Centro de Investigación en Matemáticas, A.C.
Optimización

Tarea 4

José Miguel Saavedra Aguilar

Abstract

1 Introduction

On each iteration of a descent directions algorithm, one computes a descent direction d_k such that for some $T > 0$,

$$f(x_k) > f(x_k + td_k) \qquad\qquad t \in (0, T)$$

so we shall choose the *step length* α_k for the iteration

$$x_{k+1} = x_k + \alpha_k d_k$$

Ideally, we would like α_k to be the value of α that minimizes the function $\phi(\mathring{u})$ defined for $\alpha > 0$ by:

$$\phi(\alpha) := f(x_k + \alpha d_k)$$

However, this would be too expensive to compute, so we settle for values α such that $f(x_{k+1})$ is less enough than $f(x_k)$.

1.1 The Wolfe Conditions

Philip Wolfe presented line search conditions that α_k should follow to guarantee *sufficient decrease*. The first one, known as the *Armijo condition*[3], states that for some $c_1 \in (0, 1)$,

$$f(x_k + \alpha_k d_k) \leq f(x_k) + c_1 \alpha_k \nabla f^T(x_k) d_k \tag{1}$$

This condition states that α is only acceptable if $\phi(\alpha) < l(\alpha)$ for $l(\mathring{u})$ a linear function with slope $c_1 \nabla f^T(x_k) d_k$. Note if $c_1 = 1$, $l(\mathring{u})$ is the linear approximation of f at x_k. To rule out unacceptably short values of α_k, the *curvature condition* is introduced:

$$\nabla f^T(x_k + \alpha_k d_k) d_k \geq c_2 \nabla f^T(x_k) d_k \tag{2}$$

for some constant $c_2 \in (c_1, 1)$. Note for some functions, a step length may satisfy the Wolfe conditions (1) and (2) and not be particularly close to a minimizer of ϕ. For this reason, Wolfe [4] also presented what we now know as the *strong Wolfe conditions*:

$$\begin{aligned} f(x_k + \alpha_k d_k) &\leq f(x_k) + c_1 \alpha_k \nabla f^T(x_k) d_k \\ \left|\nabla f^T(x_k + \alpha_k d_k) d_k\right| &\leq c_2 \left|\nabla f^T(x_k) d_k\right| \end{aligned} \tag{3}$$

1.2 Line search methods

Now, we shall present three *line search methods*, this is, methods that attempt to find α_k such that they satisfy some of the Wolfe conditions.

1.2.1 Backtracking

The so-called *backtracking* approach consists on starting on an initial guess $\hat{\alpha}$ and setting $\alpha_k^{j+1} = \rho \alpha_k^j$ for some $\rho \in (0, 1)$ until Armijo's condition (1) is satisfied. While this is a simple algorithm, it is effective, even though may suffer from numerical precision and slow convergence problems.

1.2.2 Bisection

For the Bisection method, we shall search for the interval where we can guarantee both (1) and (2) are satisfied by starting from an initial guess $\alpha_k * 0$ and decreasing the value $\alpha_k^{j+1} < \alpha_k^j$ whenever (1) is not satisfied for α_k^j, increasing $\alpha_k^{j+1} > \alpha_k^j$ when (1) is satisfied but (2) isn't, while taking into consideration the previous iterations of α_k^j.

1.2.3 A line search algorithm for the Wolfe conditions

The third method is presented in [2] and guarantees to find a step length α satisfying (3). This algorithm is based on previous knowledge that in an interval $(\alpha_{lo}, \alpha_{hi})$, if the conditions:

1. α_{hi} violates Armijo's condition
2. $\phi(\alpha_{hi}) \geq \phi(\alpha_{lo})$
3. $\phi'(\alpha_{hi}) \geq 0$

There is an step length satisfying the strong Wolf conditions $\alpha_k \in (\alpha_{lo}, \alpha_{hi})$. We have a *zoom* function that shortens the interval $(\alpha_{lo}, \alpha_{hi})$ while maintaining the three conditions over the interval.
Then, we start at α^1 and two values α^0 and $b > \alpha^0$. Now, if α_k^j doesn't satisfy 1 or 2, we shall *zoom* in the interval (α^{j-1}, α^j), otherwise if $\phi'(\alpha^j) \geq 0$, we *zoom* in the interval (α^j, b).

