Ministerul Educaţiei şi Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică

REPORT

Laboratory work no.6
Nth digit of PI

Elaborated:
st. gr. FAF-213 Gutu Dinu

Verified:
asist. univ. Fiștic Cristofor

Chişinău – 2023

Table of Contents

ALGORITHM ANALYSIS

Objective

1. Analysis and study of the algorithms.
2. Empirical analysis of the aforementioned algorithms.

Tasks

1. Implement the algorithms listed above in a programming language
2. Establish the properties of the input data against which the analysis is performed
3. Choose metrics for comparing algorithms
4. Perform empirical analysis of the proposed algorithms
5. Make a graphical presentation of the data obtained
6. Make a conclusion on the work done.

Theoretical notes

Empirical analysis is an alternative to mathematical analysis in evaluating algorithm complexity. It can be used to gain insight into the complexity class of an algorithm, compare the efficiency of different algorithms for solving similar problems, evaluate the performance of different implementations of the same algorithm, or examine the efficiency of an algorithm on a specific computer. The typical steps in empirical analysis include defining the purpose of the analysis, choosing a metric for efficiency such as number of operations or execution time, determining the properties of the input data, implementing the algorithm in a programming language, generating test data, running the program on the test data, and analyzing the results. The choice of efficiency metric depends on the purpose of the analysis, with number of operations being appropriate for complexity class evaluation and execution time being more relevant for implementation performance. After the program is run, the results are recorded and synthesized through calculations of statistics or by plotting a graph of problem size against efficiency measure.

Introduction

Algorithms for computing the Nth digit of Pi have been a subject of interest for mathematicians and computer scientists for centuries. Pi is an irrational number, and as such, it has an infinite number of decimal places that continue infinitely without repeating. Therefore, finding the Nth digit of Pi is a challenging problem that requires sophisticated algorithms and advanced computational techniques.

There are several methods for computing the digits of Pi, each with its own advantages and limitations. The Bailey–Borwein–Plouffe (BBP) formula is a popular approach for computing the Nth digit of Pi, as it allows for rapid calculation of individual digits without requiring the previous digits. Spigot algorithms provide a different approach, generating the digits of Pi in a streaming fashion. Monte Carlo methods use probabilistic techniques to approximate Pi by estimating the ratio of points inside and outside of a circle. In this laboratory work I will be covering the following ones: Chudnovsky algorithm, Gauss-Legendre method and Machin formula.

Despite the considerable research and progress made in computing the digits of Pi, finding the Nth digit of Pi remains an active area of research and a challenging problem. Researchers continue to seek new algorithms and methods that offer improved efficiency, accuracy, and applicability to different scenarios. The discovery of more efficient algorithms for computing the digits of Pi has the potential to impact a wide range of fields, including numerical analysis, cryptography, and scientific computing.

Comparison Metric

The comparison metric for this laboratory work will be considered the time of execution of each algorithm (T(n)) with attention to their accuracy as well.

Input Format

As input the algorithms will be taking a list of values that would represent the digits to search for in PI.

IMPLEMENTATION

Chudnovsky algorithm

The Chudnovsky algorithm is a fast method for calculating the digits of π, based on Ramanujan's π formulae. It was published by the Chudnovsky brothers in 1988.
It was used in the world record calculations of 2.7 trillion digits of π in December 2009, 10 trillion digits in October 2011, 22.4 trillion digits in November 2016, 31.4 trillion digits in September 2018–January 2019, 50 trillion digits on January 29, 2020, 62.8 trillion digits on August 14, 2021, and 100 trillion digits on March 21, 2022.

Algorithm Description:

The algorithm is based on the negated Heegner number $d = -163$, the *j*-function $j\left(\frac{1+i\sqrt{163}}{2}\right) = -640320^3$, and on the following rapidly convergent generalized hypergeometric series:[2]

$$\frac{1}{\pi} = 12\sum_{q=0}^{\infty}\frac{(-1)^q(6q)!(545140134q+13591409)}{(3q)!(q!)^3(640320)^{3q+\frac{3}{2}}}$$

A detailed proof of this formula can be found here:[10]

