Synopsis of Nesterov's Random Gradient-Free Minimization

Pat Mellady

2022-12-08

Motivation

Gradient free optimization aims to optimize a function without the explicit use of the gradient. The simplest problem we can consider is $\min_{x\in\mathbb{R}} f(x)$, an unconstrained minimization of some differentiable function, f, over the entirety of $\mathbb{R}$, however, constrained optimization and problems with a non-differentiable objective function can also be considered. This report will summarize a recent paper by Nesterov [1] that provides the first theoretical analyses of a randomized gradient-free optimization method.

Random optimization methods have their roots in a very simple, yet incredibly costly, minimization technique. The earliest of these methods takes as input a starting point and then evaluates the function over some random sample of nearby points to see if one of the sampled points results in the function taking on a smaller value. This has many problems, mainly that this method is slow to converge and will prove to be costly. Thus, the demand for more efficient and more sophisticated techniques was high.

As an improvement from the techniques mentioned above, Nesterov provides a review of certain directional derivative methods presented by his advisor, Polyak, in 1987. Although these techniques were presented by Polyak, he provides no information about: the choice of step size, optimizing the finite differences calculation, or convergence of the methods. Nesterov's goal is to remedy these holes in Polyak's work.

Turning our attention to general optimization, many times we are incapable of using the gradient. Whether this be from the non-differentiability of a function or that the dimension of the problem makes finding the gradient time prohibitive, sometimes implementing gradient-free methods is the most efficient route. For this reason, Polyak noted that using a numerically approximated directional derivative of the function in a random direction proves to be a better use of the programmer's time when optimizing certain problems, as the numerical directional derivative is much simpler to implement and work with [1].

Beyond the man-power time efficiency of gradient-free methods, we have certain computational benefits from using the numerical directional derivative, mainly that we can make a more efficient use of system memory. Compared to Newton's method, where intermediate calculations must be stored in order to calculate the descent direction, the random gradient-free methods presented by Nesterov use no memory on intermediate calculations, but rather get directly to finding the search direction.

Applicability

Although, as stated above, the implementation of numerically approximated directional derivatives is best applied to problems where f is differentiable and convex, we may still utilize these methods in the case of non-smooth functions by estimating forward differentiation with finite differences. We will also see that useful theoretical convergence bounds can be placed on these methods when our objective function is convex, or has other useful properties. In this review, we will focus on one method outlined by Nesterov pertaining to random gradient-free unconstrained optimization of a convex and differentiable function, f. This particular method is limited in applicability by the theoretical requirements of the objective function to obtain convergence.

Mechanism

Nesterov presents several methods in his paper, but they all share the same foundation. To be comfortable with this foundation, we start with the definition of the directional derivative [2].

Definition Directional Derivative

> Given some scalar function $f : \mathbb{R}^n \to \mathbb{R}$ and a vector $v \in \mathbb{R}^n$, we define the directional derivative of f in the direction v to be
>
> $$f'(x, v) = \lim_{h \to 0} \frac{f(x+hv)-f(x)}{h}$$

It is important to note that $f'(x, v) = \nabla f(x) \cdot v \in \mathbb{R}$ where $\nabla f(x)$ is the gradient of f. When we calculate the directional derivative using finite differences, we call this a numerical approximation of the directional derivative.

We have one final preliminary definition [3] to achieve the best understanding of the methods outlined by Nesterov.

Definition Gaussian Random Vector

> Let $u_k \in \mathbb{R}^n$, then u_k is a Gaussian random vector if $u_k = b + Az$ where $b \in \mathbb{R}^n$, $A \in \mathbb{R}^{nxk}$, $z = \left[\ z_1, z_2 \cdots, z_n \ \right]^T \in \mathbb{R}^n$ where z_i~N(0,1) are iid with $k \leq n$.

