Optimizing the Racing Line

Shravan Godse, Niranjan Bhombe, Parth Malpathak

24-703: Numerical Methods Project (Spring 2023)

Contents

1 Objective

The objective of this projects is to leverage on numerical methods concepts and techniques especially domain discretization and gradient descent optimization to conclude on the most optimal racing line which the drive should take while executing a turn on a race track. By applying these techniques, the project aims to explore the physics and mechanics involved in racing on a track, tire performance, to develop a comprehensive understanding of how to achieve the fastest and most efficient racing line. Ultimately, the objective of the project is to provide insights and recommendations that can be applied by drivers and teams to optimize their racing lines, leading to improved performance and increased chances of success on the track. The project aims to demonstrate the effectiveness of numerical methods and gradient descent optimization techniques in solving complex problems and enhancing performance in the field of Formula 1 racing.

2 Motivation

Formula 1 engineers have long relied on data generated from the track to improve racing performance. Even before the advent of Machine Learning and AI algorithms, engineers would meticulously record lap times and driver inputs to fine-tune the performance of F1 cars. However, with the advent of technology, F1 cars are now equipped with over 300 sensors, which capture a multitude of parameters, such as throttle input, suspension moment, tyre wear, engine performance, and more.

The primary objective of this project was to apply numerical methods to motor-sport. Engineers in F1 and other motor-sports frequently employ optimization techniques to formulate race strategies that can effectively reduce tyre wear and improve engine performance and longevity. For this project, we focused on discovering the most optimized racing line for reducing tyre wear and improving lap times, utilizing various gradient descent optimization algorithms.

2.1 Race Line

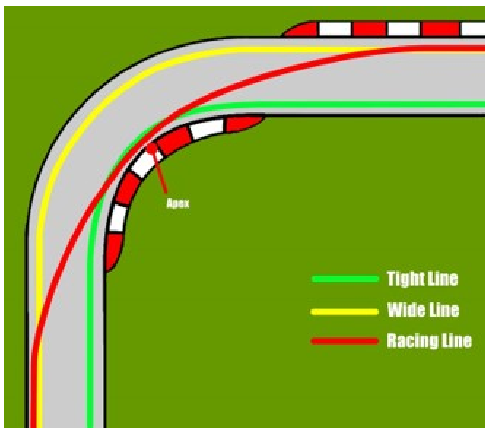

Figure 1: Ideal Racing Line

Racing line is the optimal path 1 around a race-track which the participating cars of the motor-sport use to get the most optimized performance out of the vehicle [1]. Using the racing line has multiple benefits in a race. Following are a few of those:

1. Minimum lap time during a qualifying session to achieve better grid slots during the race.

2. Conservation of Tires and Fuel for better track battle.

3. Fending off a pass from a competitor vehicle during a race.

In a race, a corner can be divided into three parts: the entry, apex, and exit. The entry 2 is the initial phase of the corner where the driver begins to decelerate the car, and prepares to negotiate the turn. The driver has to position the car correctly on the track and decide on the speed at which they will take the turn. This speed depends on factors such as the car's capabilities, track conditions, and the driver's skill level. A good driver will aim to take the corner at the highest possible speed while maintaining control of the car.

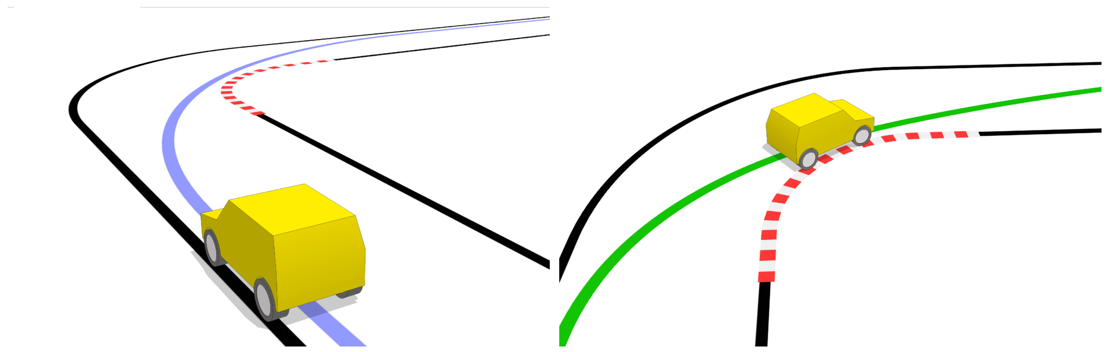
Figure 2: Phases of Corner Entry and Apex

