

PEPit.jl tutorial: Worst-case regret of online gradient descent

John Doe

December 12, 2025

In this tutorial, we will consider the online convex minimization problem. We want to sequentially minimize the regret

$$R_n \triangleq \min_{x \in Q} \sum_{i=1}^n f_i(x_i) - f_i(x),$$

where the functions f_i are M -Lipschitz and convex, and where Q is a bounded closed convex set with diameter upper bounded by D . We also denote by $x_\star \in Q$ a reference point with respect to which we will compute the regret. Classical references on the topic include [1, 2]. We can also study these algorithms using computer assisted approach such as performance estimation programming (PEP) in [3] and using the related IQCs in [4].

The following code computes a worst-case guarantee for **online gradient descent** (OGD) with a step-size $\gamma = \frac{D}{M\sqrt{n}}$. That is, it computes the smallest possible $\tau(n, M, D)$ such that the guarantee

$$R_n \leq \tau(n, M, D)$$

is valid for any such sequence of queries of OGD; that is, x_t are the query points of OGD.

In short, for given values of n, M, D : $\tau(n, M, D)$ is computed as the worst-case value of R_n . Let us see how to do it step by step in PEPit.jl!

Algorithm Recall that online gradient descent (OGD) is described by

$$x_{t+1} = x_t - \gamma \nabla f_t(x_t),$$

where $\gamma = D/M\sqrt{n}$ is a step-size. Let us start with loading the necessary PEPit.jl related packages.

```
using PEPit, OrderedCollections
```

Let instantiate the performance estimation problem (PEP) that we will populate step by step.

```
problem = PEP()
```

We now set the parameters of the problem: the Lipschitz constant M of the losses, the diameter D of the constraint set, and the number of iterations n . For this example, we set $M = 1$, $D = 0.5$, and $n = 2$.

```
M, D, n = 1.0, 0.5, 2
```

Let us now set the stepsize of the online gradient descent method.

```
gamma = D / (M * sqrt(n)) # Step-size  $\gamma = \frac{D}{M\sqrt{n}}$ 
```

Now we define a sequence of convex M -Lipschitz losses f_1, \dots, f_n .

```
fis = [declare_function!(problem, ConvexLipschitzFunction, OrderedDict("M" => M)) for _ in  
↪ 1:n] # Convex  $M$ -Lipschitz losses  $f_1, \dots, f_n$ 
```

The next step is to declare the indicator function $h = \iota_Q$ of a convex set Q with diameter upper bounded by D .

```
h = declare_function!(problem, ConvexIndicatorFunction, OrderedDict("D" => D)) # Indicator  
↪  $h = \iota_Q$  of a convex set  $Q$  with  $\text{diam}(Q) \leq D$ 
```

Next, define the aggregate loss $F = \sum_{i=1}^n f_i$ used in the regret term.

```
F = sum(fis) #  $F = \sum_{i=1}^n f_i$ 
```

Define a reference point $x_\star \in Q$, which we are modeling here as a projected point in the support of h .

```
x_ref = set_initial_point!(problem) # Start with some arbitrary reference point  $x_\star$ .  
  
x_ref, _, _ = proximal_step!(x_ref, h, 1) # We want to ensure that  $x_\star$  is in  $Q$ , so we do  
↪  $x_\star \leftarrow \text{Proj}_Q(x_\star)$  here. Because  $h$  is an indicator function, the proximal step is a  
↪ projection onto  $Q$ .  
  
_, F_ref = oracle!(F, x_ref) # Computes  $F(x_\star)$  that is function value at  $x_\star$ , because this is  
↪ needed in the regret definition.
```

Define the starting point $x_1 \in Q$.

```
x = set_initial_point!(problem) # Pick some arbitrary starting point  $x_1$ .  
x, _, _ = proximal_step!(x, h, 1) # Again want to ensure that  $x_1 \in Q$ , so we project  $x_1$  onto  
↪  $Q$ , i.e.,  $x_1 \leftarrow \text{Proj}_Q(x_1)$ .
```

Let us now run n projected subgradient steps and record the incurred losses $f_t(x_t)$.

```

f_saved = Vector{Expression}(undef, n) # In this array we will save the incurred losses  $f_t(x_t)$ .

for i in 1:n
    g_i, f_i = oracle!(fis[i], x) # Get a subgradient  $g_i \in \partial f_i(x_i)$  and the function value  $f_i(x_i)$ 
    ↪ at the current iterate  $x_i$ .
    f_saved[i] = f_i # Save the incurred loss  $f_i(x_i)$ .
    x, _, _ = proximal_step!(x - gamma * g_i, h, gamma) # Projected subgradient step:
    ↪  $x_{t+1} = \text{Proj}_Q(x_t - \gamma f'_t(x_t))$ 
end

```

Finally, we set the performance metric to be the regret after n iterations!

```

set_performance_metric!(problem, sum(f_saved) - F_ref) #  $R_n = \sum_{t=1}^n f_t(x_t) - F(x_*)$ .

```

Okay, we have built the PEP! Now, we can solve it to obtain a worst-case guarantee on the regret of online gradient descent.

```

pepit_tau = solve!(problem; verbose=true)

```

What does the theory say about the regret of online gradient descent? Theoretical results from [2, Section 2.1.2] says that:

$$R_n \leq MD\sqrt{n}.$$

```

theoretical_tau = M * D * sqrt(n)

```

Let us now compare both results!

```

@info "PEPit guarantee:\t R_n <= $(round(pepit_tau, digits=6))"

@info "Theoretical guarantee:\t R_n <= $(round(theoretical_tau, digits=6))"

```

If everything went well, both guarantees should match up to numerical precision!

References. Ther references are as follows.

- [1] E. Hazan (2016). Introduction to online convex optimization. Foundations and Trends in Optimization, 2(3-4), 157-325. <<https://arxiv.org/pdf/1912.13213>>
- [2] F. Orabona (2025). A Modern Introduction to Online Learning. <<https://arxiv.org/pdf/1912.13213>>
- [3] J. Weibel, P. Gaillard, W.M. Koolen, A. Taylor (2025). Optimized projection-free algorithms for online learning: construction and worst-case analysis <<https://arxiv.org/pdf/2506.05855>>
- [4] F. Jakob, A. Iannelli (2025). Online Convex Optimization and Integral Quadratic Constraints: A new approach to regret analysis <<https://arxiv.org/pdf/2503.23600>>