For a high performance iterative implementation, this can be simplified to

$$\frac{(640320)^{\frac{3}{2}}}{12\pi} = \frac{426880\sqrt{10005}}{\pi} = \sum_{q=0}^{\infty}\frac{(6q)!(545140134q+13591409)}{(3q)!(q!)^3(-262537412640768000)^q}$$

There are 3 big integer terms (the multinomial term M_q, the linear term L_q, and the exponential term X_q) that make up the series and π equals the constant C divided by the sum of the series, as below:

$$\pi = C\left(\sum_{q=0}^{\infty}\frac{M_q \cdot L_q}{X_q}\right)^{-1}, \text{ where:}$$

$$C = 426880\sqrt{10005},$$
$$M_q = \frac{(6q)!}{(3q)!(q!)^3},$$
$$L_q = 545140134q + 13591409,$$
$$X_q = (-262537412640768000)^q.$$

The terms M_q, L_q, and X_q satisfy the following recurrences and can be computed as such:

$$L_{q+1} = L_q + 545140134 \qquad \text{where } L_0 = 13591409$$
$$X_{q+1} = X_q \cdot (-262537412640768000) \qquad \text{where } X_0 = 1$$
$$M_{q+1} = M_q \cdot \left(\frac{(12q+2)(12q+6)(12q+10)}{(q+1)^3}\right) \text{ where } M_0 = 1$$

The computation of M_q can be further optimized by introducing an additional term K_q as follows:

$$K_{q+1} = K_q + 12 \qquad \text{where } K_0 = -6$$
$$M_{q+1} = M_q \cdot \left(\frac{K_{q+1}^3 - 16K_{q+1}}{(q+1)^3}\right) \text{ where } M_0 = 1$$

Note that

$$e^{\pi\sqrt{163}} \approx 640320^3 + 743.99999999999925\ldots \text{ and}$$
$$640320^3 = 262537412640768000$$
$$545140134 = 163 \cdot 127 \cdot 19 \cdot 11 \cdot 7 \cdot 3^2 \cdot 2$$
$$13591409 = 13 \cdot 1045493$$

Implementation

```
def chudnovsky_pi(one):
    """
    Calculate pi using Chudnovsky's series

    This calculates it in fixed point, using the value for one passed in
    """
    k = 1
    a_k = one
    a_sum = one
    b_sum = 0
    C = 640320
    C3_OVER_24 = C**3 // 24
    while 1:
        a_k *= -(6*k-5)*(2*k-1)*(6*k-1)
        a_k //= k*k*k*C3_OVER_24
        a_sum += a_k
        b_sum += k * a_k
        k += 1
        if a_k == 0:
            break
    total = 13591409*a_sum + 545140134*b_sum
    pi_c = (426880*sqrt(10005*one, one)*one) // total
    len_c = len(str(pi_c)) - 1
    pi_c = decimal.Decimal(pi_c)
    pi_c = pi_c.scaleb(-len_c)
    return pi_c
```

Figure 1. Chudnovsky algorithm

Results

Nth PI digit algorithms	Accuracy	Time to compute PI (s)	PI Digits
Chudnovsky algorithm	0.0	4.576094099989859	100000

Figure 2. Chudnovsky computation result

Nth PI digit algorithms	Nth digit accuracy [10, 100, 500, 1000, 5000, 15000]	Actual PI digits [3, 7, 1, 8, 2, 7]
Chudnovsky algorithm	[True, True, True, True, True, True]	[3, 7, 1, 8, 2, 7]

Figure 3. Chudnovsky nth digit result

We can see that chudnovsky algorithm is also very precise

Nth PI digit algorithms	Nth digit time taken [10, 100, 500, 1000, 5000, 15000]
Chudnovsky algorithm	[1.9e-05, 2.6e-05, 0.000156, 0.000483, 0.012278, 0.094423]

Figure 4. Chudnovsky digit result

The time complexity of the Chudnovsky's series algorithm is O(n * log n^3) because it involves the multiplication of large numbers and the calculation of factorials which are computationally expensive. It also uses a second-order Newton-Raphson convergence which doubles the number of significant figures on each iteration. The algorithm uses floating-point arithmetic to make an initial guess and then uses Newton's method to refine the guess.

