Conjugate gradient method

Abdelwahid Benslimane

wahid.benslimane@gmail.com

Recall of the principle of gradient descent

Gradient descent methods are approaches to solving systems of linear equations $Ax = b$ where A is a symmetric positive definite matrix. The principle of these methods is based on an unconstrained optimisation technique. The objective is then to minimize the following positive quadratic definite form:

$f : x \mapsto \frac{1}{2}\langle Ax, x\rangle - \langle b, x\rangle$

with A being a symmetric positive definite square matrix of size n x n. Note that :

$\nabla f(x) = Ax - b.$

A positive quadratic definite form has a unique stationary point, i.e. a unique global minimum. Therefore, the global minimum of $f(x)$ is reached for $\nabla f(x) = Ax - b = 0 <=> Ax = b$. Reaching the minimum of the $\frac{1}{2}x^TAx - x^Tb$ will then give the solution of $Ax = b$.

A gradient descent algorithm is an iterative algorithm that consists, starting from x_0, of approximating the unknown solution x^* corresponding to the unique minimizer of the quadratic form:

$f(x) = \frac{1}{2}x^TAx - x^Tb, x \in \mathbb{R}^n.$

The idea is that if $f(x)$ decreases after an iteration, then we are approaching x^*.

How to choose the best descent direction?

At the initial state, we have a given vector x_0 and the value $f(x_0)$ and we look for a vector x_1 such that $f(x_1) < f(x_0)$.

Let $x_1 = x_0 + p \iff p = x_1 - x_0$ (where p is the slope vector which gives the direction of descent). The aim is to find the best possible expression for p. To do this we use the first order limited expansion of f:

$f(x_0 + hp) - f(x_0) = \langle \nabla f(x_0), hp\rangle_{\mathbb{R}^n} + o(h)$ (1).

To have the largest possible negative slope $f(x_0 + hp) - f(x_0)$ in absolute value for infinitely small h we use the Cauchy-Schwarz theorem which shows, among other things, that if we have the vectors $(x, y) \in E^2$ such that $|\langle x, y\rangle| = \| x \| \| y\|$, then x and y are related, i.e. :

$x = 0$ or $\exists \lambda \in \mathbb{R}, y = \lambda x.$

We must therefore choose p such that $p = -\alpha\nabla f(x_0), \alpha > 0$. Indeed, if we replace p by $-\alpha\nabla f(x)$ in the scalar product of equation (1), we have:

$\langle \nabla f(x_0), hp\rangle_{\mathbb{R}^n} = \langle \nabla f(x_0), -h\alpha\nabla f(x_0)\rangle_{\mathbb{R}^n} = -h\alpha \parallel \nabla f(x_0) \parallel^2 < 0,\ h > 0,\ \alpha > 0.$

Thus, the point x_1 which verifies $f(x_1) < f(x_0)$ is given by : $x_1 = x_0 - \alpha_1\nabla f(x_0)$. We descend in the direction given by $-\nabla f(x_0)$ and α_1 is chosen so as to minimize the value of $f(x_1)$. And we iterate the process to obtain the sequence (x_k) defined iteratively by :

$x_{k+1} = x_k - \alpha_{k+1}\nabla f(x_k)$

Let us now note r_k the residual at the iteration k:

$r_k = b - Ax_k.$

Note that r_k is the opposite of the gradient of f for $x = x_k$:

$r_k = -(Ax_k - b) = -\nabla f(x_k).$

The gradient algorithm thus evolves in the direction r_k:

$x_{k+1} = x_k + \alpha_{k+1}r_k.$

Conjugate gradient

Definition: Let A be a symmetric positive definite matrix. Two non-null vectors u and v are said mutually A-conjugate if and only if:

$u^TAv = 0.$

We deduce the following scalar product:

$\langle u, v\rangle_A = u^TAv$

Two vectors u and v are therefore conjugate with respect to A if they are orthogonal for this scalar product, i.e. if :

$\langle u, v \rangle_A = 0.$

Conjugation is a symmetrical relationship.

a) Direct method approach

Let (p_k) be a sequence of n directions conjugated two by two. Then the p_k form a basis of $\mathbb{R}^n$, so the solution $x^* \in \mathbb{R}^n$ of $Ax = b$ in this basis is written :

$$x^* = \sum_{i=1}^{n} \alpha_i p_i$$

The procedure for finding the coefficients α is then as follows:

$$b = Ax^* = \sum_{i=1}^{n} \alpha_i A p_i$$

$$=> p_k^T b = p_k^T A x^* = \sum_{i=1}^{n} \alpha_i p_k^T A p_i = \alpha_k p_k^T A p_k \text{ (as } \forall i \neq k \; p_i, p_k \text{ are conjugate to each other)}$$

$$=> \alpha_k = \frac{p_k^T b}{p_k^T A p_k} = \frac{<p_k, b>}{<p_k, p_k>_A} = \frac{<p_k, b>}{||p_k||_A^2}.$$

To solve the system $Ax = b$, we need to find a sequence of n conjugate directions p_k and then compute the coefficients α_k.

If we follow this direct approach, it is necessary to calculate exactly n conjugate directions p_k. But if chosen wisely, it is not necessary to determine all of them to obtain a good approximation of the solution x^*. In this case we have to opt for an iterative approach. This choice thus allows us to consider solving very large systems, where the computation of the set of directions p_k would have been very time consuming.

b) Itérative method approach

The idea of the iterative conjugate gradient method is to construct a sequence (p_k) of descent directions where all the vectors p_k are conjugated two by two but where we also want the next direction p_{k+1} to be built from the current residual and the previously constructed directions. This is the particularity of the conjugate gradient method, which makes it very efficient and avoids the convergence of the sequence (x_k) in "zigzag".