2 Algorithm

Remember the descent directions algorithm:

Algorithm 1: Descent directions algorithm.

Input: f, x_0
Output: x^*

$k \leftarrow 0$;
while $\|\nabla f(x_k)\| > 0$ **do**
  Compute a descent direction d_k;
  Compute α_k;
  Update $x_{k+1} \leftarrow x_k + \alpha_k d_k$;
  $k \leftarrow k + 1$;
end

In order to compute α_k, we have the three algorithms:

Algorithm 2: Backtracking line search.

Input: $f, x_k, d_k, \alpha_0, \rho, c_1$
Output: α^*

$j \leftarrow 0$;
while $\phi(\alpha_j) > \phi(0) + c_1 \alpha_j \phi'(0)$ **do**
  Update $\alpha_{j+1} \leftarrow \rho \alpha_j$;
  $j \leftarrow j + 1$;
end

Algorithm 3: Bisection line search.

Input: $f, x_k, d_k, \alpha_0, c_1, c_2$
Output: α^*

$\alpha_{lo} \leftarrow 0$;
$\alpha_{hi} \leftarrow \infty$;
$j \leftarrow 0$;
repeat
  if $\phi(\alpha_j) > \phi(0) + c_1 \alpha_j \phi'(0)$ **then**
    $\alpha_{hi} \leftarrow \alpha_j$;
    $\alpha_{j+1} \leftarrow \frac{\alpha_{lo}+\alpha_{hi}}{2}$;
  else if $\phi'(\alpha_j) > c_2 \phi'(0)$ **then**
    $\alpha_{lo} \leftarrow \alpha_j$;
    $\alpha_{j+1} \leftarrow \begin{cases} 2\alpha_{lo}, & \alpha_{hi} = \infty \\ \frac{\alpha_{lo}+\alpha_{hi}}{2}, & \alpha_{hi} < \infty \end{cases}$;
  else
    α_j satisfies Wolfe's conditions;
  end
  $j \leftarrow j + 1$;
until *the Wolfe conditions are satisfied*;

Algorithm 4: A Line Search Algorithm.

Input: $f, x_k, d_k, \alpha_1, \alpha_{\max}, c_1, c_2$
Output: α^*

$\alpha_0 \leftarrow 0$;
$j \leftarrow 1$;
repeat
  if $\phi(\alpha_j) > \phi(0) + c_1 \alpha_j \phi'(0)$ ***or*** $\phi(\alpha_j) \geq \phi(\alpha_{j-1})$ **then**
    $\alpha_{j+1} \leftarrow$ **zoom**(α_{j-1}, α_j);
    α_{j+1} satisfies Wolfe's conditions;
  else if $|\phi'(\alpha_j)| \leq -c_2 \phi'(0)$ **then**
    α_j satisfies Wolfe's conditions;
  else if $\phi'(\alpha_j) \geq 0$ **then**
    $\alpha_{j+1} \leftarrow$ **zoom**$(\alpha_j, \alpha_{\max})$;
    α_{j+1} satisfies Wolfe's conditions;
  else
    Choose $\alpha_{j+1} \in (\alpha_j, \alpha_{\max})$;
  end
  $j \leftarrow j + 1$;
until *the Strong Wolfe conditions are satisfied*;

Algorithm 5: zoom.

```
Input: f, x_k, d_k, α_lo, α_hi, c_1, c_2
Output: α*
j ← 0;
repeat
    Choose α_j ∈ (α_lo, α_hi) a trial step length;
    if φ(α_j) > φ(0) + c_1 α_j φ'(0) or φ(α_j) ≥ φ(α_{j−1}) then
        α_hi ← α_j;
    else if |φ'(α_j)| ≤ −c_2 φ'(0) then
        α_j satisfies Wolfe's conditions;
    else if φ'(α_j)(α_hi − α_lo) ≥ 0 then
        α_hi ← α_lo;
    end
    α_lo ← α_j;
    j ← j + 1;
until the Strong Wolfe conditions are satisfied;
```