Graph

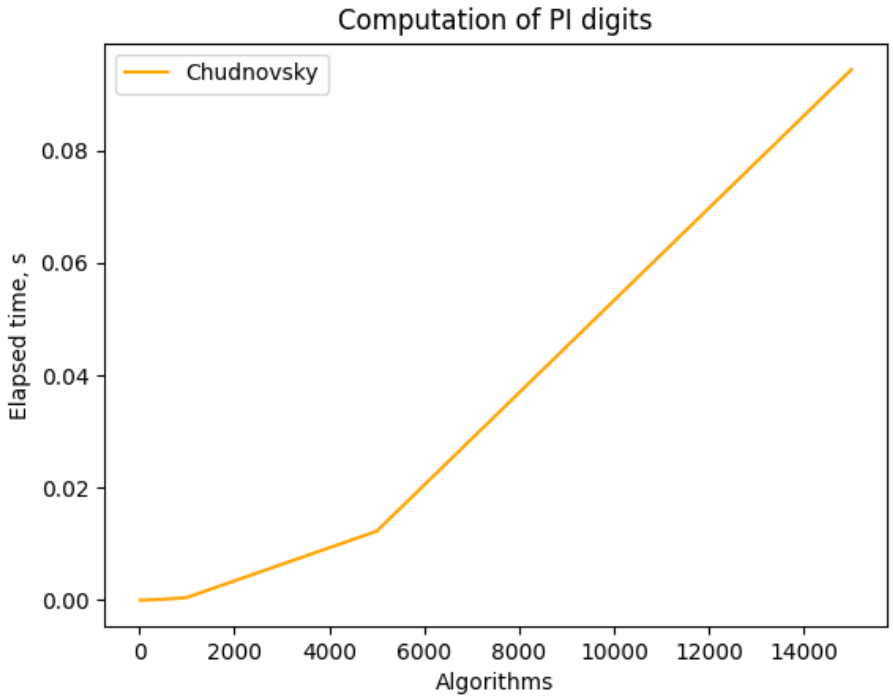

Figure 5. Chudnovsky graph results

Gauss-Legedre algorithm:

The Gauss–Legendre algorithm is an algorithm to compute the digits of π. It is notable for being rapidly convergent, with only 25 iterations producing 45 million correct digits of π. However, it has some drawbacks (for example, it is computer memory-intensive) and therefore all record-breaking calculations for many years have used other methods, almost always the Chudnovsky algorithm. For details, see Chronology of computation of π.
The method is based on the individual work of Carl Friedrich Gauss (1777–1855) and Adrien-Marie Legendre (1752–1833) combined with modern algorithms for multiplication and square roots. It repeatedly replaces two numbers by their arithmetic and geometric mean, in order to approximate their arithmetic-geometric mean.

Algorithm Description:

1. Initial value setting:

$$a_0 = 1 \qquad b_0 = \frac{1}{\sqrt{2}} \qquad t_0 = \frac{1}{4} \qquad p_0 = 1.$$

2. Repeat the following instructions until the difference of a_n and b_n is within the desired accuracy:

$$a_{n+1} = \frac{a_n + b_n}{2},$$

$$b_{n+1} = \sqrt{a_n b_n},$$

$$t_{n+1} = t_n - p_n(a_n - a_{n+1})^2,$$

$$p_{n+1} = 2p_n.$$

3. π is then approximated as:

$$\pi \approx \frac{(a_{n+1} + b_{n+1})^2}{4t_{n+1}}.$$

The first three iterations give (approximations given up to and including the first incorrect digit):

$3.140\ldots$
$3.14159264\ldots$
$3.14159265358979323 82\ldots$

The algorithm has quadratic convergence, which essentially means that the number of correct digits doubles with each iteration of the algorithm.

Implementation

```python
def gauss_legendre_pi(iteration, precision):
    getcontext().prec = precision

    # Initialize variables
    a = Decimal(1)
    b = Decimal(1) / Decimal(2).sqrt()
    t = Decimal(1) / Decimal(4)
    p = Decimal(1)

    # Compute digits of PI
    for i in range(iteration):
        a_next = (a + b) / 2
        b = (a * b).sqrt()
        t_next = t - p * (a - a_next) ** 2
        p = 2 * p
        a = a_next
        t = t_next

    # Calculate PI
    pi = (a + b) ** 2 / (4 * t)

    return pi
```

Figure 6. Gauss-Legendre algorithm

Results

Nth PI digit algorithms	Accuracy	Time to compute PI (s)	PI Digits
Gauss-Legendre algorithm	0.0	15.969558000011602	20000

Figure 7. Gauss-Legendre computation result

Nth PI digit algorithms	Nth digit accuracy [10, 100, 500, 1000, 5000, 15000]	Actual PI digits [3, 7, 1, 8, 2, 7]
Gauss-Legendre algorithm	[True, True, True, True, True, True]	[3, 7, 1, 8, 2, 7]

Figure 8. Gauss-Legendre nth digit result

We can see in the results that altough slow the Gauss-Legendre algorithm is very accurate.

Nth PI digit algorithms	Nth digit time taken [10, 100, 500, 1000, 5000, 15000]
Gauss-Legendre algorithm	[0.000171, 0.000945, 0.013067, 0.051701, 1.223037, 10.851579]

Figure 9. Gauss-Legendre digit result

The time complexity of the above algorithm is O(n^2). The reason is that each iteration requires a square root operation and a multiplication operation. The square root operation has a time complexity of O(m), where m is the number of bits used to represent the input. The multiplication operation has a time complexity of O(m^2), where m is the number of bits used to represent the input. Since both operations are performed in each iteration, the total time complexity of the algorithm is O(n^2).

Graph

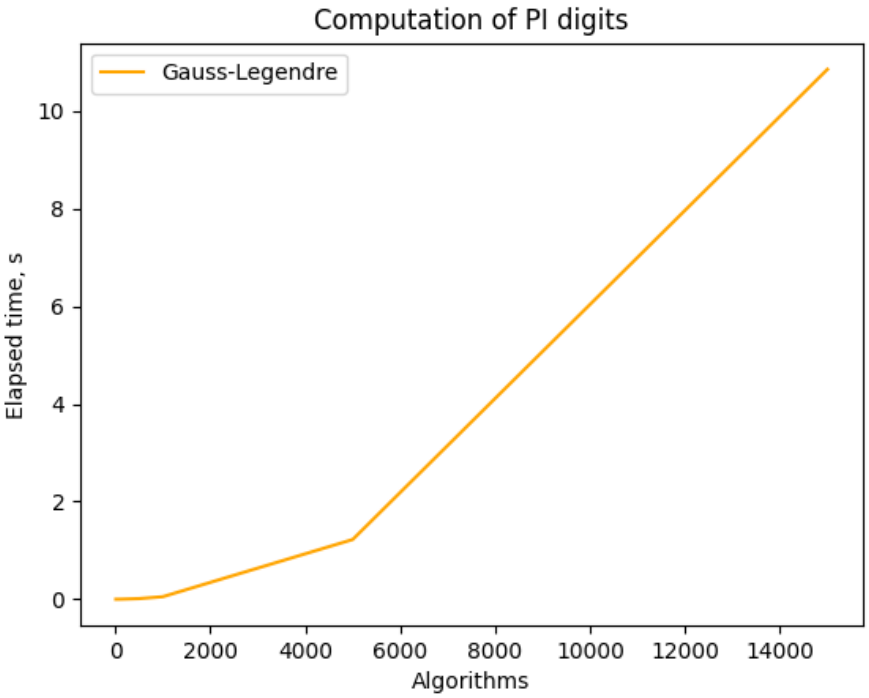

Figure 10. Gauss-Legendre graph results

Machin algorithm:

In mathematics, Machin-like formulae are a popular technique for computing π (the ratio of the circumference to the diameter of a circle) to a large number of digits.

Algorithm Description:

$$\frac{\pi}{4} = 4\arctan\frac{1}{5} - \arctan\frac{1}{239}$$

which he used to compute π to 100 decimal places.[1][2]

Machin-like formulas have the form

$$c_0\frac{\pi}{4} = \sum_{n=1}^{N} c_n \arctan\frac{a_n}{b_n}$$

where c_0 is a positive integer, c_n are signed non-zero integers, and a_n and b_n are positive integers such that $a_n < b_n$.

These formulas are used in conjunction with Gregory's series, the Taylor series expansion for arctangent:

$$\arctan x = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} x^{2n+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots$$

Implementation

```
def machin_pi(precision):
    mpmath.mp.dps = precision
    pi = 4 * (4 * mpmath.atan(1 / 5) - mpmath.atan(1 / 239))
    return pi
```

Figure 11. Machin algorithm

Results

Nth PI digit algorithms	Accuracy	Time to compute PI (s)	PI Digits
Machin formula algorithm	1.7224065202086849e-16	0.9655364000063855	100000

Figure 12. Machin computation result

Nth PI digit algorithms	Nth digit time taken [10, 100, 500, 1000, 5000, 15000]
Machin formula algorithm	[0.000168, 0.000147, 0.00041, 0.000621, 0.004799, 0.031516]

Figure 13. Machin nth digit result

The time complexity of the Machin formula is O(1) because it does not depend on the number of iterations. Although it depends on other factors, such as precision.

Nth PI digit algorithms	Nth digit accuracy [10, 100, 500, 1000, 5000, 15000]	Actual PI digits [3, 7, 1, 8, 2, 7]
Gauss-Legendre algorithm	[True, True, True, True, True, True]	[3, 7, 1, 8, 2, 7]

Figure 14. Machin digit result

We can see that it is by far the least accurate.

Graph

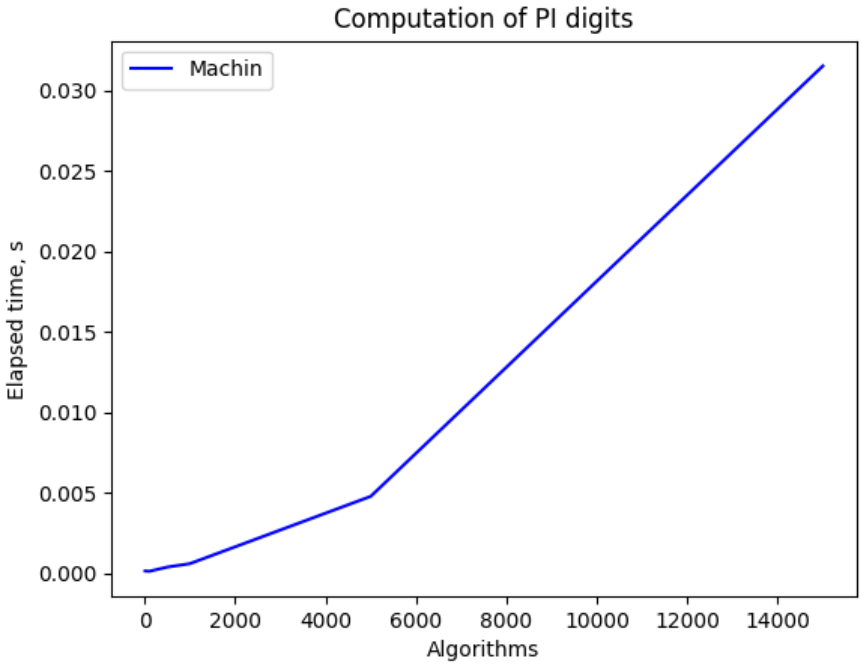

Figure 15. Machin graph results

All Algorithms

All results

Nth PI digit algorithms	Accuracy	Time to compute PI (s)	PI Digits
Chudnovsky algorithm	0.0	4.576094099989859	100000
Gauss-Legendre algorithm	0.0	15.969558000011602	20000
Machin formula algorithm	1.7224065202086849e-16	0.9655364000063855	100000

Figure 16. All computation result

It should be noted that Gauss-Legendre has only computed PI up to 30000 digits since to 100000 it was extremely slow.

Nth PI digit algorithms	Nth digit accuracy [10, 100, 500, 1000, 5000, 15000]	Actual PI digits [3, 7, 1, 8, 2, 7]
Chudnovsky algorithm	[True, True, True, True, True, True]	[3, 7, 1, 8, 2, 7]
Gauss-Legendre algorithm	[True, True, True, True, True, True]	[3, 7, 1, 8, 2, 7]
Machin formula algorithm	[True, True, False, False, False, False]	[3, 7, 6, 1, 4, 3]

Figure 17. All digit result

Nth PI digit algorithms	Nth digit time taken [10, 100, 500, 1000, 5000, 15000]
Chudnovsky algorithm	[1.9e-05, 2.6e-05, 0.000156, 0.000483, 0.012278, 0.094423]
Gauss-Legendre algorithm	[0.000171, 0.000945, 0.013067, 0.051701, 1.223037, 10.851579]
Machin formula algorithm	[0.000168, 0.000147, 0.00041, 0.000621, 0.004799, 0.031516]

Figure 18. Machin nth digit result

Graph

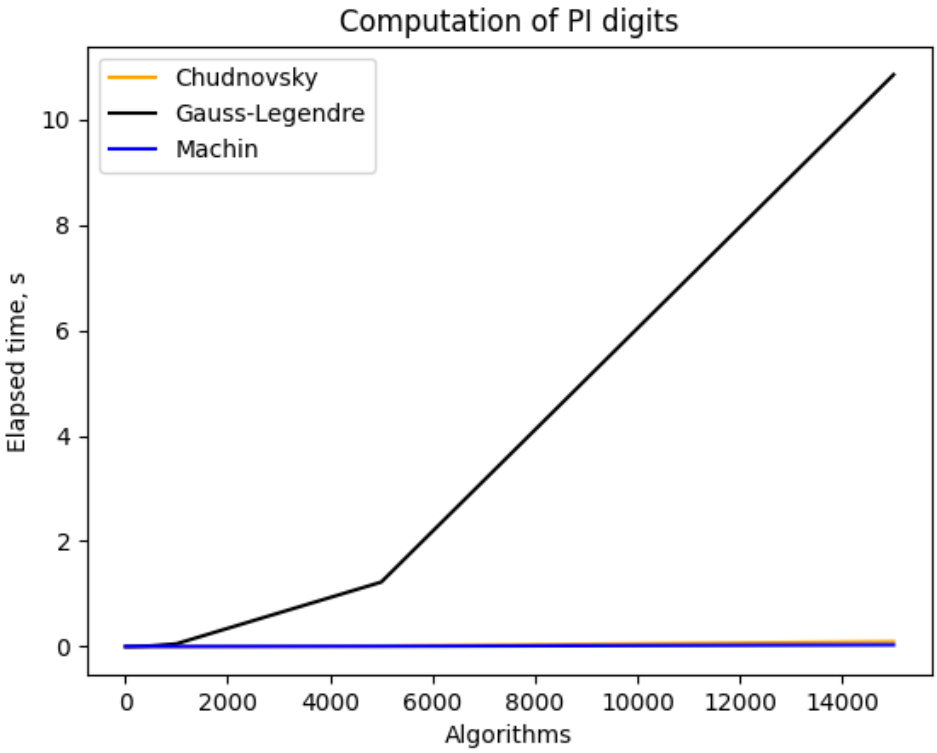

Figure 19. All computation graph

CONCLUSION

In conclusion, the Chudnovsky algorithm, Machin formula, and Gauss-Legendre algorithm are all efficient ways of calculating the value of pi, but with different trade-offs in terms of computation time and implementation complexity.

The Chudnovsky algorithm is an extremely fast algorithm, with a complexity of O(n * log n^3) and an impressive speed of convergence. It is also a more recent algorithm, developed in the 1980s, and uses a combination of a series expansion and a fast multiplication algorithm. However, the implementation of the Chudnovsky algorithm requires a high level of mathematical expertise and computational power, making it less accessible for practical purposes.

The Machin formula for pi is a relatively simple formula that was first discovered in the 18th century. It has a complexity of O(1) (it is being affected by other precision computation) and can be easily implemented using basic arithmetic operations. However, it converges more slowly than the Chudnovsky algorithm and the Gauss-Legendre algorithm, making it less suitable for high-precision calculations.

The Gauss-Legendre algorithm is an iterative algorithm that uses a series of recursive approximations to calculate pi. It has a complexity of O(n^2) and requires fewer mathematical operations than the Chudnovsky algorithm. However, it requires a significant number of iterations to converge, which can increase computation time.

Overall, the choice of algorithm for calculating pi depends on the specific application and the desired level of precision. For high-precision calculations, the Chudnovsky algorithm is the most efficient option, while the Machin formula and Gauss-Legendre algorithm are better suited for less demanding applications.

BIBLIOGRAPHY

Github: https://github.com/Grena30/APA_Labs/tree/main/Lab6