For the analysis done in this paper, we will take $b = \left[\ 0, 0, \cdots, 0 \ \right]^T$ and $A = I_n$. We now proceed to the algorithms outlined in the paper. Each method is built upon the implementation of a random oracle of the form

$$g_\mu(x_k) = \frac{f(x_k+\mu_k u)-f(x_k)}{\mu_k} \cdot u$$

where $\mu_k \in \mathbb{R}$ and $u \in \mathbb{R}^n$ is a vector as described above. Note that this preliminary definition of the random oracle closely mimics a numerical directional derivative of the objective function. Nesterov presents three versions of this oracle [1]:

1. $g_\mu(x_k) = \frac{f(x_k+\mu_k u)-f(x_k)}{\mu_k} \cdot u$
2. $\hat{g}_\mu(x_k) = \frac{f(x_k+\mu_k u)-f(x_k-\mu_k u)}{2\mu_k} \cdot u$
3. $g_0(x_k) = \lim_{\mu_k \to 0} \frac{f(x_k+\mu_k u)-f(x_k)}{\mu_k} \cdot u = f'(x_k, u) \cdot u$

We can see that each of these three oracles closely follows the definition of the directional derivative where the third method uses the true definition. The main difference between each of these methods

and the true definition as presented above is that the real directional derivative is a scalar, while these oracles are in $\mathbb{R}^n$. This is because we are multiplying the random vector u by the directional derivative.

The above oracles present methods to find random search directions to be used in an updating step. Similar to the algorithms we are familiar with from class, we will have an updating step of

$$x_{k+1} = x_k - hg_\mu(x_k) \cdot u$$

where h is our step size, which will be described in more detail later.

Beyond the simple implementation of the method above, Nesterov [1] outlines two modifications: accelerated gradient-free minimization and non-convex gradient-free minimization. Both of these methods will make use of the oracles above and the principles of the numerical directional derivative, but will have different implementation schemes regarding acceleration and the handling of the non-convexity of the objective function. The difference in implementation parameters produce different choices for the finite difference constant, the step size, and affect the convergence rate of each method.

The last mechanistic component of these methods is how they are implemented in the constrained optimization case. As outlined by Nesterov [1], the same random oracle and updating formula will be used, but it is necessary to project that updated estimator onto the set defined by the constraints. As discussed in class, this is not always an easy task, and finding the projection mapping can prove to be impossible in some cases.

Convergence Analysis

The primary component in understanding the convergence analysis of the random gradient-free unconstrained method is that we do **not** have deterministic convergence like we do when implementing non-random algorithms, such as gradient descent. However, in the approaches outlined by Nesterov, we have probabilistic convergence. We obtain convergence from the expected value of the objective function with respect to u_{k-1}. Understanding this leads to an important definition [1].

Definition ϕ_k

Let U_k be the k^{th} random vector used in the algorithm when computing $g_\mu(x_k)$. We then define:

$$\phi_k = E_{U_{k-1}}(f(x_k))$$

where $x_{k+1} = x_k - hg_\mu(x_k) \cdot u_k$.

Armed with the definition of ϕ_k, we can now present the method and its convergence rate.

The RG_μ Method

Our problem will be unconstrained minimization of a convex and differentiable function, f, over $\mathbb{R}^n$. We will assume that the problem is solvable with minimum f^* occurring at some $x = x^*$. We will denote this method by RG_μ and will make use of the first random oracle from our list.

For the RG_μ method, we have the following algorithm [1]:

Initialize: x_0, μ, N, h, and k=0
While(k<N){
choose u_k **as described above**
calculate $g_\mu(x_k)$
$x_{k+1} = x_k - hg_\mu(x_k)$
k=k+1
}
return x_{k+1}

From this method, we have the following convergence result [1], which doubles as a method to choose the step size parameter:

Theorem

Let $f : \mathbb{R}^n \to \mathbb{R}$ be convex and have Lipschitz continuous gradient with Lipschitz constant L_1 and denote by $\{x_k\}_{k\geq 0}$ a sequence generated by the RG_μ method, then for $h = \frac{1}{4(n+4)L_1}$ and $N \geq 0$, we have:

$$\frac{1}{N+1}\sum_{k=0}^{N}(\phi_k - f^*) \leq \frac{4(n+4)L_1||x_0-x^*||^2}{N+1} + \frac{9\mu^2(n+4)^2L_1}{25}$$