3 Results

Algorithm 1 was implemented in Julia[1] with all three algorithms 2, 3 and 4 for the descent direction $d_k = -\nabla f(x_k)$. In order to test the algorithms, we use Rosenbrock's function with $n = 2$ and $n = 100$, Wood (or Colville's) function and the following function:

$$f(x) = \sum_{i=1}^{n} (x_i - y_i)^2 + \lambda \sum_{i=1}^{n-1} (x_{i+1} - x_i)^2$$

for given y and $\lambda \in \{1, 10, 100\}$.

3.1 Rosenbrock's function

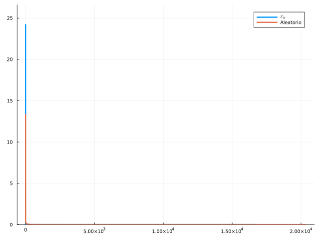
For Rosenbrock's function:

$$\sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2 \right]$$

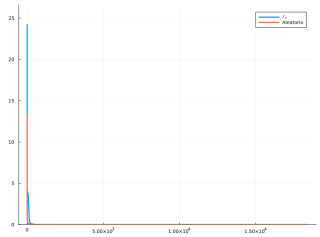
we attempt to find $x^* = \mathbb{1}$ starting from $x_0 = [-1.2, 1, 1, \ldots, 1, -1.2, 1]$ and from a random point. The results for $n = 2$ are presented on table 1 and for $n = 100$ on table 2.

Algorithm	x_0		Random point	
2	9.98963E-09	16677	9.94248E-09	20353
3	9.94534E-09	18426	9.96605E-09	18293
4	5.18805E-05	50000	2.83540E-05	50000

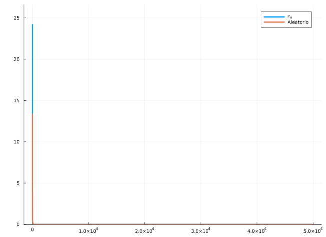
Table 1: $\|\nabla f(x_k)\|$ and iterations for Rosenbrock's function with $n = 2$.



(a) Backtracking line search

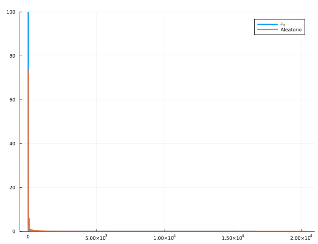


(b) Bisection line search

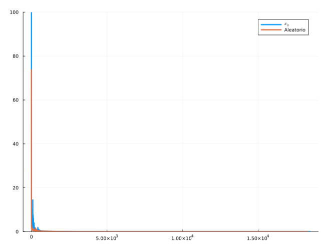


(c) Zoom line search

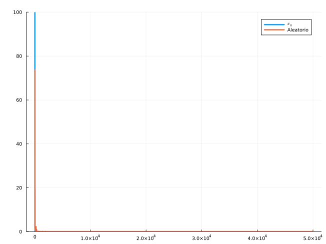
Figure 1: Evolution of Rosenbrock's function $(n = 2)$ value for x_k



(a) Backtracking line search



(b) Bisection line search

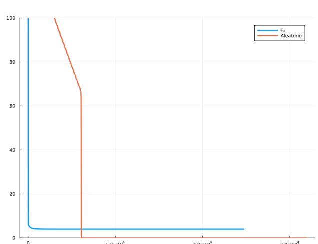


(c) Zoom line search

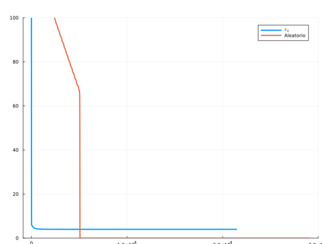
Figure 2: Evolution of the gradient of Rosenbrock's function $(n = 2)$ for x_k

Algorithm	x_0		Random point	
2	5.78938E-09	24735	9.74918E-09	31900
3	5.46950E-07	21449	9.97573E-09	29137
4	2.31670E-04	50000	9.54295E-04	50000

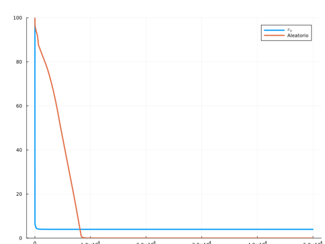
Table 2: $\|\nabla f(x_k)\|$ and iterations for Rosenbrock's function with $n = 100$.