Calculation of the descent directions p_k:

Constraining two vectors to be conjugate amounts to an orthonormality constraint, and for this we use a procedure based on that of Gram-Schmidt. Thus, we define the directions p as follows:

$$p_{k+1} = \underbrace{r_k - \frac{<r_k, p_k>_A}{<p_k, p_k>_A} p_k}_{\text{Gram-Schmidt orthogonalization process}} = r_k - \frac{p_k^T A r_k}{p_k^T A p_k} p_k$$

Along this direction, the vector x_{k+1} is given by :

$$x_{k+1} = x_k + \alpha_{k+1} p_{k+1}$$

Computation of the descent coefficients α_k**:**

As mentioned in my explanation of the gradient method with variable optimal step size, the step α_{k+1} is determined so as to minimize the quantity :

$f(x_{k+1}) = f(x_k + \alpha_{k+1} p_{k+1}) = f(\alpha_{k+1}) = f(\alpha)$ with $\alpha = \alpha_{k+1}$.

As:

$$f(x_{k+1}) = \frac{1}{2}(x_k + \alpha_{k+1} p_{k+1})^T A (x_k + \alpha_{k+1} p_{k+1}) - (x_k + \alpha_{k+1} p_{k+1})^T b$$

then:

$f(\alpha_{k+1}) = \frac{1}{2}\alpha_{k+1} p_{k+1}^T A \alpha_{k+1} p_{k+1} + \alpha_{k+1} p_{k+1}^T A x_k - \alpha_{k+1} p_{k+1}^T b$ + a constant independent from α_{k+1}

$<=>$

$f(\alpha_{k+1}) = \frac{1}{2}\alpha_{k+1}^2 p_{k+1}^T A p_{k+1} + \alpha_{k+1} p_{k+1}^T (A x_k - b)$ + a constant independent from α_{k+1}.

The minimum of f is reached for:

$$\frac{df}{d\alpha}(\alpha_{k+1}) = 0$$

$<=>$

$$\alpha_{k+1} p_{k+1}^T A p_{k+1} + p_{k+1}^T (A x_k - b) = 0.$$

As the matrix A is positive definite, $p_{k+1}^T A p_{k+1} > 0$, therefore we can write:

$$\alpha_{k+1} = \frac{p_{k+1}^T (b - A x_k)}{p_{k+1}^T A p_{k+1}} = \frac{p_{k+1}^T r_k}{p_{k+1}^T A p_{k+1}}.$$

Below, my (basic) implementation of the conjugate gradient method within the function conjugateGradient that takes as arguments the symmetric definite matrix A and the vector b of a linear system of equations $Ax = b$ as well as an initial vector x_0.

In [1]:
```
#Abdelwahid Benslimane

import numpy as np

def conjugateGradient(A, b, x):

    A = A.copy()
    xold = x.copy()
    conjugate = b - np.dot(A, x)
    iterations = 0
    epsilon = 1e-12
    alpha = np.vdot(conjugate, conjugate)/np.vdot(conjugate, np.dot(conjugate, A))

    while (iterations < 500 and np.linalg.norm(conjugate, 2) > epsilon ) :

        xnew = xold + alpha*conjugate

        residual = b - np.dot(A, xnew)

        nextConjugate = residual - np.vdot(conjugate, np.dot(A, residual))/np.vdot(conjugate, np.dot(A, conjugate))

        alpha = np.vdot(nextConjugate, residual)/np.vdot(nextConjugate, np.dot(A, nextConjugate))

        xold = xnew

        iterations += iterations

        conjugate = nextConjugate

    if(iterations == 500):

        return np.nan
    else:

        return xnew
```

Let's use the function to find the solution of $Ax = b$ with A (symmetric definite positive matrix) and b as follows:

$$A = \begin{pmatrix} 2 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 1 & 2 \end{pmatrix}$$

$$b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

and the below initial vector x_0:

$$x_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

In [2]:
```
A = np.array([ [2., 1., 0., 0., 0.], [1., 2., 1., 0., 0.], [0., 1., 2., 1., 0.],
               [0., 0., 1., 2., 1.], [0., 0., 0., 1., 2.]])

b = np.array([1., 1., 1., 1., 1.])

x_init = np.array([1., 0., 0., 0., 0.])

solution = conjugateGradient(A, b, x_init)

print("The solution of the linear system of equations found with conjugateGradient is the vector:")

print(solution)
```

```
The solution of the linear system of equations found with conjugateGradient is the vector:
[5.00000000e-01 1.32100909e-12 5.00000000e-01 1.09944814e-12
 5.00000000e-01]
```

We can consider that 1.32100909e-12 and 1.09944814e-12 are equivalent to 0, thus the solution must be $[0.5\ \ 0.\ \ 0.5\ \ 0.\ \ 0.5]$. Let's compare now with the function linalg.solve available is the NumPy package.

In [3]:
```
solution = np.linalg.solve(A, b)

print("The solution of the linear system of equations found with linalg.solve is the vector:")
```

```
print(solution)
```

```
The solution of the linear system of equations found with linalg.solve is the vector:
[0.5 0.  0.5 0.  0.5]
```

The results have shown that the conjugate gradient method has very well aproximated the solution of $Ax = b$.

In [ ]:

In [ ]: