Diophantine Approximation

Fernando Javier López Cerezo

November 5, 2021

1 Objectives

The objective of this project is to estimate the complexity of an algorithm which given $\epsilon > 0$ and $x \in \mathbb{R}$ will find the best diophantine approximation. In other words the algorithm will return a rational number $\frac{p}{q}$ such that $|x - \frac{p}{q}| < \epsilon$ and $|x - \frac{p}{q}| < |x - \frac{p'}{q'}| \quad \forall \frac{p'}{q'} \in \mathbb{Q}$ such that $0 < q' \leq q$. In particular we want to find the average complexity of an algorithm implementing the mediant approximation approach.

2 Experimental Setup

For this project we used:

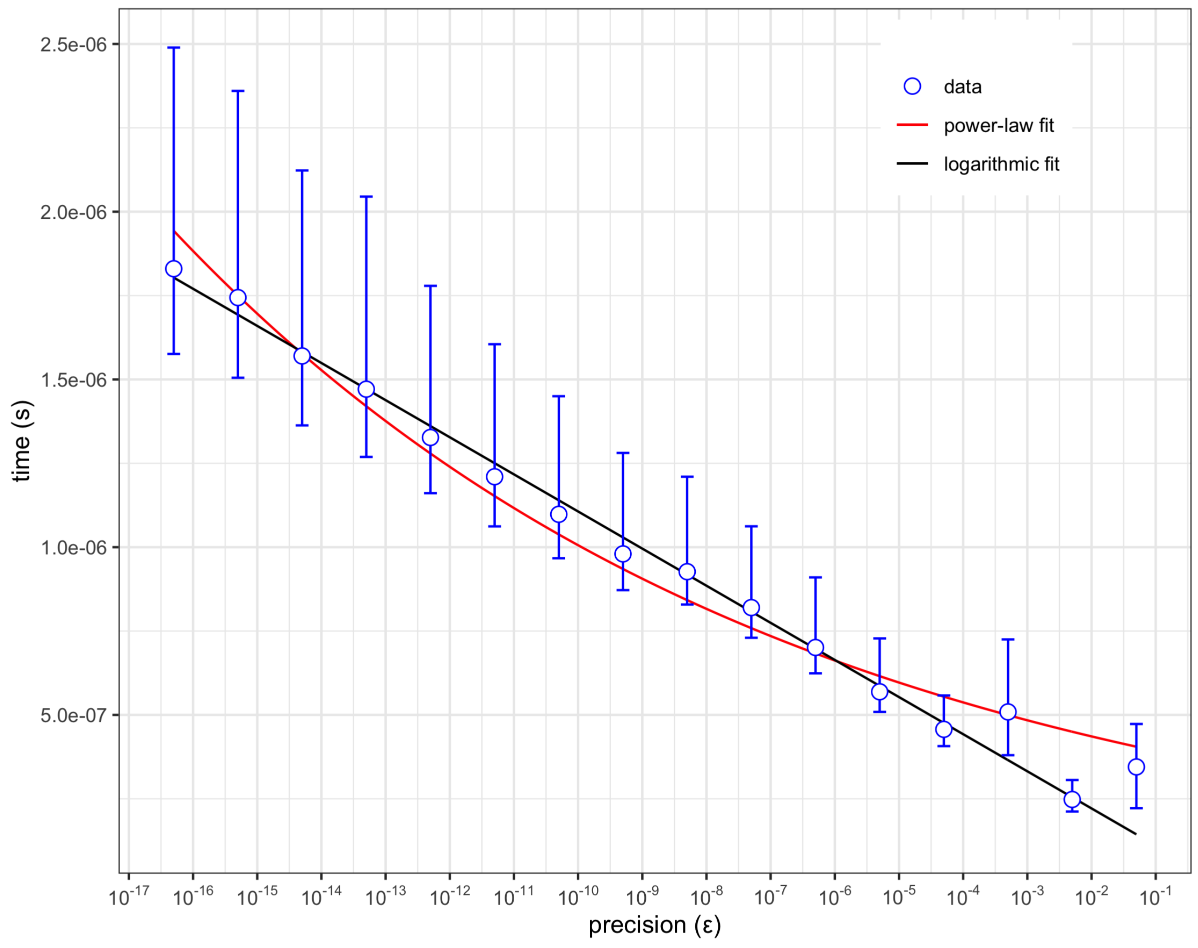
- MacBook Air 13 (1,8 GHz Dual-Core Intel Core i5 CPU, 8 GB 1600 MHz DDR3 RAM and macOS Catalina 10.15.5 OS)
- Java 15.0.2 and R 4.1.1