The apex 2 is the point at which the car is closest to the inside of the corner. This is the point where the driver aims to "clip" the inside of the turn, and is critical to achieving the fastest possible lap time. The apex point is determined by a variety of factors, such as the shape of the corner, the car's handling, and the driver's skill level. Typically, the apex point is located towards the end of the entry phase, and it is important for the driver to be able to identify the correct point, and hit it with precision.

After passing the apex, the driver begins to accelerate the car towards the exit of the corner. This is the final phase of the turn, where the driver aims to maintain the maximum possible speed while ensuring that the car stays within the track limits. A good driver will try to maximize their speed on the exit of the corner, as this will help them to carry momentum onto the following straight, and ultimately achieve a faster lap time.

3 Problem Setup

3.1 Domain

We chose a 90°corner represented in fig. 3 as our base problem. We are using polar coordinates to discretize the system. Polar coordinates are chosen over cartesian coordinates as several points have either the same x or y cartesian coordinate. For θ, the domain is given by $\theta \in [0, \pi/2]$. Domain for r is given by the following equations

$$r \in \begin{cases} \left[\frac{d}{\cos\theta}, \frac{d+w}{\cos\theta}\right], & \text{if } \theta \in [0, \pi/4] \\ \left[\frac{d}{\sin\theta}, \frac{d+w}{\sin\theta}\right], & \text{if } \theta \in [\pi/4, \pi/2] \end{cases} \tag{1}$$

Where $d = 40$ meters, which is representative of the inner radius and $w = 6$ meters which is the minimum permissible track-width[2]. We discretize θ over 25 uniformly spaced points from 0 to $\pi/2$. Thus, the racing line could be uniquely described by a set of $\{r, \theta\}$ coordinates. We tried using more points in near the corner so that the turn will have higher resolution, however, this resulted in unphysical trajectories during optimization.

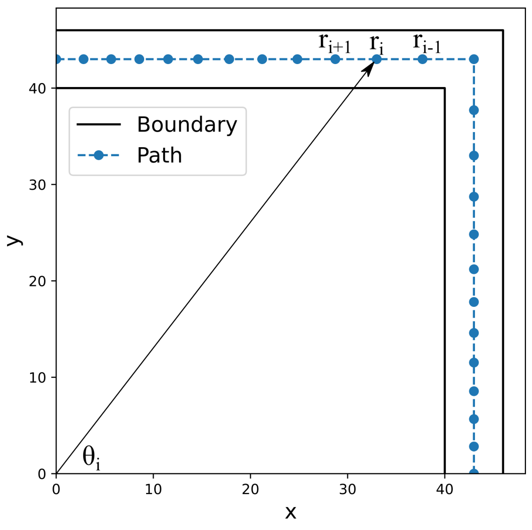

Figure 3: Domain Discretization

3.2 Strategy

To simplify the model, we assume that the corner is flat without any banking and the centripetal force is provided solely by friction whose coefficient is given by μ. If the instantaneous radius of curvature is given by r_{curv}, then the maximum speed that the F1 car can have without toppling over at that instant is given by

$$v_{\max} = \sqrt{\mu g r_{\text{curv}}} \tag{2}$$

Thus, at every node, i, the maximum speed possible at the node will be determined by the corresponding radius of curvature r_{curv_i}, given by

$$r_{\text{curv}_i} = \frac{r_{\text{i}}^2 + r_{\theta_i}^2}{|r_i^2 + r_{\theta_i}^2 - r_i r_{\theta\theta_i}|} \tag{3}$$

