Gradient descent implementation example applied to Celestial mechanics

Samy Badjoudj

Abstract

This document aims to describe gradient descent algorithm, its implementations and how it can be used on a simple celestial mechanics example.

1 Introduction

1 Gradient descent is an optimization algorithm. The aim of gradient descent is to converge to a local minimum/maximum of a function. We call it a first order derivative iterative algorithm. It is labeled first order derivative, since we use first derivative of the function during the iterations.

There are plethora of applications, easy to imagine, that requires converging to a local minimum/maximum (Machine Learning, mechanics, biology...), often the function we want to minimize is called cost function noted $J(\theta_n)$.

Since Gradient descent is widely used, you can find easily an implementation in many programming languages.

Here we will go through the basics of the theory, then we will apply it on a practical case, Earth Sun orbital system.

2 Requirement

In order to apply the gradient descent algorithm using a function these conditions must be met.

Given $f : R^n \to R,\ \ f(x_n)$ then $\forall x_n \ \exists\ f'(x_n)$

Where for a given a we have $f'(a) = \lim_{h\to 0} \frac{f(a+h)-f(a)}{h}$ exists

2.1 Few differentiable functions

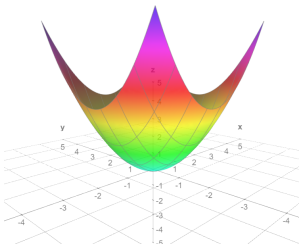

Figure 1: $x^2 + y^2$

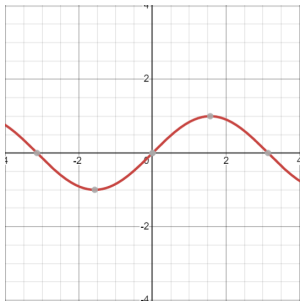

Figure 2: $sin(x)$

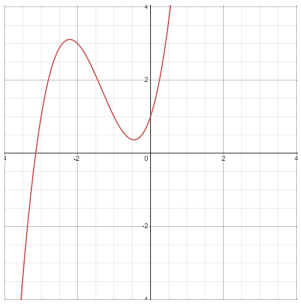

Figure 3: $(x+1)^3 + x^2(x)$

2.2 Few non differentiable functions

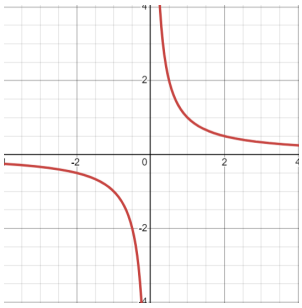

Figure 4: $1/x$

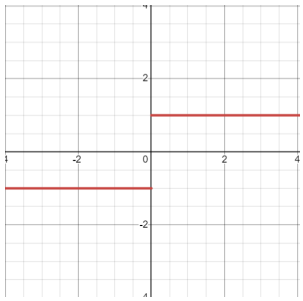

Figure 5: $x/|x|$

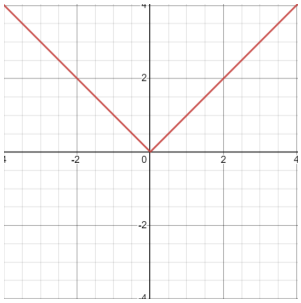

Figure 6: $|x|$

Note: that the algorithm, can fall on what we call a saddle point, meaning for a given point (x_0, y_0) *we have* $f_x(x_0, y_0) = 0$ *and*$f_y(x_0, y_0) = 0$ *so defined as critical point, but since it is not an extremum, it ends up being only a saddle point. To sort it out we can use the Hessian matrix for a second partial derivatives test*

3 Gradient descent algorithm description

The algorithm consist in finding the minimum through an iterative process. The value S_{min} will be updated and will end with the value we are looking for.

Let's define some parameters

1. $f : R^n \to R$, *noted* $f(x_n)$ *where* $x_n \in R$
2. α a scalar as learning rate
3. M as number of iterations
4. i current iteration
5. S_i current point
6. $\nabla f(S_i)$ where the gradient is $\nabla f(\, x_1, x_2, \ldots, x_n) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(x_1, x_2, \ldots, x_n) \\ \frac{\partial f}{\partial x_2}(x_1, x_2, \ldots, x_n) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x_1, x_2, \ldots, x_n) \end{bmatrix}$

$S_i \leftarrow [a_0, a_1..]$ ▷ initialize starting point
$S_{min} \leftarrow S_i$ ▷ initialize minumum
$M \leftarrow 50$ ▷ maxmium of iterations
$\alpha \leftarrow random(\,]0, 1[\,)$ ▷ ranged learning rate
for $i = 0$, $i < M$, $i++$ **do**
    $S_{i+1} \leftarrow S_i - \alpha * \nabla f(S_i)$
    if $f(S_{n+1}) < f(S_{min})$ **then**
        $S_{min} \leftarrow S_{i+1}$ ▷ update minium if necessary
    end if
end for

---

4 Application on orbital Earth Sun system

Let's apply this algorithmic on a concrete case. Earth Sun system could be an interesting candidate.

In our Solar system Earth orbits around the Sun along an ellipse in a period of $\backsimeq 365$ *days* with an average speed of $29.7827\, km.s^{-1}$.

Here is a figure that helps visualize (ellipse is flatened on purpose) :

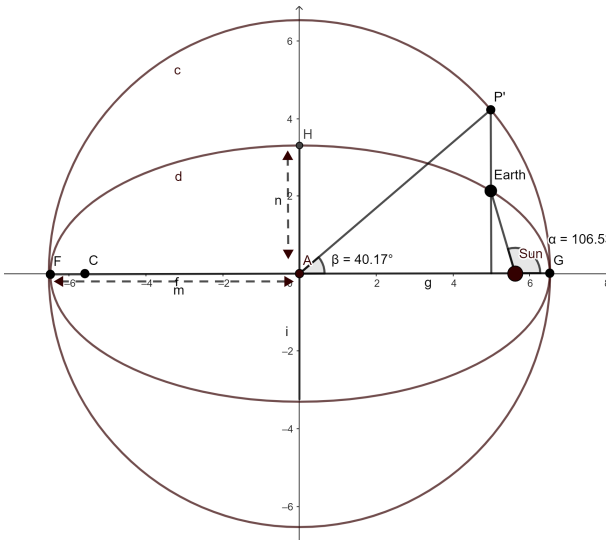

Figure 7: Orbital Earth Sun system

4.1 Ellipse characteristics

Few elements that characterizes an ellipse d :

- Semi-major axis $m = \overline{\mathrm{F\,A}}$
- Semi-minor axis $n = \overline{\mathrm{H\,A}}$
- Right focal length $g = \overline{\mathrm{A\,Sun}}$
- Two focuses, points C and S
- Eccentricity $= \frac{g}{m} = \sqrt{1 - (\frac{n}{m})^2}$
- Any point P located on the ellipse $(x, y) = (mcos(t), nsin(t)), t \leq 0 \leq 2\pi$

4.2 Orbital Earth Sun system description

We are interested in computing the orbital speed of the Earth around the Sun. To have a better understanding, let's write some contextual formulas notions and data.

4.2.1 Some parameters

- True anomaly (α in Figure 7) angle $\angle Earth\,Sun\,G$
- Orbital speed, speed of Earth orbiting around the Sun's center
- $G \backsimeq 6.674 \times 10^{-11}\, m^3.kg^{-1}.s^{-2}$ empirical constant factor used when describing force between two objects
- AU, Astronomical Unit, distance between Earth Sun, $1.4959787 \times 10^{11} m$

- Aphelion $152,097,597\,km$, distance when Earth is the furthest from the Sun
- Perihelion $147,098,450\,km$, distance when Earth is the nearest from the Sun
- Semi-major axis $149,598,023\,km$ (ellipse figure is flattened for visibility)
- Eccentricity 0.0167086
- Vis-Viva equation: $OrbitalSpeed = \sqrt{G \times SunMass \times (\frac{2}{radius} - \frac{1}{SemiMajorAxis})}$

5 Example of implementation

5.1 Code relative to Earth Sun System

5.1.1 Compute Radius from Sun ≡ focus on the right, in Km

```
public double getRadiusFromFocusInKm(double semiMajorAxisInMeter, double eccentricity, double angleInDegree){
    return  (semiMajorAxisInMeter * (1 - Math.pow(eccentricity, 2))) /
            (1 + (eccentricity * Math.cos(getRadianFromDegree(angleInDegree))));
}
```

5.1.2 Compute orbital speed around massive object

```
public double getOrbitalSpeed(double massObjectInKg,double radiusInMeter, double semiMajorAxisInMeter){
    return Math.sqrt(G * massObjectInKg * ((2 / radiusInMeter) - (1 / semiMajorAxisInMeter)));
}
```

5.2 Example of Gradient descent Implementation

```
public GradientDescent<Double> getMinByGradientDescent(int maxIteration, Double learningRate, Double start){
    Derivative2DFunction function = computationService.getEarthOrbitalSpeed2D();
    Double min = start;
    List<Double[]> path = new ArrayList<>();
    for (int i = 0; i < maxIteration; i++) {
        double scaledGradient = learningRate * function.getDerivative(start);
        path.add(new Double[]{start,function.apply(start)});
        Double newPoint = start - scaledGradient;
        start = newPoint;
        if (function.apply(newPoint) < function.apply(min)) {
            min = newPoint;
        }
    }
    return new GradientDescent<>(path,min);
}
```

In red we see the orbital speed oscilliating between ≡ 30,3 km/s and 29,3 km/s.

If we draw the function for degrees more than 360°, we will see more clearly

5.3 Gradient descent plot

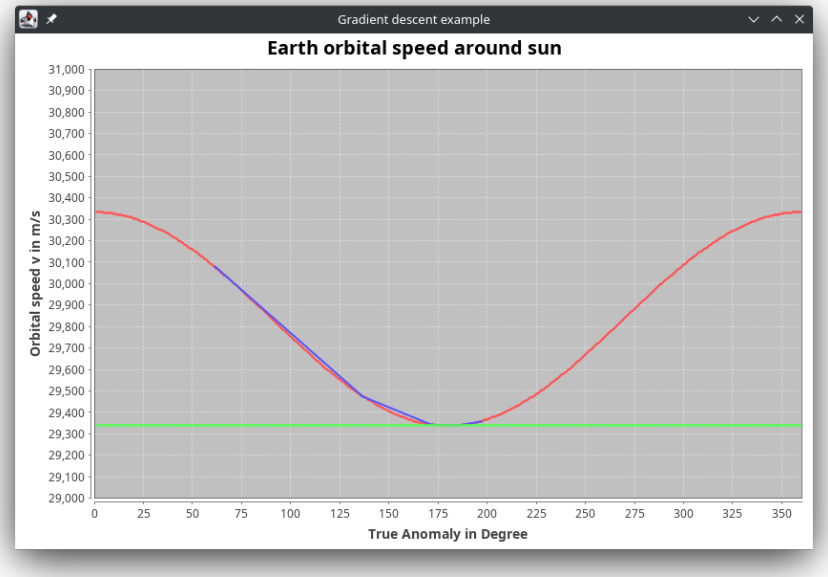

Figure 8: Orbital Earth Sun system

that the function is periodic with a period of $360^{\circ}(2\pi$ in gradient).

Then we see in blue the path followed by the gradient descent (α learning rate has been put high on purpose to see the broken line).

We can notice that over the iterations, the blue, follows the red curve. The blue curve is a broken line, the gradient direction scaled by the learning rate. Behind the scenes, the algorithm keeps track of the minimum. Here we can see that green line, is drawn at 180 $^{\circ}(e.g\,\pi\,radians)$.

5.4 Comments on results

The above implementation shows how a local minimum can be found by using a first derivative order, and a simple structured algorithm. Gradient descent, can be scaled to a function with higher dimensions.

We will see in a next article in the context of machine learning, how we can leverage Gradient descent to solve a polynomial regression problem, by minimizing the cost function