PolyMC

Ruben Navasardyan

August 2022

Introduction

This paper will explain the intuition and the mathematics behind the PolyMC algorithm intended for generating a step-size for an interval for the ODE solver. The algorithm is intended to be seen as a mere extension of the already existing IPA (formerly HarvardX-v2) partitioning algorithm.

The general intuition behind the algorithm is that there more consistently volatile the data is within a given sub-interval, the larger the step size. It further utilizes the Graham Scan algorithm for finding the convex hull of a set of points and the Monte Carlo simulations together with the Inside-Outside problem for estimating the area of the convex hull.

Graham Scan

This is an algorithm with worst case complexity of $O(n \log n)$, and this is also the worst case complexity of the whole PolyMC algorithm, that is designed to generate a list of points of vertices of the smallest possible polygon that contains all the datapoints.

The algorithm works by first assigning the leftmost point as the pivot. Then the list of points (excluding the pivot) is generated in the ascending order of slopes of line segments of each point with the pivot. The output hull is generated by creating a new list and placing the pivot and the first point of the aforementioned list as the first and second elements respectively and one by one adding points from that previous list to this output list. However, once the length of the output list exceeds 2, every time a new point is added, a condition must be checked. that is the cross product of the last three points (the newly added one included) is not a positive number to ensure that the points are neither colinear and nor is the middle point to the right of the line segment that connects the first and the third points or else the middle point will be removed. This process will be repeating every time a new point is added and by the end of it, the needed convex hull will be generated.

Monte Carlo Simulations

To estimate the area of any given manifold in $\mathbb{R}^2$ we use Monte Carlo simulations for the area. we start by assigning a rectangle in which the manifold is a proper subset of (in our case we take the rectangle defined by the start and the end of the interval as the vertical lines and the maximum and minimum y values for the horizontal lines). We further generate n number of random (pseudorandom is also acceptable) points in the rectangle and count the number of them that land within the bounds of the manifold and also calculate the area of the rectangle. The estimated area of the manifold is

presented by the following formula

$$A_{\text{manifold}} = A_{\text{rectangle}} \times \frac{\text{number of points in manifold}}{\text{total number of points}}$$

Note: the larger the n value, the more accurate the estimate is and if $n \to \infty$, then the estimate converges to the actual area.

Inside-Outside Problem

In the Monte Carlo simulations section, we mentioned that the algorithm requires the knowledge of whether a given point lies inside or outside the given manifold. This is the so called Inside-Outside Problem and the solution to it is to generate a ray pointing in any direction starting at our given point. The solution entails counting the number of times the ray intersects the boundary of the manifold. If it's an even number of times, then the point is outside and if odd, then inside.

In our specific case, we generate a horizontal line with slope of 0 and generate line segments representing every edge of the polygon. We further loop through the list of these edges and find the points of intersection. If the x value of the point of intersection is larger that the x value of the point and is within the bounds of the x values of the endpoints of a given edge, only then do we count this intersection as valid. Performing this process for every edge and checking whether the sum is even or odd, we get to know if the point is inside or outside.

Note: points on the boundary count as outside of the polygon.

Putting This All Together

The step PolyMC algorithm works by generating the convex hull of all data points within the given interval and further estimates the area of the polygon using the Monte Carlo simulations and utilizing the solution to the Inside-Outside Problem to determine if a given point inside or outside the polygon. This gives us a rather accurate estimate of the area of the polygon and we further calculate the ratio of its area to the rectangle defined by the x bounds of the interval and the y maximum and minimum values of the data.

The recommended step size is $\inf\left\{\Delta_{\text{interval}}, \frac{A_{\text{rec}}}{A_{\text{poly}}}k\right\}$. Where k is some user defined coefficient, default is $k = 1$. The reason to why we're taking the reciprocal of the above ratio is because the more consistent the volatility, the more space the polygon take up from rectangle and we reciprocate the original ratio to find by what factor it is smaller and use that as one of the recommendation (unless it's larger than the size of the interval in which case we take the step to be the entire interval).

The algorithm further permits the user to define a coefficient to be applied universally to all intervals. That is, the step size can be uniformly customized by providing a coefficient that the ratio will be multiplied by. The uniformity comes from the fact that the coefficient itself doesn't modify the way the ratio is generated and that the final pre-infimum step size is merely a factor away from the original algorithm calculates step size (i.e. the reciprocated ratio).

Time Complexity Analysis

Given a dataset, the algorithm first runs the Graham scan which has $O(n \log n)$ time complexity, then it finds the maximum and minimum values in the dataset which is $O(n)$ and lastly, the algorithm performs the Monte Carlo method (which includes finding solutions to the Inside-Outside problem for each randomly generated point) and it is either $O(1)$ if the number of randomly generated points is constant or $O(n)$ if the number depends on the size of the data set however this doesn't change the overall complexity of the algorithm. Hence the worst case time complexity of PolyMC is $O(n \log n) + O(n) + O(n) = O(n \log n + 2n) = O(n \log n)$.

The step by step of the PolyMC algorithm:

1. Given the data the algorithm calculates the convex hull of the data set (the smallest convex polygon that contains it) using the Graham scan algorithm that has $O(n \log n)$ time complexity. It generates a sequence of points when connected with a closed straight-edged path will be the polygon P

2. Creates the smallest rectangle R that contains the polygon by finding the absolute max and min values of the data set and taking the difference of them to be the width while the length is Δ

3. To estimate the area of the polygon rather fast and within a small error we use the Monte Carlo method. The algorithm generates n random tuples of (x, y) coordinates within R (the larger the n the more accurate the estimate) and counts how many land within the polygon P and the area of the polygon is calculated as $A_P = A_R \cdot \frac{\#\text{ of points in P}}{n}$ but since we use $\frac{A_R}{A_P}$ as the ratio to calculate the step size then we just use $\frac{n}{\#\text{ of points in P}}$ right away.

4. Given a point inside the rectangle R to check whether it's inside or outside P we generate a ray starting at that point in any direction (computationally easiest is to have a ray with slope 0) and count how many times in intersects the boundary of the polygon. Since the edges are straight lines it intersects each edge at most once hence we just need to find the point of intersection of the ray with each linear function representing an edge (if it exists) and check whether the point of the intersection is within the bounds of the edge it represents.

5. If the total number of times the ray intersects the boundary of P is odd then the point is in P and if even then outside of P. Having counted the number of points in P we calculate s.