where r_{θ_i} is the first derivative of r with respect to θ evaluated at r_i and $r_{\theta\theta_i}$ is the second derivative. These derivatives are approximated using the central difference scheme,

$$r_{\theta_i} = \frac{dr}{d\theta} = \frac{r_{i+1} - r_{i-1}}{\theta_{i+1} - \theta_{i-1}} \tag{4}$$

$$r_{\theta\theta_i} = \frac{d^2 r}{d\theta^2} = \frac{r_{i+1} - 2r_i + r_{i-1}}{\theta_{i+1} - \theta_{i-1}} \tag{5}$$

We adopt the following strategy:

> For a given path, find the maximum possible speed at every node. Choose the minimum of these and traverse along the path with this chosen speed.

With this strategy, the time required will be given by

$$t = \frac{\sum_{i=1}^{N} \sqrt{(r_{i+1} - r_i)^2 + r_i(\theta_{i+1} - \theta_i)^2}}{\min_i \sqrt{\mu g r_{\text{curv}_i}}} \tag{6}$$

Since, we have discrete values $r_1, r_2, \ldots r_N$, eqn. 6 is a function of these parameters which we want to optimize, thus,

$$\text{Raceline} = \arg \min_{r_1, r_2, \ldots, r_N} t(r_1, r_2, \ldots, r_N) \tag{7}$$

3.3 Gradient Descent Optimization

From a numerical methods perspective, gradient descent [3] is an iterative method that is used to solve optimization problems by minimizing a given cost function. The cost function is usually a function of multiple variables, and the goal of gradient descent is to find the values of these variables that minimize the cost function.

The basic idea of gradient descent is to start with an initial guess for the variables, and then iteratively update the variables using the gradient of the cost function. The gradient of the cost function is the vector of partial derivatives of the cost function with respect to each variable.

To perform a single iteration of gradient descent, the following steps are performed:

1. Compute the gradient of the cost function: This involves calculating the partial derivative of the cost function with respect to each variable.

2. Choose a step size: This is a small value that determines how far to move in the direction of the gradient. The step size is often denoted by alpha.

3. Update the variables: The variables are updated by subtracting the product of the step size and the gradient from the current values of the variables.

4. Repeat steps 1-3 until convergence: The process of computing the gradient, choosing a step size, and updating the variables is repeated until the cost function is minimized or a stopping criterion is met.

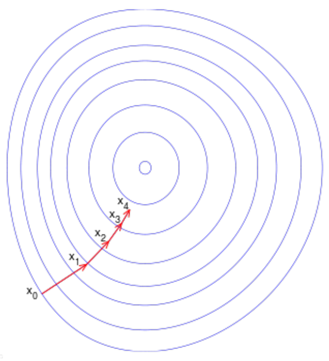

Figure 4: Vanilla Gradient Descent

Fig. 4 shows the Vanilla Gradient Descent Optimization. Specifically, every iterations, we can see the steps reaching the most optimized parameter value. One of the key advantages of gradient descent is that it is a simple and intuitive method that can be applied to a wide range of optimization problems. The following equation demonstrates the update step of Gradient Descent Optimization:

$$r_{t+1} = r_t - \alpha \nabla f(r_t) \tag{8}$$

In eqn. 8, r represents the vector of parameters that we want to optimize, $J(r)$ is the cost function that we want to minimize, α is the learning rate (a hyperparameter that controls the step size of each iteration), and $\triangledown J(r)$ is the gradient of the cost function with respect to the parameters.

However, it also has several limitations, such as the possibility of getting stuck in local minima, slow convergence, and sensitivity to the choice of step size. To address these issues, there are many variations of gradient descent [4], such as stochastic gradient descent, mini-batch gradient descent, and momentum-based methods, that can improve the speed and accuracy of the optimization process. To overcome few of these limitations, we have studied the incorporation of momentum in Gradient Descent Optimization.

3.4 Gradient Descent Optimization with Momentum

Gradient descent with momentum [5] is an advanced optimization algorithm that builds upon the basic principles of gradient descent to accelerate the convergence of the optimization process. The momentum algorithm adds a momentum term to the standard gradient descent update rule, which helps the optimization process to continue in the same direction as the previous update.

The basic idea of gradient descent with momentum is to use the gradient of the cost function to update the parameters, but to also add a fraction of the previous update vector to the current update vector. This means that if the previous update was in the same direction as the current update, the momentum term amplifies the current update, leading to faster convergence. If the previous update was in the opposite direction, the momentum term reduces the current update, helping to avoid oscillations and overshooting.

The update rule for gradient descent with momentum can be expressed as follows:

$$v_t = \beta v_{t-1} + (1-\beta)\nabla f(r_t) \tag{9}$$

$$r_{t+1} = r_t - \alpha v_t \tag{10}$$

In this equation, r_t is the current estimate of the parameters, $f(r_t)$ is the cost function, $\nabla_\theta f(r_t)$ is the gradient of the cost function with respect to the parameters, α is the learning rate, β is the momentum coefficient (a hyperparameter that controls the contribution of the momentum term), and v_t is the momentum term at time step t.

The first equation (eqn. 9) computes the momentum term v_t as a weighted sum of the previous momentum term v_{t-1} and the current gradient $\nabla f(r_t)$. The momentum term effectively "remembers" the previous updates, which helps to smooth out the optimization process and reduce oscillations.

The second equation (eqn. 10) updates the parameters by subtracting the momentum term from the current estimate of the parameters. This means that the update is a combination of the current gradient and the previous momentum, which helps to accelerate the optimization process in the direction of the previous updates.

3.5 Gradient Descent Optimization with ADAM

As an extension of our work of testing the effect of Momentum on Gradient Descent, we have also tested the effect of ADAM on the Optimization process through Gradient Descent [6].

Gradient descent optimization with ADAM (Adaptive Moment Estimation) is another advanced optimization algorithm that improves upon the basic principles of gradient descent by adapting the learning rate for each parameter. ADAM combines the ideas of momentum-based optimization and adaptive learning rate methods to achieve faster convergence and better performance on a wide range of optimization problems.

The update rule for gradient descent optimization with ADAM can be expressed as follows:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla f(r_t) \tag{11}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla f(r_t))^2 \tag{12}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{13}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{14}$$

$$r_{t+1} = r_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{15}$$

3

In this equation, r_t is the current estimate of the parameters, $J(r_t)$ is the cost function, $\nabla f(r_t)$ is the gradient of the cost function with respect to the parameters, α is the learning rate, β_1 and β_2 are exponential decay rates for the moment estimates, ϵ is a small constant used to avoid division by zero, m_t and v_t are the first and second moment estimates of the gradients, and $\hat{m}_t$ and $\hat{v}_t$ are bias-corrected estimates of the moment vectors.

The first equation (eqn. 11) computes the exponentially-weighted moving average of the gradients, which is used to estimate the mean of the gradients.

The second equation (eqn. 12) computes the exponentially-weighted moving average of the squared gradients, which is used to estimate the variance of the gradients.

The third and fourth equations (eqn. 13 and eqn. 14 respectively) correct the bias in the estimates of the moment vectors by dividing them by the corresponding powers of the decay rates.

The fifth equation (eqn. 15) updates the parameters by subtracting the product of the bias-corrected moment vector and the learning rate, scaled by the square root of the bias-corrected variance vector plus a small constant ϵ.

Overall, ADAM is a powerful optimization algorithm that can adaptively adjust the learning rate for each parameter based on the estimated gradient statistics. This helps to accelerate convergence and improve stability on a wide range of optimization problems. However, it also has additional hyperparameters to tune, such as the learning rate, decay rates, and small constant, which can affect the performance of the algorithm.

4 Results

As an initial guess, we start with the path at the exact center of the domain. Gradient descent algorithms are performed for 400 iterations which are enough to obtain convergence. Further, using same number of iterations allows us to compare stability and effectiveness of different gradient descent algorithms. The results for each algorithm are shown below followed by a discussion comparing the different methods.

4.1 Vanilla Gradient Descent

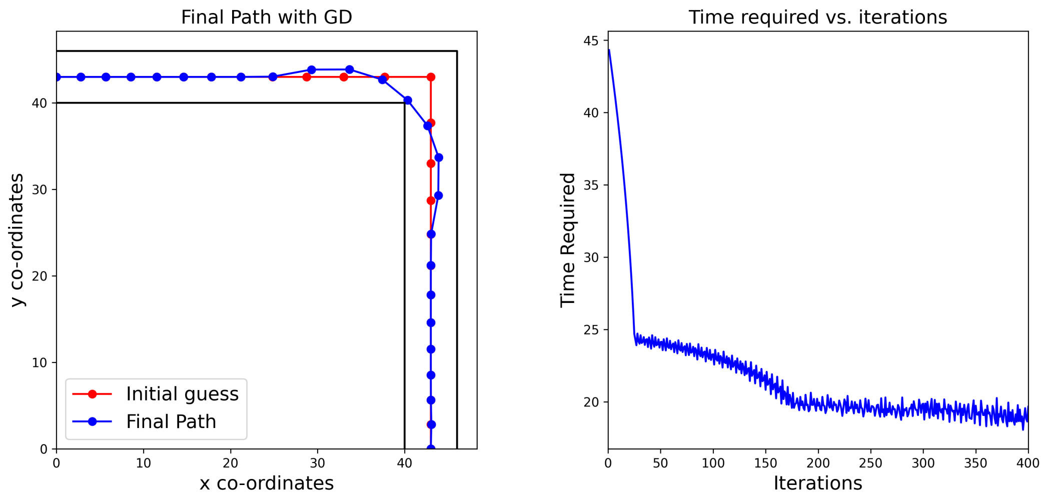

Figure 5: Optimizing with gradient descent

4.2 Gradient Descent with Momentum

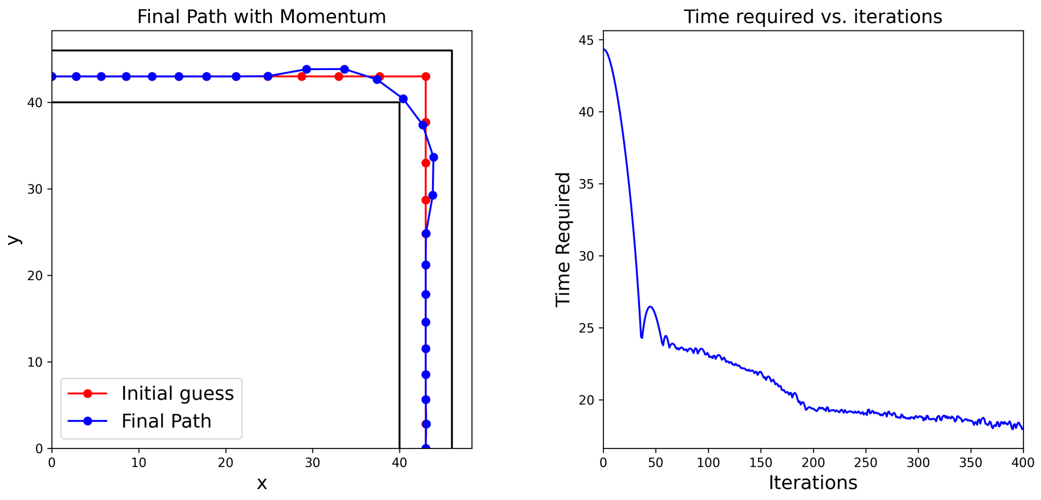

Figure 6: Gradient Descent with Momentum for fastest time

4.3 Gradient Descent with ADAM

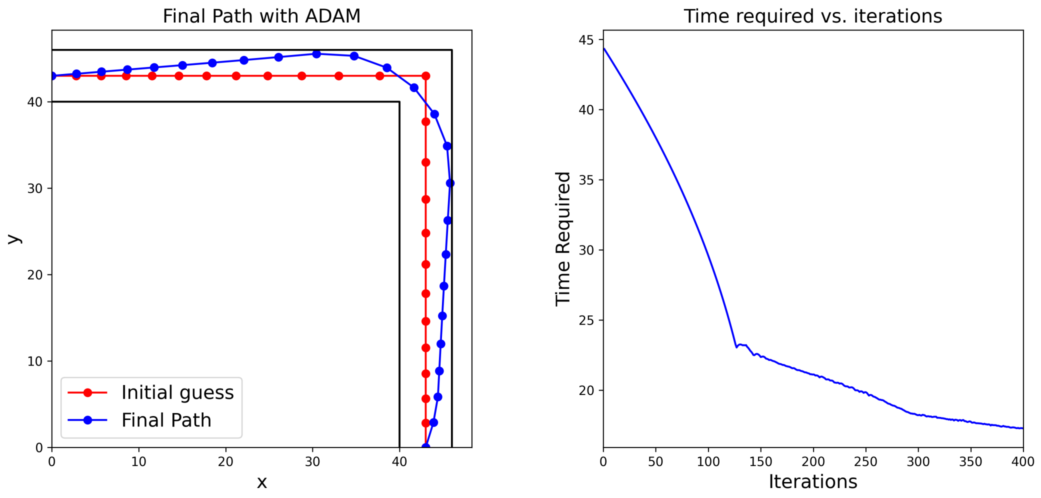

Figure 7: Gradient descent with ADAM for fastest time

4.4 Comparison

We observe that convergence is achieved within 400 iterations for all three methods. However, near the optimal path, we observe fluctuations when using vanilla gradient descent. Since our learning rate is a constant, we overshoot the optimal values when taking a step, which causes the time required to fluctuate with iterations. These fluctuations are mitigated when using an adaptive learning rate in momentum and ADAM techniques. The fluctuations are significantly lower when using momentum GD and are nearly absent when using the ADAM optimizer.

Table 1 shows a comparison of final time required to traverse the path. ADAM obtains the lowest time amongst the tree GD variants. Further, all of them nearly require the same time.

Method	Optimized Time (s)	Time for Iterations (s)
Vanilla GD	19.6712	7.5555
Momentum	18.0243	7.0968
ADAM	17.2812	7.2172

Table 1: Comparison of GD and variants

Since we have vectorized the update operations, they all nearly take the same time. However, momentum and ADAM require more memory to store additional parameters such as v_t and m_t.

5 Conclusions

1. By discretizing the domain and applying gradient descent, we are able to obtain the optimal racing line, minimizing tire wear and reduce time spent in the corner.
2. Using gradient descent with momentum helps reduce fluctuations that arise in vanilla gradient descent.
3. With Adaptive moment estimation the racing line obtained is closest to the preferred racing line in Formula 1.

References

[1] The Racing Line. `https://physicsofformula1.wordpress.com/the-racing-line/`. [Online; accessed April-2023].

[2] FIA track grades: Requirements to hold an F1 race, potential tracks. `https://us.motorsport.com/f1/news/fia-track-grades-requirements-f1-potential/6508331/`, 2021. [Online; accessed April-2023].

[3] Gradient Descent. `https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21/`, 2021. [Online; accessed April-2023].

[4] Gradient Descent Optimizing Algorithms. `https://www.ruder.io/optimizing-gradient-descent/https://www.ruder.io/optimizing-gradient-descent/`, 2016. [Online; accessed April-2023].

[5] Gradient Descent with Momentum. `https://towardsdatascience.com/gradient-descent-with-momentum-59420f626c8f/`, 2020. [Online; accessed April-2023].

[6] Gradient Descent with ADAM. `https://towardsdatascience.com/understanding-gradient-descent-and-adam-optimization-472ae8a78c10/`, 2020. [Online; accessed April-2023].