Computation and Analysis of Basel's Problem

Project for Applied Mathematics Course MA101

Under the Esteemed guidance of Dr. Dinesh Udar

Department of Applied Mathematics

Delhi Technological University

Bawana Road. Delhi –110042

December -2020

submitted towards the partial fulfillment of

the requirement for the award of the degree of

Bachelor of Technology

Submitted by

Ansh Anurag

2K20/B2/71

Abhay

2K20/B2/11

Certificate

This is certified to be the Bonafide work of Ansh Anurag(2K20/B2/71) and Abhay(2K20/B2/11) in the course MA101 class during the academic year 2020-2021.

(Dr. Dinesh Udar)

DECLARATION

We hereby certify that the work, which is presented in the Project, entitled, **Computation and analysis of Basel's Problem** in fulfilment of the requirement for the award of the Degree of Bachelor of Technology in Computer Engineering and submitted to the Department of Applied Mathematics, Delhi Technological University, Delhi is an authentic record of my/our own, carried out under the supervision of Dr. Dinesh Udar.

The work presented in this report has not been submitted and not under consideration for the award for any other course/degree of this or any other Institute/University.

TABLE OF CONTENT

Abstract

The given project is based on the concept of convergence of the Basel Problem i.e., the infinite summation of the inverse of n squared, where n varies from 1 to infinity.

$$1+\frac{1}{4}+\frac{1}{9}+\cdots=\frac{\pi^2}{6}.$$

The project will be focusing on solving the problem by the intuitive method (the first ever solution to the problem), using trigonometric functions. We will be approximating the value using a computer program based on iteration.

Finally, we will visualize the Asymptotic Graph of the value to understand convergence and finally apply it in a computational analysis to calculate the value of pi and check error.

Understanding the Problem

Basel Problem

The Basel problem is a problem in mathematical analysis with relevance to number theory, first posed by Pietro Mengoli in 1650 and solved by Leonhard Euler in 1734, and read on 5 December 1735 in *The Saint Petersburg Academy of Sciences*. Since the problem had withstood the attacks of the leading mathematicians of the day, Euler's solution brought him immediate fame when he was twenty-eight. Euler generalized the problem considerably, and his ideas were taken up years later by Bernhard Riemann in his seminal 1859 paper "On the Number of Primes Less Than A Given Magnitude", in which he defined his zeta function and proved its basic properties. The problem is named after Basel, hometown of Euler as well as of the Bernoulli family who unsuccessfully attacked the problem.

The Basel problem asks for the precise summation of the reciprocals of the squares of the natural numbers, i.e. The sum of the series is approximately equal to 1.644934. The Basel problem asks for the *exact* sum of this series (in closed form), as well as a proof that this sum is correct. Euler found the exact sum to be $\pi^2/6$ and announced this discovery in 1735. His arguments were based on manipulations that were not justified at the time, although he was later proven correct. He produced a truly rigorous proof in 1741.

Reimann Zeta function and Basel’s problem

The Riemann zeta function $\zeta(s)$ is one of the most significant functions in mathematics because of its relationship to the distribution of the prime numbers. The zeta function is defined for any complex number*S* with real part greater than 1 by the following formula:

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}.$$

Taking *s* = 2, we see that *ζ*(2) is equal to the sum of the reciprocals of the squares of all positive integers:

$$\zeta(2) = \sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \cdots = \frac{\pi^2}{6} \approx 1.644934.$$

Proving the Convergence

Convergence can be proven by the integral test, or by the following inequality:

$$\begin{aligned} \sum_{n=1}^{N} \frac{1}{n^2} &< 1 + \sum_{n=2}^{N} \frac{1}{n(n-1)} \\ &= 1 + \sum_{n=2}^{N} \left(\frac{1}{n-1} - \frac{1}{n} \right) \\ &= 1 + 1 - \frac{1}{N} \xrightarrow{N\to\infty} 2. \end{aligned}$$

This gives us the upper bound 2, and because the infinite sum contains no negative terms, it must converge to a value strictly between 0 and 2. It can be shown that *ζ*(*s*) has a simple expression in terms of the Bernoulli numbers whenever *s* is a positive even integer. With *s* = 2*n*:

$$\zeta(2n) = \frac{(2\pi)^{2n}(-1)^{n+1}B_{2n}}{2 \cdot (2n)!}.$$

Euler’s Proof

Euler's original derivation of the value $\pi^2/6$ essentially extended observations about finite polynomials and assumed that these same properties hold true for infinite series.

Of course, Euler's original reasoning requires justification (100 years later, Karl Weierstrass proved that Euler's representation of the sine function as an infinite product is valid, by the Weierstrass factorization theorem), but even without justification, by simply obtaining the correct value, he was able to verify it numerically against partial sums of the series. The agreement he observed gave him sufficient confidence to announce his result to the mathematical community.

To follow Euler's argument, recall the Taylor series expansion of the sine function

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

Dividing through by *x*, we have

$$\frac{\sin x}{x} = 1 - \frac{x^2}{3!} + \frac{x^4}{5!} - \frac{x^6}{7!} + \cdots$$

Using the Weierstrass factorization theorem, it can also be shown that the left-hand side is the product of linear factors given by its roots, just as we do for finite polynomials (which Euler assumed as a heuristic for expanding an infinite degree polynomial in terms of its roots, but in fact is not always true for general

$$\begin{aligned}\frac{\sin x}{x} &= \left(1-\frac{x}{\pi}\right)\left(1+\frac{x}{\pi}\right)\left(1-\frac{x}{2\pi}\right)\left(1+\frac{x}{2\pi}\right)\left(1-\frac{x}{3\pi}\right)\left(1+\frac{x}{3\pi}\right)\cdots \\ &= \left(1-\frac{x^2}{\pi^2}\right)\left(1-\frac{x^2}{4\pi^2}\right)\left(1-\frac{x^2}{9\pi^2}\right)\cdots\end{aligned}$$

If we formally multiply out this product and collect all the x^2 terms (we are allowed to do so because of Newton's identities), we see by induction that the x^2 coefficient of sin *x*/*x* is

$$-\left(\frac{1}{\pi^2}+\frac{1}{4\pi^2}+\frac{1}{9\pi^2}+\cdots\right) = -\frac{1}{\pi^2}\sum_{n=1}^{\infty}\frac{1}{n^2}.$$

But from the original infinite series expansion of sin *x*/*x*, the coefficient of x^2 is $-1/3! = -1/6$. These two coefficients must be equal; thus,

$$-\frac{1}{6} = -\frac{1}{\pi^2}\sum_{n=1}^{\infty}\frac{1}{n^2}.$$

Multiplying both sides of this equation by $-\pi^2$ gives the sum of the reciprocals of the positive square integers.

$$\sum_{n=1}^{\infty}\frac{1}{n^2} = \frac{\pi^2}{6}.$$

Coding Basel’s Problem using Python

Basic Source Code

```
import math

N=input(int())

total = 0

for j in range(1,N):

     total += (1/j**2)

print(".\t", total)
```

Improving the Code

Step 1:

Create the *basel.py* program that will take one positive integer argument on the command-line and add up the reciprocals of the squares of all numbers from 1 to that positive integer, and print out 1) that positive integer, 2) the sum, 3) the value of pi*pi/6, and 4) the difference between them. If you are given the value 2 as the command-line argument, you'll add up 1/(1*1) + 1/(2*2), which is equal to 1 + .25 = 1.25. You can get the value of *pi* by importing the math library, and its value is *math.pi*.

For example:

```
C:/temp> python basel.py 2
2 1.25 1.6449340668482264 0.3949340668482264
```

Remember that your program should be able to handle any (reasonable) positive integer. But, take a look at the difference and observe it diminish as one tries larger numbers.

Step 2:

We will still work with *basel.py* program. But we are supposed to modify it now. Instead of only one positive argument on the command-line, we can take as many as the user wishes to type.

For example:

```
c:/temp> python basel.py 2 5 17
2 1.25 1.6449340668482264 0.3949340668482264
5 1.4636111111111112 1.6449340668482264 0.1813229557371152
17 1.587806741057444 1.6449340668482264 0.0571273257907825
```

However, in addition, we are needed to first check all the values on the command-line that we receive and if there is a problem anywhere, then print out how to use this program instead. For instance, if the user doesn't give you any numbers at all, or if any one of the arguments is not a positive integer, then we shall not print out any values.

Instead, we can print something similar to the following:

```
c:/temp> python basel.py 3 18 fred 9
Correct usage: python basel.py num [num ...]
Must have one or more positive integers
```

A string method called *s.isdigit()* which returns *True* if the string in the variable *s* is a non-negative integer, and *False* otherwise.

For instance: *"34".isdigit()* returns *True* but *"-6".isdigit()* returns *False*, and *"f5".isdigit()* also returns *False*. Basically, this method simply checks if that every character in the string is a digit.

Step 3:

In this stage, we will modify the program to accommodate another argument: the name of a file to write your results to. It will be the first command-line argument. Thus, instead of printing out the results, we will write them to the requested file.

```
c:/temp> python basel.py fred.txt 2 5 17
c:/temp> type fred.txt
2 1.25 1.6449340668482264 0.3949340668482264
5 1.4636111111111112 1.6449340668482264 0.1813229557371152
17 1.587806741057444 1.6449340668482264 0.0571273257907825
```

Step 4:

Now, we will have the same calling sequence: a filename followed by any number of positive integers. However, we will write the HTML of a webpage into the requested file, instead of just ordinary text. We will be displaying a title as well as a table of values.

For instance:

```
c:/temp> python basel.py BaseProblemTable.html 2 5 17
```

The Basel Problem Calculations

N	Sum	pi*pi/6	Difference
2	1.25	1.6449340668482264	0.3949340668482264
5	1.4636111111111112	1.6449340668482264	0.1813229557371152
17	1.587806741057444	1.6449340668482264	0.0571273257907825

Data Visualizing of the Asymptotic Result

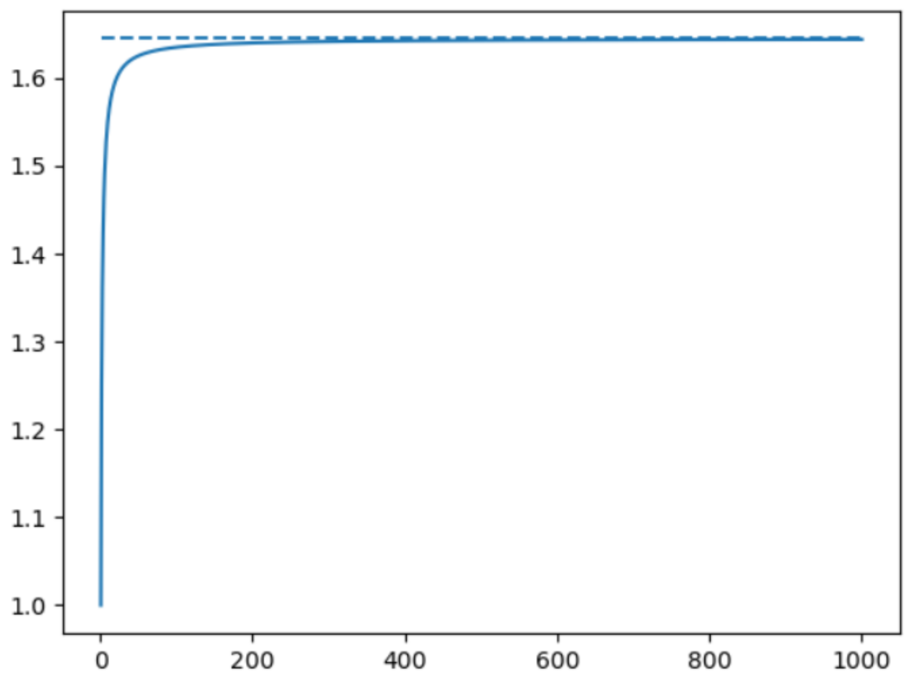
The code cell below allows the user to choose values N and s and plots the values of the partial sums *is* for $n=1,...,N$ and compares them with the limits $\zeta(s):=\sum n=1/n^s$ using the Riemann zeta function from the SciPy package.

Source Code

```
import numpy as np

import matplotlib.pyplot as plt

from scipy.special import zeta#, bernoulli

# Set your value for n and s here!

n, s = 1000, 2

def plot_inverse_power_partial_sums(n, s):

    fig = plt.figure()

    ax = fig.gca()

    # Create an array of integers and compute the corresponding summands and
partial sums

    x = np.arange(1., n+1)

    summands = np.reciprocal(np.power(x, s))

    partial_sums = np.cumsum(summands)

    # Plot the partial sums and the reference line

    ax.hlines(zeta(s), x[0], x[-1], linestyles='dashed')

    ax.plot(x, partial_sums)

    plt.show()

plot_inverse_power_partial_sums(n, s)
```

Output

From the visualized output we observe that the graph tends to a value which is approximately equal to 1.6449 as we increase the number of iterations.


1.6
1.5
1.4
1.3
1.2
1.1
1.0
0
200
400
600
800
1000

Application of the Computations in Basel Problem

A short python script to compute an approximation of PI via the probability of two random numbers being co-prime and the basel problem.

Currently 256^3 Iterations and random numbers generated between 0 and 256^2.

It uses the standard python libraries:

- random
- fractions
- math
- sys (sys.maxsize to use the maximum integer size)

Source Code

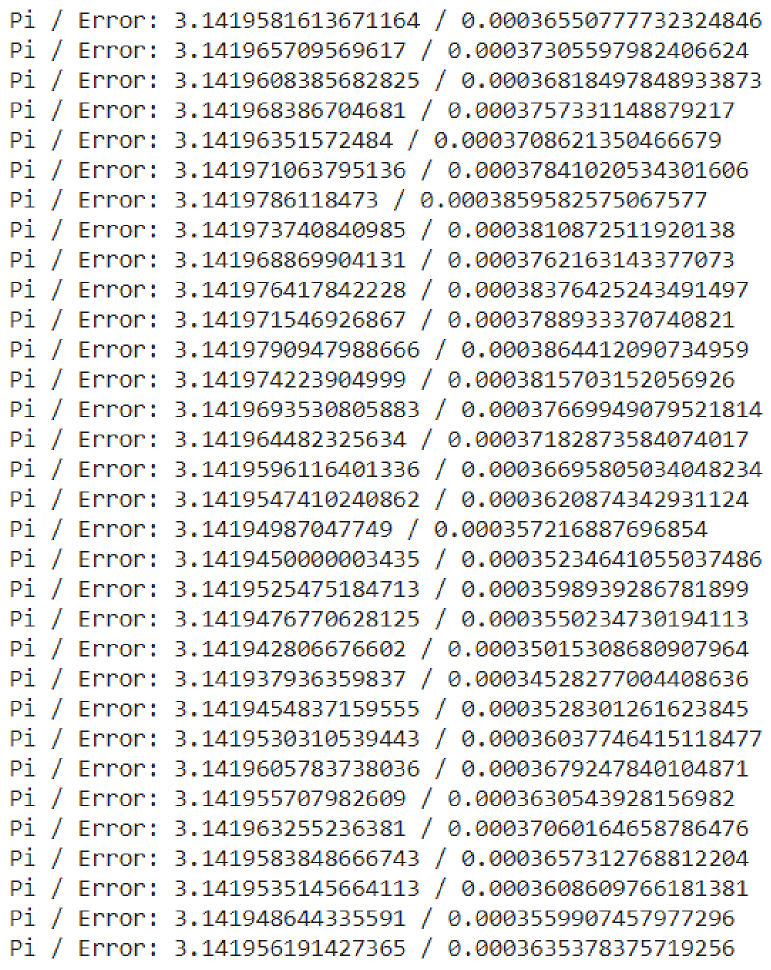
```
import random

import fractions

import math

import sys




max_random = (256**2)

max_iterations = (256**3)



def get_random_pair():

    a, b = random.randint(0, max_random), random.randint(0, max_random)

    return a, b



def get_pi_from_probability(count, iter):

    c = count / iter

    pi = math.sqrt(6/c)

    return pi
```

```
def approx_pi():

    count = 0

    lowest_error = 256

    best_pi = 0



    for i in range(1, max_iterations):

        if (fractions.gcd(*get_random_pair()) == 1):

            count += 1

        if (count > 0):

            d = get_pi_from_probability(count, i)

            err = abs(d - math.pi)

            print("[" + str(i) + "/" + str(max_iterations) + "] Pi / Error: "
+ str(d) + " / " + str(err))

            if (0 < err < lowest_error):

                lowest_error = err

                best_pi = d



    accuracy = (lowest_error / math.pi) * 100

    print("--------------------------------------------------------------")

    print("Aproximated PI: " + str(best_pi))

    print("------ math.pi: " + str(math.pi))



    print("\nAccuracy: " + str(accuracy) + " % | Lowest Error: " +
str(lowest_error))

    print("--------------------------------------------------------------")
```

```
approx_pi()
```

Output

Hence, we calculate the value of pi using the result of Basel Problem.



```
Pi / Error: 3.1419581613671164 / 0.00036550777732324846
Pi / Error: 3.141965709569617 / 0.00037305597982406624
Pi / Error: 3.1419608385682825 / 0.00036818497848933873
Pi / Error: 3.141968386704681 / 0.0003757331148879217
Pi / Error: 3.14196351572484 / 0.0003708621350466679
Pi / Error: 3.141971063795136 / 0.00037841020534301606
Pi / Error: 3.1419786118473 / 0.0003859582575067577
Pi / Error: 3.141973740840985 / 0.0003810872511920138
Pi / Error: 3.141968869904131 / 0.0003762163143377073
Pi / Error: 3.141976417842228 / 0.00038376425243491497
Pi / Error: 3.141971546926867 / 0.0003788933370740821
Pi / Error: 3.1419790947988666 / 0.00038644120907349590
Pi / Error: 3.141974223904999 / 0.0003815703152056926
Pi / Error: 3.1419693530805883 / 0.00037669949079521814
Pi / Error: 3.141964482325634 / 0.00037182873584074017
Pi / Error: 3.1419596116401336 / 0.00036695805034048234
Pi / Error: 3.1419547410240862 / 0.00036208743429311240
Pi / Error: 3.1419498704774 9 / 0.0003572168876968540
Pi / Error: 3.1419450000003435 / 0.00035234641055037486
Pi / Error: 3.1419525475184713 / 0.00035989392867818990
Pi / Error: 3.1419476770628125 / 0.0003550234730194113
Pi / Error: 3.141942806676602 / 0.0003501530868090796 4
Pi / Error: 3.141937936359837 / 0.0003452827700440863 6
Pi / Error: 3.1419454837159555 / 0.0003528301261623845
Pi / Error: 3.1419530310539443 / 0.00036037746415118477
Pi / Error: 3.1419605783738036 / 0.0003679247840104871
Pi / Error: 3.141955707982609 / 0.0003630543928156982
Pi / Error: 3.141963255236381 / 0.0003706016465878647 6
Pi / Error: 3.1419583848666743 / 0.0003657312768812204
Pi / Error: 3.1419535145664113 / 0.0003608609766181381
Pi / Error: 3.141948644335591 / 0.0003559907457977296
Pi / Error: 3.141956191427365 / 0.0003635378375719256
```

Conclusion

Hence, we were able to understand the convergence by the virtue of mathematics as well as programming. We visualized the convergence of the Basel Problem and were able to estimate the value of Pi as well.