Furthermore, if f is strongly convex and $\tau(f) \geq 0$ is the convexity parameter, we denote $\delta_\mu = \frac{18\mu^2(n+4)^2L_1}{25\tau(f)}$, and we have that:

$$\phi_N - f^* \leq \frac{L_1}{2}\left[\delta_\mu + (1 + \frac{\tau(f)}{8(n+4)L_1})^N(||x_0 - x^*||^2 - \delta_\mu)\right]$$

Some Analysis and Intuition about the Theorem

This theorem gives us ample information about the choice of step size and the theoretical probabilistic convergence of the method. Firstly, we should realize that the theorem tells us an optimal choice for the step size exists, as stated in the initial assumptions. Nesterov does not go into the derivation of this step size, but does illustrate that this choice of step size will be beneficial to us when applying earlier theorems to achieve the convergence bounds stated in the above result.

Secondly, another important lesson to learn from this theorem is that the right hand side of the above inequalities place upper bounds on the convergence of the method. As we will see, the choice of μ will be dependent on how accurate we want our result to be, which depends on a predetermined constant $\epsilon > 0$. Thus, the right hand side of the inequality can be made arbitrarily small by allowing $N \to \infty$ and choosing ϵ to be small, which will give us a bound on the probabilistic convergence of the method.

Choosing the Finite Difference Step Size

As stated above, we will need to choose μ sufficiently small in order to generate a numerical solution within an acceptable predetermined accuracy of the true solution. By choosing $\mu \leq \frac{5}{3(n+4)}\sqrt{\frac{\epsilon}{2L_1}}$, we will obtain that the right hand side will approach an arbitrarily small error bound. It is important to note that, when we substitute this value of μ into the error bound, we obtain:

$$\frac{9\mu^2(n+4)^2L_1}{25} \leq \frac{9(\frac{5}{3(n+4)}\sqrt{\frac{\epsilon}{2L_1}})^2(n+4)^2L_1}{25} = \frac{9\frac{25}{9(n+4)^2}\frac{\epsilon}{2L_1}(n+4)^2L_1}{25} = \frac{\epsilon}{2}$$

Complexity Analysis

From results earlier in Nesterov's paper, we know that $E_u(||u||)$ is on the order of $O(n^{\frac{1}{2}})$, which results in the method being on the order of $O(\sqrt{\frac{\epsilon}{nL_1}})$ [1]. It is important to note that the method is quicker to converge in the strongly convex case, due to a necessary alteration in the finite difference step size. When f is strongly convex, the method will be on the order of $O(\frac{nL_1}{\tau(f)} ln(\frac{L_1}{\epsilon}))$ [1].

Strengths

High Dimensional Problems

One obvious strength of the method is the ease of use when solving a high dimensional problem. In the case of other methods, the programmer would be forced to calculate the gradient, and possibly the Hessian, of the objective function. These computations may not be much for a problem of two or three dimensions, but when problems can reach one-hundred dimensions, they can be quite difficult. Thus the use of finite differences and random directions can be significantly more time efficient for the programmer.

Weaknesses

Differentiability

Although other methods in Nesterov's paper may be used to circumvent this restriction, the main drawback of this method is the necessity for our function to be differentiable with a Lipschitz continuous gradient. Part of the attraction of gradient-free methods is to rid the problem of the differentiability restriction. That being said, the more appropriate name for the method presented would be random *exact* gradient-free minimization, as this implementation necessitates the differentiability of the function.

Finding the Lipschitz Constant

When choosing the step and finite difference size, we are confronted with needing to know the Lipschitz constant for the gradient. If we wish to avoid computing the gradient, we will need to find some approximation for the Lipschitz constant.

Numerical Experiment

For our example, we will consider the minimization of the Booth function, defined by:

$$f(x_1, x_2) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$$

The Booth function achieves its minimum at $x^* = (1, 3)$ where $f(x^*) = 0$.

