimization seen (fast Householder-vector product, cancellation problem resolution, manually changing known entries) in order to factorize A as $Q_1 R_1$. We use a variant of the *thin QR factorization* where we do not form the matrix Q, but we keep the Householder vector u_k to perform implicit product with Q and Q^T. With this factorization, we can write $\|Ax - b\|$ as $\left\| \begin{bmatrix} R_1 x - Q_1^T b \\ Q_2^T b \end{bmatrix} \right\|$ and now we can chose x such that $R_1 x - Q_1^T b = 0$, which is $x = R_1^{-1} Q_1^T b$ (if R_1 is invertible). Finally we should have $x = \text{argmin}_{x \in \mathbb{R}^n} \|Ax - b\| = R_1^{-1} Q_1^T b$ and $\|Ax - b\| = \|Q_2^T b\|$.

2 What to expect from the algorithms

2.1 Conjugate Gradient

Our system $Ax = b$ is equivalent to the system $Bx = c$, where $B = A^T A$ is symmetric and positive definite and $c = A^T b$. However, from a numerical point of view, the two systems are different, because of rounding-off errors that occur in joining the product $A^T A$. But since our algorithm does not involve the computation of $B = A^T A$, then we have the same convergence result as in the standard CG method. In particular we know that this method gives the solution in m steps if no rounding-off error occurs, where m is the size of B, namely the smallest size of our original rectangular matrix A. In realistic cases, the absence of rounding-off errors cannot be assured and so we need more than m step or even to restart the algorithm many times. Also the basic relations concerning convergence ($< r_i, r_k >= 0, < Ad_i, d_k >= 0$ for $i \neq k$) will not be satisfied

exactly and therefore x_m will not be as good an estimate the exact solution as desired. From [1] follows that the larger the ratios α_i/α_{i-1}, the more rapidily the rounding-off errors accumulate. Moreover, since α_i lie on the range

$$1/\lambda_{max} < \alpha_i < 1/\lambda_{min}$$

the ratio $\rho = \lambda_{max}/\lambda_{min}$ gives us an upper bound of the critical ratio α_i/α_{i-1} which determines the stability of the entire process. So we would like to have ρ near one, that means our matrix is near multiple of the identity and then the CG method is relatively stable.

The algorithm performs at every step two multiplications (one is enough if we compute it and store it) of A with the m-vector d_i ($O(m^2n)$) and one multiplication of A^T with the n-vector r_i ($O(n^2m)$). So, the total cost is $m(O(m^2n)+O((n^2m))+O(m^2))$, where $O(m^2)$ comes from the scalars product between m-vectors to compute the norms for every steps.

The convergence of the Conjugate Gradient Method depends on the maximum eigenvalue λ_{max} and the minimum eigenvalue λ_{min}, and CG converges with rate
$\|x - x_k\| \leq \left\| \frac{\sqrt{\lambda_{max}}-\sqrt{\lambda_{min}}}{\sqrt{\lambda_{max}}+\sqrt{\lambda_{min}}} \right\|^k \leq \|x - x_0\|$

2.2 QR with HouseHolder

The QR factorization has a cancellation problem during the Householder reflector construction that is easily fixed by summing first entry and norm instead of subtract it.

Apart from that, since every step is *backward stable*, the factorization algorithm is *backward stable*: the computed Q, R are the exact result of $qr(A+\Delta A)$ where $\|\Delta A\| \leq O(u)\,\|A\|$, so we only have intrinsic representation errors.

The computational cost for thin QR factorization is $2mn^2 - \frac{2}{3}n^3 + O(mn)$ flops,

which represents two cases: if $m \approx n$, we have that the cost is $\frac{4}{3}n^3$, if $m \gg n$ the cost scales like $2mn^2$, so it scales linearly with respect to the biggest dimension of A .

Before we said that $x = R_1^{-1}Q_1^T b$, but to be able to calculate it we need R_1 to be invertible.

We know that A has full column rank $\iff$ $Az \neq 0 \forall z \neq 0$ $\iff$ $z^T A^T Az = \|Az\|^2 \forall z \neq 0$ $\iff$ $A^T A$ is positive definite $\iff$ all eigenvalues of $A^T A$ are > 0 $\iff$ 0 is not an eigenvalue of $A^T A$ $\iff$ $A^T A = (Q_1 R_1)^T Q_1 R_1 = R_1^T Q_1^T Q_1 R_1 = R_1^T R_1$ is invertible $\iff$ $det(A^T A) = det(R_1^T) det(R_1) \neq 0$ $\iff$ $det(R_1) \neq 0$ $\iff$ R_1 is invertible.

So if A has full column rank, $\min_{x \in \mathbb{R}^n} \|Ax - b\|$ has solution and it is unique.

R_1 is invertible also if all elements on its diagonal are $\neq 0$. The cost to solve $x = R_1^{-1}(Q_1^T)b$ is a multiplication $c = (Q_1^T)b$, which costs $O(mn)$, and the resolution of the triangular system $R_1 x = c$ with back-substitution, which costs $O(n^2)$. Anyway, the overall cost $O(mn) + O(n^2)$ is negligible with respect to the cost $O(mn^2)$ to compute Q_1, R_1. This factorization algorithm is bandwidth heavy and not parallelizable, as every reflection that produces a new zero element changes the entirety of both Q and R matrices

3 Input data

Our input data is the matrix A, which is the tall thin matrix used as input in the *ML-cup 2019–2020* competition. Its shape is $(1765, 20)$, we augmented it with few functions of the features of the dataset:

- **21$^{\text{th}}$ column**: logarithm of the absolute value of the 1$^{\text{st}}$ column
- **22$^{\text{th}}$ column**: product of 2$^{\text{nd}}$, 3$^{\text{rd}}$ and 4$^{\text{th}}$ columns
- **23$^{\text{th}}$ column**: 5$^{\text{th}}$ column to square

So the final shape of A is $(1765, 23)$. The condition number of the matrix A (calculated with *numpy.linalg.cond*, which does SVD and $\frac{\sigma_1}{\sigma_n}$) is $\approx 400000(4 \times 10^5)$, while the condition number of a random matrix with the same dimension and same range of values (min and max of the 2 matrices are the same) is ≈ 1.7. Those values doesn't change with respect to the augmented columns, because the augmented columns are not linear combination of other columns. We can conclude that the matrix A is ill conditioned, so it is almost singular and the solution of our problem could be not accurate. Anyway, we know that the solution of the least square problem exists and it is unique, because A is a full column rank matrix because $rank(A) = n = 23$, where rank is calculated as the number of singular values $> \max(\sigma_i) \times max(m, n) \times eps$, which is the same approach used in MATLAB.

References

[1] Magnus R Hestenes, Eduard Stiefel, et al. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49(6):409–436, 1952.