(a) Backtracking line search

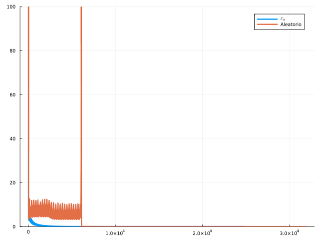


(b) Bisection line search

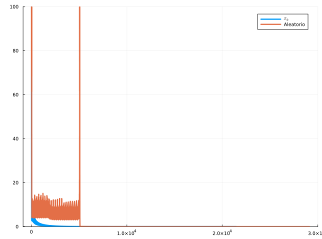


(c) Zoom line search

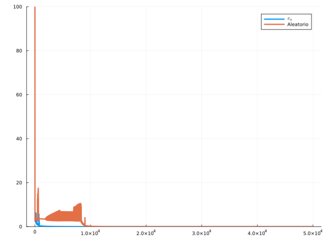
Figure 3: Evolution of Rosenbrock's function $(n = 100)$ value for x_k



(a) Backtracking line search



(b) Bisection line search



(c) Zoom line search

Figure 4: Evolution of the gradient of Rosenbrock's function ($n = 100$) for x_k

3.2 Wood function

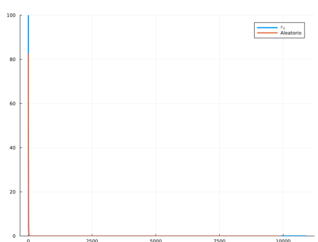
For Wood (or Colville) function:

$$f(x) = 100\left(x_1^2 - x_2\right)^2 - (x_1 - 1)^2 + (x_3 - 1)^2 + 90\left(x_3^2 - x_4\right)^2$$
$$+ 10.1\left((x_2 - 1)^2 + (x_4 - 1)^2\right) - 19.8\,(x_2 - 1)\,(x_4 - 1)$$

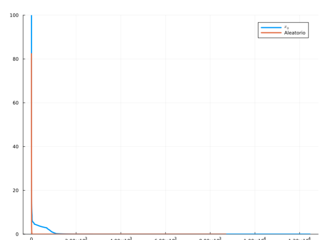
we attempt to find $x^* = \mathbb{1}$ starting from $x_0 = [-3, -1, -3, -1]$ and from a random point. We present the results on table 3.

Algorithm	x_0		Random point	
2	9.83209E-09	10893	9.90098E-09	9900
3	9.72676E-09	12461	9.66632E-09	8695
4	9.01513E-06	50000	4.76129E-04	50000

Table 3: $\|\nabla f(x_k)\|$ and iterations for Wood function.



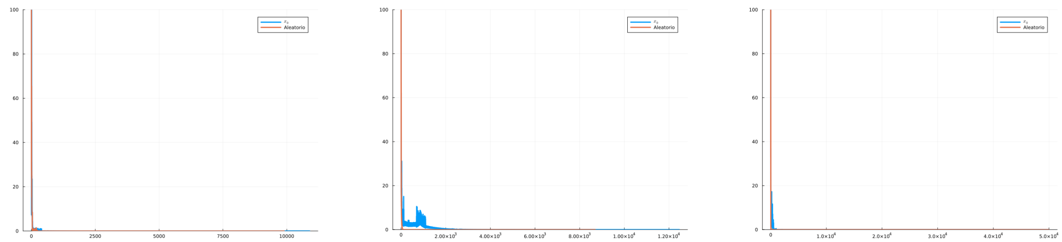
(a) Backtracking line search



(b) Bisection line search

(c) Zoom line search

Figure 5: Evolution of the Wood function value for x_k



(a) Backtracking line search (b) Bisection line search (c) Zoom line search

Figure 6: Evolution of the gradient of the Wood function for x_k

3.3 Noisy function

Let f be given by:

$$f(x) = \sum_{i=1}^{n}(x_i - y_i)^2 + \lambda \sum_{i=1}^{n-1}(x_{i+1} - x_i)^2$$

$$\begin{aligned} t_i &= \frac{2}{n-1}(i-1) - 1 \\ y_i &= t_i^2 + \eta \end{aligned}$$

for $i = 1, \ldots, n$ and $\eta \sim N(0, \sigma)$ a normal random variable with standard deviation $\sigma > 0$ and $n = 128$. We let $\sigma = 1$ and compute x^* starting from $x_0 = y$ using algorithm 4. We present the results on table 4.

	$\|\nabla f(x)\|$	Iterations
$\lambda = 1$	8.20692E-09	54
$\lambda = 10$	7.43941E-09	273
$\lambda = 1000$	4.09911E-05	50000

Table 4: $\|\nabla f(x_k)\|$ and iterations for different values of λ.

As required, we also present a scatter plot of the points t_i, y_i and x_i in figure 7.

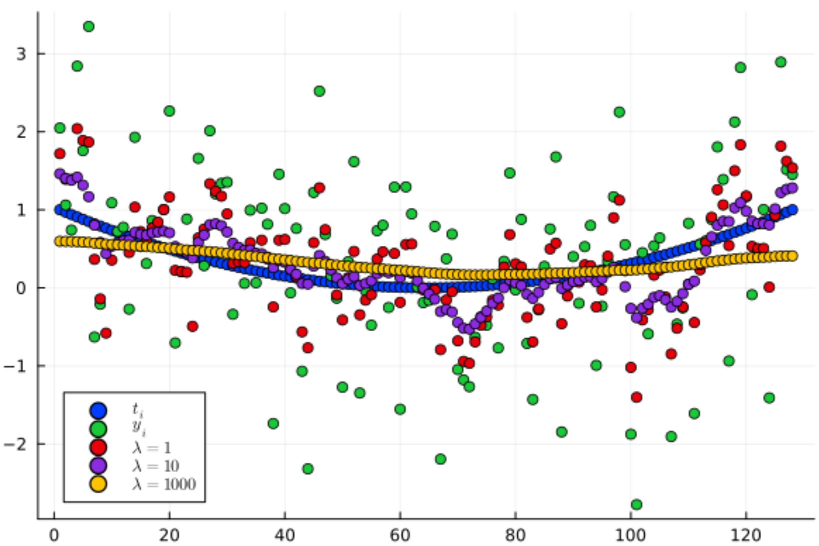


Figure 7: Scatter plot of t_i, y_i and $x^*(\lambda$

4 Results discussion and conclusions

We note both the backtracking and bisection algorithms perform as expected, with all instances but one achieving the expected tolerance 10^{-9}. However, for algorithm 4 it's not as expected, with the norm of the gradient $\|\nabla f(x_k)\|$ oscilating once it gets close to the optimal value. This may be explained by the fact the interval $(\alpha_{lo}, \alpha_{hi})$ becomes too small and $\phi(\alpha_{lo}$ and $\phi(\alpha_{hi}$ are equal to machine precision, so it won't be able to find an optimal value α_k.
However, all of the algorithms converge near the minimizer of each function, so we can say they are successful.

References

[1] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Review*, vol. 59, no. 1, pp. 65–98, 2017. [Online]. Available: https://epubs.siam.org/doi/10.1137/141000671

[2] J. Nocedal and S. Wright, *Numerical Optimization*, 2nd ed., ser. Springer Series in Operations Research and Financial Engineering. New York, NY: Springer, Jul. 2006.

[3] P. Wolfe, "Convergence conditions for ascent methods," *SIAM Review*, vol. 11, no. 2, pp. 226–235, Apr. 1969. [Online]. Available: https://doi.org/10.1137/1011036

[4] ——, "Convergence conditions for ascent methods. II: Some corrections," *SIAM Review*, vol. 13, no. 2, pp. 185–188, Apr. 1971. [Online]. Available: https://doi.org/10.1137/1013035