Implementation of the Booth Function

Firstly, Let us implement the necessary related functions in R.

```r
#Booth function
f<-function(x){
  x1<-x[1]
  x2<-x[2]
  f_x<-(x1+2*x2-7)^2+(2*x1+x2-5)^2
  return(f_x)
}

gradf<-function(x){
  x1<-x[1]
  x2<-x[2]
  return(c(10*x1+8*x2-34, 8*x1+10*x2-38))
}

hess_f<-function(x){
  hf<-matrix(c(10,8,8,10), byrow=TRUE, ncol=2)
  return(hf)
}
```

For the purposes of the RG_μ method, we only need to have the Booth function itself. However, we are interested in the best Lipschitz bound for ∇f and we want to compare the RG_μ method against other methods, specifically the gradient descent method, so we add the necessary gradient and Hessian for finding the direction, performing the back-tracking line search for the step size, and using the multivariate mean-value theorem to find the Lipschitz constant.

Implementation of the RG_μ Method

Next, we will implement the RG_μ method as outlined above. We will need a function to calculate random oracle $g_\mu(x_k)$ and a function to repeat the algorithm to desired accuracy. These are outlined below.

```r
# Gradient free random method ######################################################

g_mu<-function(x, u, mu){
  gmu_k<-(1/mu)*(f(x+mu*u)-f(x))*u
  return(gmu_k)
}

RG_mu<-function(guess, K, sol, epsilon=1e-3){
  n<-length(guess)

  H<-hess_f(x0)
  L<-max(eigen(H)$value)
  mu<-(3/(3*(n+4)))*sqrt((epsilon)/(2*L))

  x_k<-guess
  h<-1/(4*(n+4)*L)
  k<-0
```

```
  x_trace<-c(f(x_k), x_k)
  while(k<K){
    u_k<-matrix(rnorm(n, 0, 1), ncol=1)
    g_k<-g_mu(x_k, u_k, mu)
    x0<-x_k
    x_k<-x_k-h*g_k
    k<-k+1

    x_trace<-rbind(x_trace, c(f(x_k), x_k))

    if(sqrt(sum(sol-x_k)^2)<epsilon){
      break
    }
  }
  x_trace<-as.data.frame(x_trace)
  names(x_trace)<-c("f",paste('X', 1:n, sep=" "))
  rownames(x_trace)<-NULL
  return(list(x_k=x_k, x_trace=x_trace, k=k))

}
```

For purposes of analysis in this portion, and since we know the minimizer of the Booth function, we will artificially stop the algorithm when we are near enough to the actual solution, which can be seen in the if statement nested in the algorithm. This will give us a better understanding of the performance of the algorithm in comparison with the gradient descent method.

Implementation of Gradient Descent for Comparison

Now, we will add the gradient descent method with back-tracking line search for comparison with the RG_μ method.

```
# Gradient descent for comparison ##############################################

back_track<-function(rho, c, alpha0, x_k, dir){
  alpha_k<-alpha0
  while(f(x_k+alpha_k*dir)>f(x_k)+c*alpha_k*gradf(x_k)%*%dir){
    alpha_k<-alpha_k*rho
  }

  return(alpha_k)
}

steep_descent<-function(guess, epsilon=1e-3, rho=.3, c=1e-4, alpha0=1){
  x_k<-guess
  n<-length(x_k)
  k<-0
  x_trace<-c(f(x_k), x_k)
  while(sqrt(sum(gradf(x_k)^2))>epsilon){
```

```
    dir<--gradf(x_k)
    alpha<-back_track(rho, c, alpha0, x_k, dir)
    x_k<-x_k+alpha*dir
    k<-k+1
    x_trace<-rbind(x_trace, c(f(x_k), x_k))
    if(alpha<1e-15){
      break
    }
  }
  x_trace<-as.data.frame(x_trace)
  names(x_trace)<-c("f",paste('X', 1:n, sep=" "))
  rownames(x_trace)<-NULL
  return(list(x_k=x_k, x_trace=x_trace, k=k))
}
```

Comparing the Performance of the Methods

With the gradient descent and RG_μ methods implemented above, we are ready to compare their performance. We will observe two metrics here: the functional value at each x_k against the iteration number, and the number of iterations necessary for achieving certain levels of accuracy. To illustrate this, we have the following:

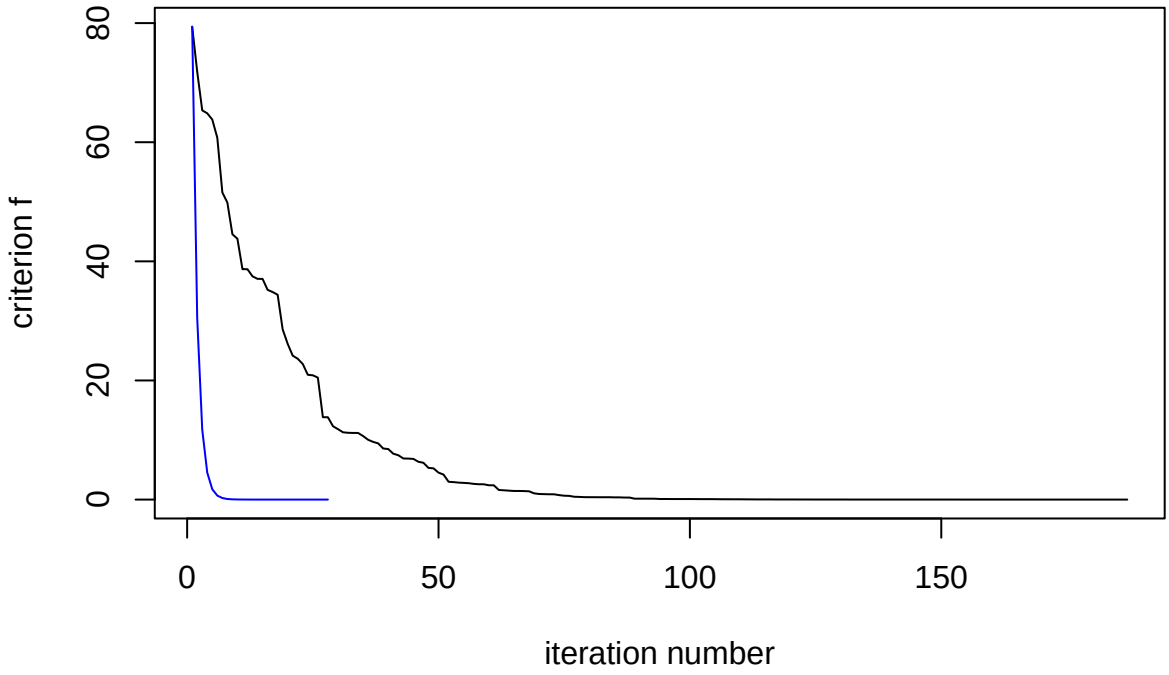
```
set.seed(405)
x0<-matrix(c(-1.2, 1), ncol=1)
sol<-matrix(c(1,3), ncol=1)
RG<-RG_mu(x0, 1e6, sol)
cat("RG took",RG$k,"iterations","\n")
```

```
## RG took 187 iterations
```

