Gradient descent with variable optimal step size

Abdelwahid Benslimane

wahid.benslimane@gmail.com

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 $\frac{1}{2}x^T Ax - x^T b$ 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:

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

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:

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

The gradient descent method with variable optimal step size

The gradient descent method with variable optimal step size consists in optimising the descent step α at each iteration of the algorithm. The new step α_{k+1} is determined so as to minimise the quantity:

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

As:

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

then:

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

$<=>$

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

The minimum of f is reached for

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

$<=>$

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

As the matrix A is positive definite, $r_k^T A r_k > 0$, therefore we can write:

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

Below, my (basic) implementation of the gradient descent method with a variable optimal step size within the function optimalStepSizeGradientDescent 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 optimalStepSizeGradientDescent(A, b, x_init):

    A = A.copy()
    xold = x_init.copy()
    residual = b - np.dot(A, xold) #residual at initialization of the algorithm
    iterations = 0
    epsilon = 1e-15
    alpha = np.vdot(residual, residual)/np.vdot(residual, np.dot(residual, A)) #optimal step size at initialization
                                                                               #of the algorithm

    while (iterations < 500 and np.linalg.norm(residual, 2) > epsilon ) : #the max number of iterations is arbitrarily defined

        #we return from the loop if the max number of iterations is reached or if the L2 norm of the residual vector
        #is lower than the value of epsilon that has been defined,
        #which would mean that the algorithm has well converged to the solution

        xnew = xold + alpha*residual #we calculate x_{k+1}

        residual = b - np.dot(A, xnew) #we calculate the new residual that will be used at the next iteration

        alpha = np.vdot(residual, residual)/np.vdot(residual, np.dot(residual, A)) #we calculate the new optimal step size
                                                                                   #that will be used at the next iteration

        xold = xnew #we replace x_k by x_{k+1}

        iterations += iterations

    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 = optimalStepSizeGradientDescent(A, b, x_init)
```

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

print(solution)
```

```
The solution of the linear system of equations found with optimalStepSizeGradientDescent is the vector:
[5.00000000e-01 1.22361418e-15 5.00000000e-01 1.22154519e-15
 5.00000000e-01]
```

We can consider that 1.88379907e-16 and 1.53532898e-16 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]:
```python
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 gradient descent with variable optimal step size has very well aproximated the solution of $Ax = b$.

In [ ]:
```python

```