A Computational Approach for Estimating Croatia's Gini Coefficient using Lagrange Interpolation Method for Lorenz Curve Approximation

Zagrebačka Škola Ekonomije i Managementa
Student: Mariana Ávalos Arce
Professor: Bruno Klajn, Ph.D.

Numerical Methods, Spring 2020

Zagreb, Croatia

1 Introduction

Income

Income is a monetary receipt that an individual/household earns during a certain period of time, usually a year. It is constituted by **labor income**, such as salaries, and **capital income**, such as earnings from ownership of financial capital or land. **Disposable income** is the result of the addition of transfers and substraction of taxes to an individual's gross income.

Inequality of Income

Income distribution shows how income is spread among a society, usually a country's population. In order to express this income distribution, a nation's population is divided into households. All households must be ordered, starting with the poorest and finishing with the richest household in terms of income. Then, all households are divided into **income groups**. Statistic offices commonly divide the ordered population into five or ten groups. Each income group must contain the same amount of households, where the first decile represents the poorest 10% of population and the last decile represents the richest 10% percent. Finally, for each income group the share of income with respect to total national income (GDP) is calculated.

In a perfectly equal distribution of income, each group earns the same share of national income, say 10%. This is a hypothetical and ideal case called **Absolute Equality**. The other extreme is when the last decile earns 100% of total national income, and the remaining 90% of households earn none. This case is called **Absolute Inequality**. A nation's actual inequality occurs somewhere between these two cases, which is the **Lorenz Curve**.

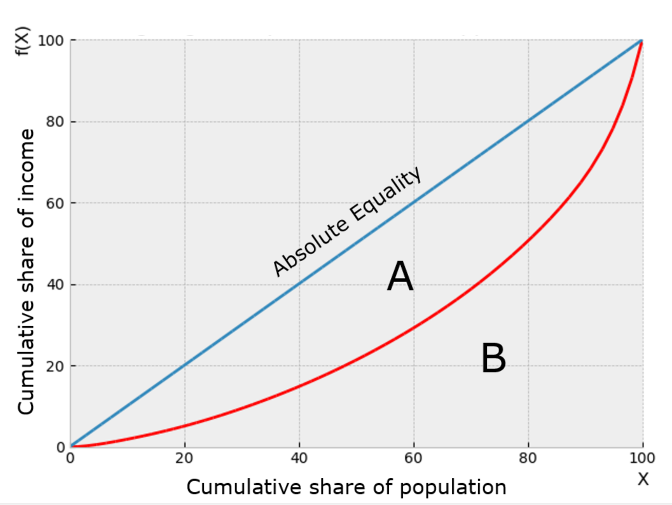


Figure 1: Lorenz Curve diagram

If we take the cumulative share of income (y axis) for each cumulative share of population (x axis), **Lorenz Curve** is plotted (red line). This curve is a representation of the actual distribution of income in a country. **Inequality** is then the deviation of the Lorenz curve from the Perfect Equality line. The bigger the area enclosed between the Equality and Lorenz curve (A), the higher inequality there will be.

Gini Coefficient

The **Gini Coefficient** is the numerical value that functions as a measure of inequality. Taking into account Figure 1, the Gini coefficient is calculated as:

$$GINI = \frac{A}{A+B} \tag{1}$$

This coefficient's value goes from 0 to 1, where 0 is **perfect equality** and 1 is **perfect inequality**. The ideal for a country is to have the lowest Gini coefficient as possible, assuring income equality. Interestingly, I noticed this coefficient can also be expressed using functions and integrals, as it involves curves and areas enclosed by said curves.

$$A = \int_0^{100} (x - L(x))dx \tag{2}$$

$$A + B = \int_0^{100} xdx \tag{3}$$

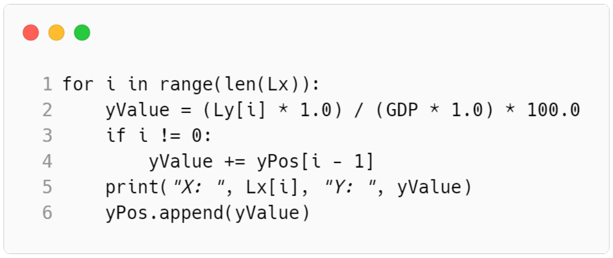
Thus, the purpose of this project is to estimate Croatia's Gini Coefficient in 2018 (0.293) using **Lagrange Interpolation Method** with the Lorenz curve data points, making use of the analytical view of the coefficient. The language chosen is Python 3.6.2 for its analytical expression's manipulation with Sympy library.