The method we used to obtain our data was the following:
We implemented an algorithm using the mediant approximation approach using Java. This algorithm took as input parameters the value of x and ϵ. It then separated the decimal and integer part of x and calculated the diophantine approximation of the decimal part. This was done by initializing two variables L and R to 0 and 1 respectively. We then repeatedly calculated the mediant M of L and R updating the values of L or R in each iteration with M so that $L < x < R$. This was done until the distance between M and x was smaller that ϵ. In other words, until M is a good enough approximation of x. Finally the rational number M (which is the diophantine approximation of the decimal part of x) was added to the integer part of x, reduced into its simplest form and returned. Note that by adding the integer part of x, M is now the diophantine approximation of x.

To test our algorithm we created 100,000 random floating point numbers and calculated their diophantine approximation using values for ϵ varying between numbers with 1 to 16 digits (values between 10^{-1} to 10^{-16}). The time taken for each of these approximations to be calculated was recorded and all of this data was anaylzed using R and studied.

3 Empirical Results

Using R we plotted the following graph showing how the time the algorithm takes to be executed varies with the precision of ϵ (number of digits of ϵ).



R approximated our data to a power-law fit and a logarithmic fit. Simply put, if we call our independent variable (precision of ϵ) simply ϵ, it plotted the functions of the type $a\epsilon^b$ (polynomial) and $a\log(\epsilon)$ (logarithmic) that best fitted our data. R also gave us estimated values for this parameters. If our data was to follow a polynomial pattern of growth the closest polynomial to it would be $3.538^{-7}\epsilon^{-0.04538}$. If, on the other hand, our algorithm followed a logarithmic pattern of growth that estimated logarithmic funcion would be $-4.805^{-8}\log(\epsilon)$. In order to estimate the order of growth of our algorithm we can actually discard the constants (a) as it won't affect its order of growth (which is what we are looking for).

4 Discussion

We are interested in estimating the order of growth of the computational cost of our algorithm when the precision of ϵ becomes arbitararily small. Lets call this order of growth $t(\epsilon)$ where ϵ is the precision of ϵ.

In the above graph we see the pattern that an algorithm with a polynomial order of growth would follow (red line). This algorithm would have an order $O(\epsilon^b)$, $b \in \mathbb{N}$. Taking a look at the pattern that our algorithm follows (blue circles) we can see that for a precision of ϵ of between 7 and 13 digits the time taken for our algorithm to be executed is larger than the time taken for an algorithm with a polynomial order of growth. However as the precision of ϵ becomes smaller (>16 digits) we can see that our algorithm takes less time than the power-law fit for our algorithm. Extrapolating these results we can conclude that $t(\epsilon) \in O(\epsilon^b)$.

If we now look at how an algorithm with a logarithmic order of growth ($O(log(\epsilon))$) would behave and compare it to our data we can see that our algorithm behaves in a very similar way for all values of the precision of ϵ. We can therefore conclude that the order of growth of our algorithm is logarithmic, $t(\epsilon) \in \Theta(\log(\epsilon))$.

It should be taken into account that this is the complexity of our algorithm in the average case. Normally we would be interested in finding the complexity in the worst case which for the mediant approximation approach is $O(\frac{1}{\epsilon})$. There are other methods which have a lower computational cost such as the bipartition approach which in the worst-case has an order of growth of $O(\log(\frac{1}{\epsilon}))$. However this method does not calculate the best diophantine approximation.

In conclusion the average complexity of the mediant approach to the diophantine approximation problem is $O(log(\epsilon))$.