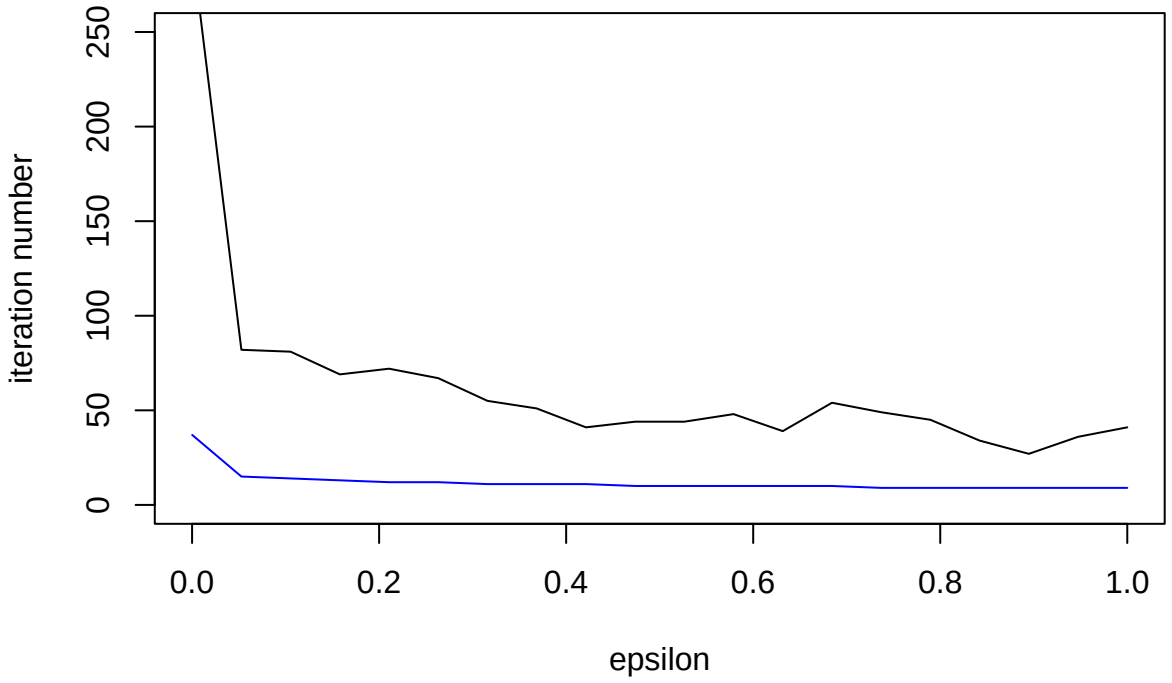
```
x0<-c(-1.2,1)
GD<-steep_descent(x0)
cat("GD took",GD$k,"iterations","\n")
```

```
## GD took 28 iterations
```

```
# Plot of f vs iteration number for a single value of epsilon
plot(1:RG$k, RG$x_trace[1:RG$k, 1], type="l",
        xlab="iteration number", ylab="criterion f")
lines(1:GD$k, GD$x_trace[1:GD$k, 1], col="blue")
```




```
# Plot of iterations required to reach certain accuracy values
m<-20
epsilon<-seq(1e-4,1, length=m)
RG_ep<-c()
GD_ep<-c()
for(i in 1:m){
  RG_ep<-rbind(RG_ep, c(RG_mu(x0, 1e6, sol, epsilon[i])$k))
  GD_ep<-rbind(GD_ep, c(steep_descent(x0, epsilon[i])$k))
}
plot(epsilon[m:1], RG_ep[m:1], type="l",
      xlab="epsilon", ylab="iteration number", ylim=c(0,250))
lines(epsilon[m:1], GD_ep[m:1], col="blue")
```



Analysis of the Results

The first thing we see is that the RG_μ method took nearly ten times as many iterations to converge as gradient descent. This is illustrated well in the first plot of the function value against the iteration number. Gradient descent is plotted on the blue line which steeply drops off after only several iterations while RG_μ has a much gentler decline to the minimum, however, both of these methods do converge.

The second plot shows the number of iterations required to reach certain levels of accuracy, ranging from $\epsilon = 1$ to $\epsilon = 0.0001$. As we would expect, both methods require more iterations to converge as the desired accuracy increases. An important note is that RG_μ uses more iterations at each accuracy level than gradient descent, with a drastic increase in required iterations as our accuracy drops below $\epsilon = 0.1$.

Although the RG_μ does not seem to perform well when compared to gradient descent, this problem is a simple illustration of the method. Supposing we had a one-hundred dimensional problem, in general, the time needed to calculate the necessary components for gradient descent, or many other methods, would be substantial when compared to the implementation of RG_μ, given we could approximate the Lipschitz constant.

Proceeding from this paper, it would be interesting to analyze the performance of the RG_μ method where the random vector has entries that follow different distributions, not just standard normal. The effects of different distributions could be implemented in certain constrained optimization cases or in performing simpler convergence analyses.

References

1. Nesterov, Y., Spokoiny, V. *Random Gradient-Free Minimization of Convex Functions. Found Comput Math 17, 527–566 (2017).* https://doi.org/10.1007/s10208-015-9296-2
2. Marsden, J. E., & Tromba, A. *Vector calculus. New York: W.H. Freeman, 164-166 (2003).*
3. Gubner, J. A. *Probability and Random Processes for Electrical and Computer Engineers. Cambridge University Press., 362-364 (2006).*