2 Generating Lorenz curve points

The first step to estimate Gini Coefficient requires data for Croatia's distribution of income among the income groups. *Eurostat* provides statistics for Croatia in 2018 as the latest [1], given that this coefficient does not change significantly each year. Data was provided in deciles as the following:

Income Distribution Croatia 2018	
Income group	Income (€)
1st decile	2,894
2nd decile	4,052
3rd decile	4,984
4th decile	5,876
5th decile	6,659
6th decile	7,620
7th decile	8,711
8th decile	10,152
9th decile	12,423
10th decile	17,874
Total (GDP)	81,245

The program reads the data from a csv file for easier testing, and stores the **income column** in a list. For each value, the cumulative ratio of the income over total GDP is calculated to generate the Lorenz curve points, following the algorithm shown below.



```python
for i in range(len(Lx)):
    yValue = (Ly[i] * 1.0) / (GDP * 1.0) * 100.0
    if i != 0:
        yValue += yPos[i - 1]
    print("X: ", Lx[i], "Y: ", yValue)
    yPos.append(yValue)
```

Figure 2: Y axis data calculation

The resulting coordinates are the data points with which the Lorenz curve is plotted:

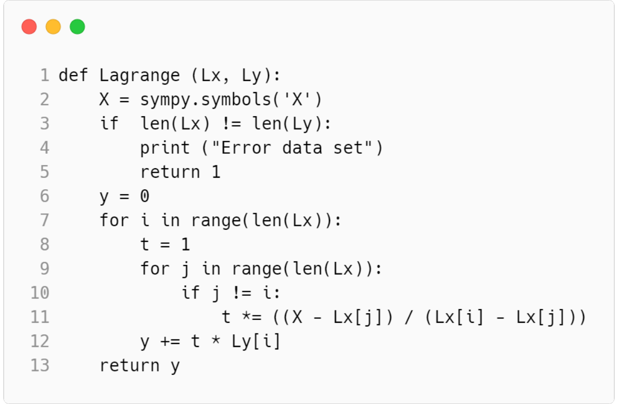
Lorenz curve Croatia 2018	
X	Y
0.0	0.0
10.0	3.5620653578681765
20.0	8.549449196873653
30.0	14.683980552649393
40.0	21.916425626192378
50.0	30.112622315219397
60.0	39.491661025293865
70.0	50.21355160317558
80.0	62.709089790140936
90.0	77.9998769155025
100.0	100.0

3 Lagrange Interpolation Method

Lagrange Interpolation Method is an approach to find a polynomial approximation of n-1 degree that connects or interpolates the n given number of points where no x values are equal [4], and which is expressed as the following:

$$L(x) = \sum_{j=1}^{n} P_j(x), \text{ where } P_j(x) = y_j \prod_{k=1, k\neq j}^{n} \frac{(x - x_k)}{(x_j - x_k)}$$

Given this expression and using Python's Sympy library, the algorithm for the calculation of a Lagrange polynomial approximation in terms of x can be coded for a given set of n points as the parameter of the function.



```python
def Lagrange (Lx, Ly):
    X = sympy.symbols('X')
    if  len(Lx) != len(Ly):
        print ("Error data set")
        return 1
    y = 0
    for i in range(len(Lx)):
        t = 1
        for j in range(len(Lx)):
            if j != i:
                t *= ((X - Lx[j]) / (Lx[i] - Lx[j]))
        y += t * Ly[i]
    return y
```

Figure 3: Function that computes a Lagrange polynomial

As mentioned before, the input parameter for the above function would be a **set of n points**, which in this case will be the set of coordinates that plot Croatia's Lorenz curve. After the list of points is sent to the function call, the resulting polynomial approximation is the following:

$$\begin{aligned}L(x) =&2.92888733459338 \times 10^{-16}x^{10} - 1.47066776487513 \times 10^{-13}x^{9} \\ &+ 3.17698630863945 \times 10^{-11}x^{8} - 3.85505917173149 \times 10^{-9}x^{7} \\ &+ 2.87809556086541 \times 10^{-7}x^{6} - 1.3608014410573 \times 10^{-5}x^{4} \\ &+ 0.000404140485432608x^{3} - 0.00722530516190734x^{2} \\ &+ 0.0764988150674863x + 0.0204593786968275\end{aligned} \tag{4}$$

This polynomial was plotted graphically using Python's Matplotlib library, producing the Lorenz curve (red line) diagram shown below:

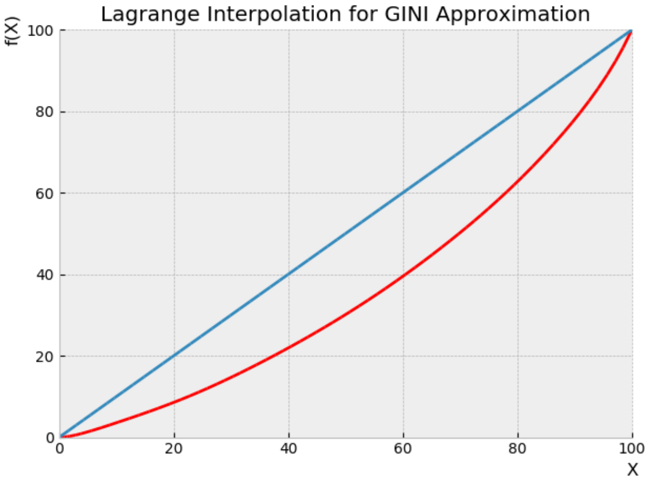


Figure 4: Lorenz curve Croatia 2018

4 Integration Approaches

4.1 Sympy Integration Module

The previous diagram included also the Absolute Equality line, f(x) = x, so as to give a better idea of the dimensions of Croatia's equality. Numerically speaking, the measure of inequality is done by the calculation of the Gini Coefficient, as mentioned before. Therefore, the integral in (2) can be calculated now that the function L(x) was determined, resulting in the following enclosed area:

$$A = \int_0^{100} (x - L(x))dx = 1431.52827736660 \tag{5}$$

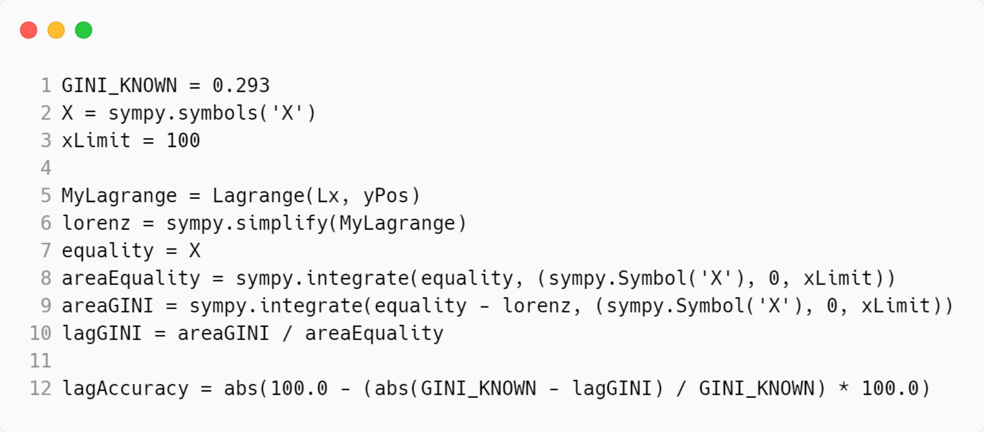
And knowing the equality area is this,

$$A + B = \int_0^{100} xdx = 5000 \tag{6}$$

The Gini Coefficient is estimated now as:

$$GINI = \frac{A}{A+B} = 0.286305655473319 \tag{7}$$

The above numerical approximations were done with Python commands, making use of Sympy's `integrate()` to avoid approximating a 10-degree polynomial integration by hand, which would be a tiresome procedure and prone to making mistakes. The Absolute Equality case is represented by f(x) = x, and the integral is a definite integral from 0 to 100, given the fact that x and y axis represent percentages. The code below shows how our custom `Lagrange()` and Sympy's `integrate()` modules are used to get the final Gini approximation.



```
GINI_KNOWN = 0.293
X = sympy.symbols('X')
xLimit = 100

MyLagrange = Lagrange(Lx, yPos)
lorenz = sympy.simplify(MyLagrange)
equality = X
areaEquality = sympy.integrate(equality, (sympy.Symbol('X'), 0, xLimit))
areaGINI = sympy.integrate(equality - lorenz, (sympy.Symbol('X'), 0, xLimit))
lagGINI = areaGINI / areaEquality

lagAccuracy = abs(100.0 - (abs(GINI_KNOWN - lagGINI) / GINI_KNOWN) * 100.0)
```

Figure 5: Integrating Lagrange resulting polynomial

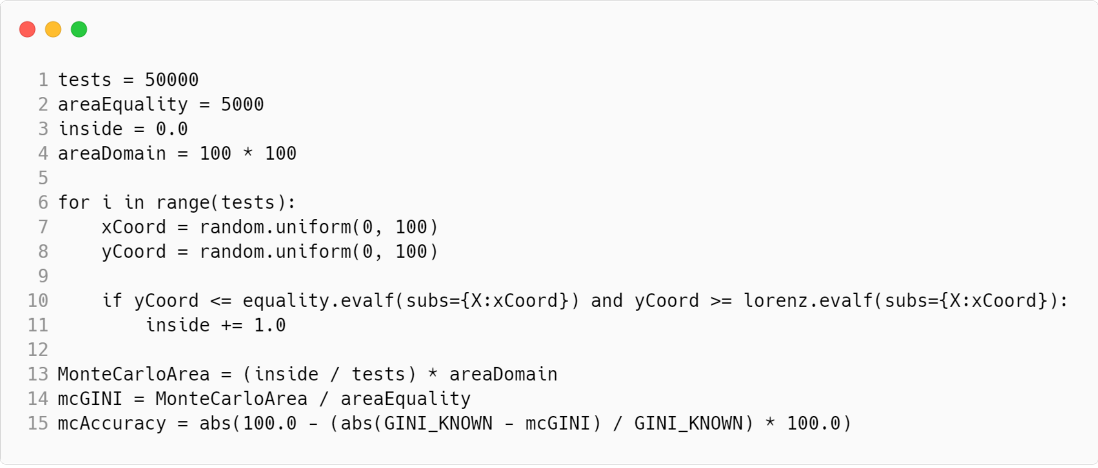
Thus, the estimated Gini has an accuracy of 97.72% when is compared to the known Gini for that year (0.293), after 1.825127 seconds of computation time.

4.2 Monte Carlo Simulation

Monte Carlo Method is a computational approach to obtain numerical approximations using *random sampling*. The exact algorithm when implementing Monte Carlo depends on the problem, but these are the essential steps of the method [2]:

1. Define a domain for possible random sample.
2. Generate the random sample using a Probability Distribution within the domain.
3. Aggregate the results.

Therefore, the next step is to implement a simple Monte Carlo simulation to approximate numerically the integral in (2), which refers to the enclosed area. Determining the **domain** is pretty evident, because the smallest squared box that contains area A goes from 0 to 100 both in x and y values as in $\{x, y | 0 \leq x \leq 100, 0 \leq y \leq 100\}$. The next step is to **generate the random sample** within the domain, for which a Uniform Distribution (all sample points are equally likely) will be used to generate the random coordinates. Each of the coordinates will get tested whether or not it is located *inside* the enclosed area A. The amount of times this happens will get recorded to get the probability of a domain point being inside A. The code below shows a very simplistic approach for this.



```
tests = 50000
areaEquality = 5000
inside = 0.0
areaDomain = 100 * 100

for i in range(tests):
    xCoord = random.uniform(0, 100)
    yCoord = random.uniform(0, 100)

    if yCoord <= equality.evalf(subs={X:xCoord}) and yCoord >= lorenz.evalf(subs={X:xCoord}):
        inside += 1.0

MonteCarloArea = (inside / tests) * areaDomain
mcGINI = MonteCarloArea / areaEquality
mcAccuracy = abs(100.0 - (abs(GINI_KNOWN - mcGINI) / GINI_KNOWN) * 100.0)
```

Figure 6: Integrating polynomial using Monte Carlo Simulation

With the record of how many times within the total tests did the point fall into enclosed area A we obtain the **probability of a point inside the enclosed area A**, which can be expressed as:

$$\text{Prob. of point inside A} = \frac{\text{Points inside}}{\text{Total generated points}}, \tag{8}$$

and therefore assuming:

$$\frac{\text{Points inside}}{\text{Total generated points}} = \frac{\text{Enclosed Area A}}{\text{Area of Domain Box}}, \tag{9}$$

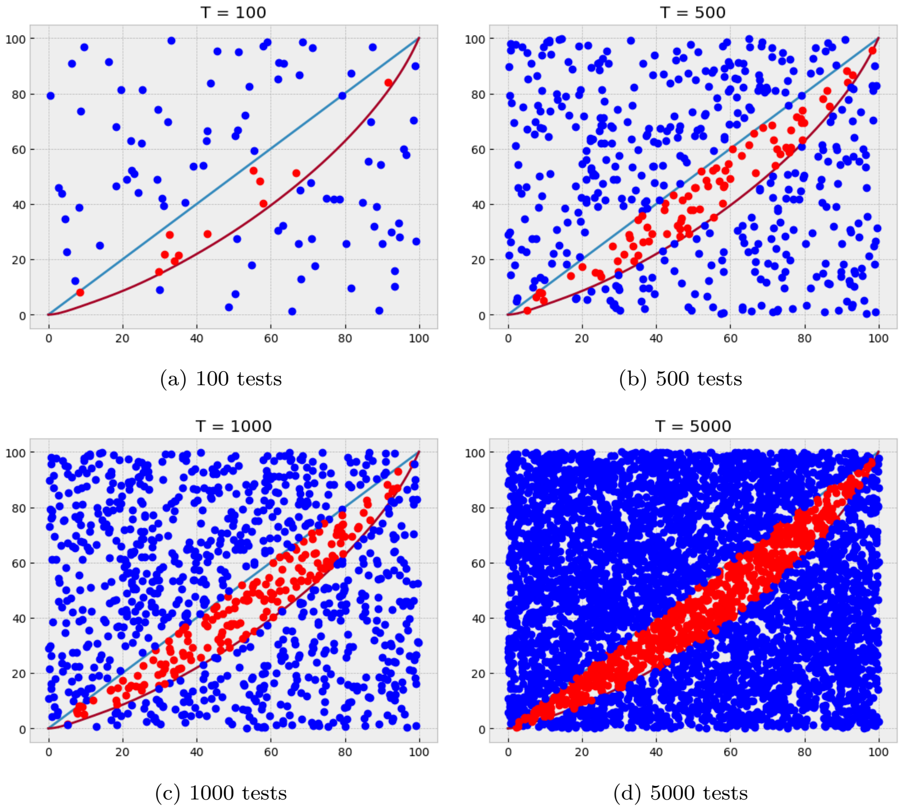
which rearranged with Eq. (2) considered gives:

$$\int_0^{100} (x - L(x))dx = \frac{\text{Points inside}}{\text{Total generated points}} \times \text{Area of Domain Box} \tag{10}$$

The equation in (10) is performed in lines 13 and 14 of Figure 6. This Monte Carlo approach took 39.7810149 seconds to calculate the integral and Gini Coefficient as:

$$\begin{aligned} A &= \int_0^{100} (x - L(x))dx = 1434.4 \\ GINI &= \frac{A}{A+B} = 0.286880000000000, \end{aligned} \tag{11}$$

Where A + B is a known constant of 5000 in all cases. The Monte Carlo Simulation done in Figure 6 takes 50 thousand tests of coordinates one by one, resulting in a Gini estimate with 97.91% of accuracy after 39.7810149 seconds. Every increment in the amount of cases will mean an increase in accuracy and time lapse, and it is also important to note that Monte Carlo is a **non-deterministic algorithm**, meaning the outcome may vary slightly between each test. A diagram of how it works is presented below.



(a) 100 tests

(b) 500 tests

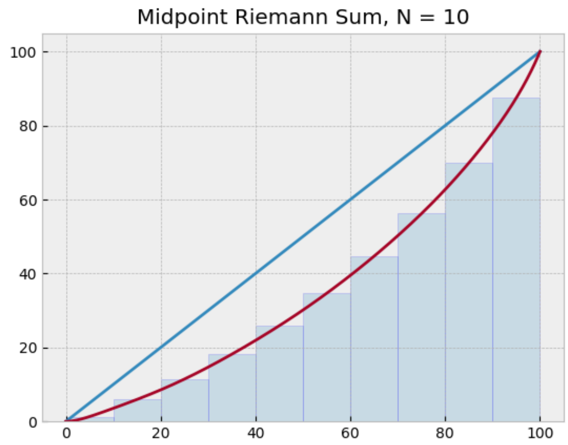
(c) 1000 tests

(d) 5000 tests

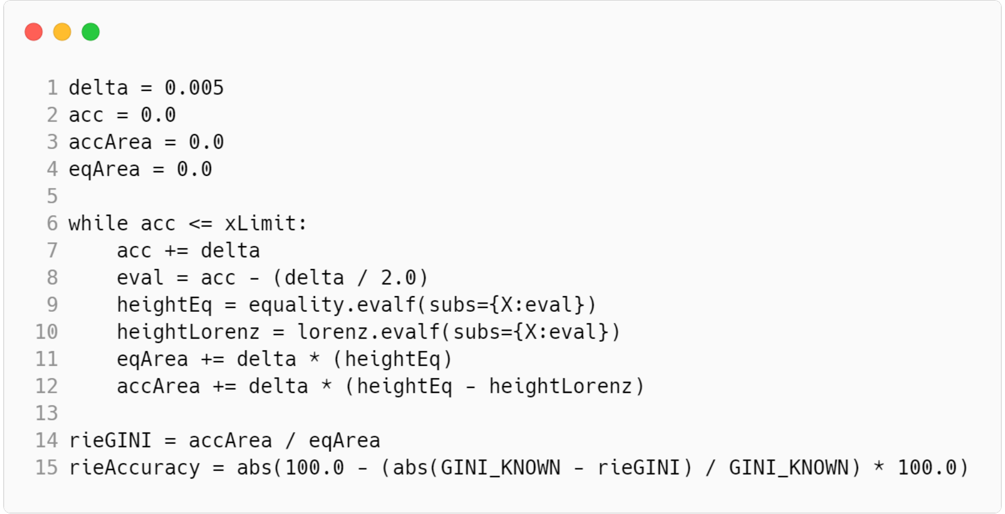
Figure 7: Simple Monte Carlo Simulation

4.3 Riemann Sum

Another way to numerically approximate the integral in (2) involves **Riemann Sum**, which is a finite sum that calculates areas under curves, or definite integrals. It consists in dividing the area in regular shapes, say rectangles, so small that their area is similar to the area in the integral in a closed interval [a, b] [5], which in our case is from 0 to 100. For example, the Riemann Sum for area (B) under the Lorenz curve does something similar to the diagram:



Where delta means the tiny base length of each of the rectangles. For its implementation, the enclosed area (A) was divided into rectangles with 0.005 of base length, for better precision. The algorithm in python was the following:



```
delta = 0.005
acc = 0.0
accArea = 0.0
eqArea = 0.0

while acc <= xLimit:
    acc += delta
    eval = acc - (delta / 2.0)
    heightEq = equality.evalf(subs={X:eval})
    heightLorenz = lorenz.evalf(subs={X:eval})
    eqArea += delta * (heightEq)
    accArea += delta * (heightEq - heightLorenz)

rieGINI = accArea / eqArea
rieAccuracy = abs(100.0 - (abs(GINI_KNOWN - rieGINI) / GINI_KNOWN) * 100.0)
```

Figure 8: Integrating polynomial using Riemann Sum

The resulting integral approximation was:

$$\begin{aligned} A &= \int_0^{100} (x - L(x))dx = 1431.52825497456 \\ GINI &= \frac{A}{A+B} = 0.286277022576955, \end{aligned} \tag{12}$$

This meant an accuracy of 97.71% to the known Gini, performed in 41.3300952 seconds. Overall, the estimated Gini has an accuracy of 98% when is compared to the known Gini for that year (0.293) using either of the three methods. Monte Carlo method seemed more accurate that the other two, but it is quite slow when compared to the 2 seconds that Sympy's `integrate()` module takes for the calculation. Nevertheless, implementing both Monte Carlo and Riemann showed the integral approximation to be quite fair for situations when Python is not available.

Coming back to Sympy's implementation: for further conclusions, the Lorenz curve for Mexico was also plotted with the program (red) using INEGI (Mexico's statistical office) data in 2018 [3] for the same calculation of the set points, alongside a curve joining each of these points with a straight line (gray). The results of Croatia (left) and Mexico (right) are shown below, giving a Gini of 0.286 and 0.435, respectively.

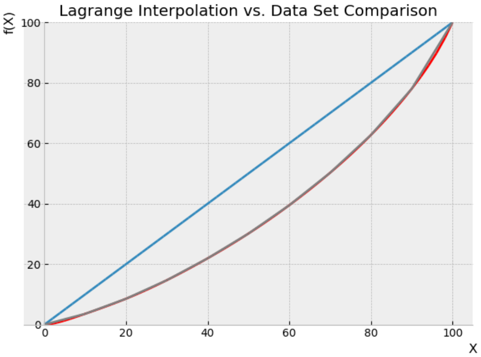


Figure 9: Approximation in Croatia

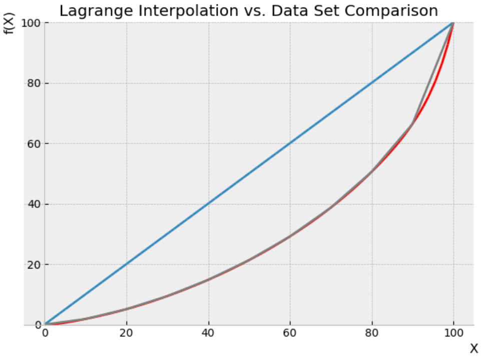


Figure 10: Approximation in Mexico

5 Observations

After observing the differences between Croatia's and Mexico's approximated polynomial for their respective Lorenz curves, one can see that Croatia's red curve is closer to the original data set curve, given that Mexico's Lorenz curve is steeper towards the end of the last decile of population, making the Lagrange approximation differ more and more as it gets more vertical, or the slope becomes larger. Professor Klajn suggests experimenting with Padé Approximation for a better result when we have steeper curve sections.

This means that the *smaller* the evaluated *derivative* of the Lorenz curve function (slope of the tangent at each point), the *higher accuracy* will be achieved in the resulting Lagrange polynomial. Countries with a smaller level of inequality (Gini closer to zero) will have their Gini approximation through Lagrange be closer to reality with higher accuracy.

When it comes to the numerical approximation for the definite integral covering the Gini area, the conclusion is that, if a built-in module is not available to integrate, a very precise but slow option is to implement a Monte Carlo Simulation.

6 Acknowledgements

I acknowledge the useful and patient lectures that Professor Bruno Klajn gave us during the course, as well as the freedom for the seminar's topic. Lectures on Distribution of Income by Anja Tkalčević and Josipa Šegota were especially useful, as well as their patience with my further doubts. My stay in Zagreb definitely taught me how patient a person can be.

References

[1] Eurostat. *Distribution of income by quantiles - EU-SILC and ECHP surveys*. 2019. URL: https://appsso.eurostat.ec.europa.eu/nui/show.do?dataset=ilc_di01&lang=en.

[2] Random Maths Inc. *The Monte Carlo Method*. URL: https://www.youtube.com/watch?v=q6gJ2T0NSwM.

[3] INEGI. *El INEGI da a conocer los resultados de la encuesta nacional de ingresos y gastos de los hogares (ENIGH) 2018*. 2019. URL: https://www.inegi.org.mx/contenidos/saladeprensa/boletines/2019/EstSociodemo/enigh2019_07.pdf.

[4] Eric Weisstein. *Lagrange Interpolating Polynomial*. URL: https://mathworld.wolfram.com/LagrangeInterpolatingPolynomial.html.

[5] Eric Weisstein. *Riemann Sum*. URL: https://mathworld.wolfram.com/RiemannSum.html.