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

REPORT

Laboratory work no. 6
N-th digit of π

Elaborated:
st. gr. FAF-213 Botnari Ciprian

Verified:
asist. univ. Fiștic Cristofor

Chișinău – 2023

Table of Contents

ALGORITHM ANALYSIS

Objective

Study and analyze 3 algorithms for finding the n-th digit of π: Bailey Borwein Plouffe, Gauss Legendre and Bellard.

Tasks

1. Implement 3 algorithms for finding the n-th digit of π
2. Decide properties of input format that will be used for algorithm analysis;
3. Decide the comparison metric for the algorithms;
4. Analyze empirically the algorithms;
5. Present the results of the obtained data;
6. Deduce conclusions of the laboratory.

Theoretical notes

An alternative approach to evaluating the complexity of an algorithm is through empirical analysis, which can provide insights on the complexity class of the algorithm, comparison of efficiency between algorithms solving the same problem, comparison of different implementations of the same algorithm, and performance on specific computers.

The process of empirical analysis typically involves:

1. Establishing the purpose of the analysis
2. Choosing the efficiency metric (number of operations or execution time)
3. Defining the properties of the input data
4. Implementing the algorithm in a programming language
5. Generating input data
6. Running the program with each set of data
7. Analyzing the results

The choice of the efficiency measure depends on the purpose of the analysis. If, for example, the aim is to obtain information on the complexity class or even checking the accuracy of a theoretical estimate then it is appropriate to use the number of operations performed. But if the goal is to assess the behavior of the implementation of an algorithm then execution time is appropriate.

After the execution of the program with the test data, the results are recorded and, for the purpose of the analysis, either synthetic quantities (mean, standard deviation, etc.) are calculated or a graph with appropriate pairs of points (i.e. problem size, efficiency measure) is plotted.

Introduction

The mathematical constant π (pi) is an irrational number, which means it cannot be expressed as a finite decimal or a fraction. Its decimal representation goes on infinitely without repeating. Therefore, determining the value of the n-th digit of π is a complex task.

Since the digits of pi are not known to follow a predictable pattern, it is impossible to directly calculate the n-th digit without extensive computation. However, there have been significant efforts to calculate and discover the value of specific digits of pi using various algorithms and supercomputers.

The current record for calculating the digits of pi stands at trillions of digits. However, for practical purposes, the commonly used approximation of pi is 3.14159 or 3.14. In most everyday applications, these few decimal places are sufficient.

To compute the n-th digit of pi, there are several algorithms available, such as the Bailey–Borwein–

Plouffe (BBP) formula, the Gauss–Legendre algorithm, and the spigot algorithms. These algorithms employ mathematical techniques to calculate the digits of pi sequentially, but they are computationally intensive and require substantial time and resources to produce accurate results.

It's worth noting that the vast majority of applications do not require knowledge of the exact value of pi beyond a few decimal places. Most mathematical calculations, scientific experiments, and engineering designs utilize approximations of pi to achieve the desired level of accuracy.

Comparison Metric

The comparison metric for this laboratory work will be considered the time of execution of each algorithm T(n) for each various number.

Input Format

The algorithms will take different n-th digits of π (pi):

- 5
- 10
- 100
- 500
- 1000
- 5000

IMPLEMENTATION

All algorithms will be put into practice in Python and objectively evaluated depending on how long it takes to complete each one. The particular efficiency in report with input will vary based on the memory of the device utilized, even though the overall trend of the results may be similar to other experimental data. As determined by experimental measurement, the error margin will be a few seconds.

Bailey Borwein Plouffe

Algorithm Description:

The Bailey-Borwein-Plouffe (BBP) formula is a formula for calculating π that was discovered in 1995 by Simon Plouffe and is named after the authors of the article in which it was published, David H. Bailey, Peter Borwein, and Plouffe. The formula is:

$$\pi = \sum_{k=0}^{\infty} \frac{1}{16^k}\left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6}\right)$$

Implementation

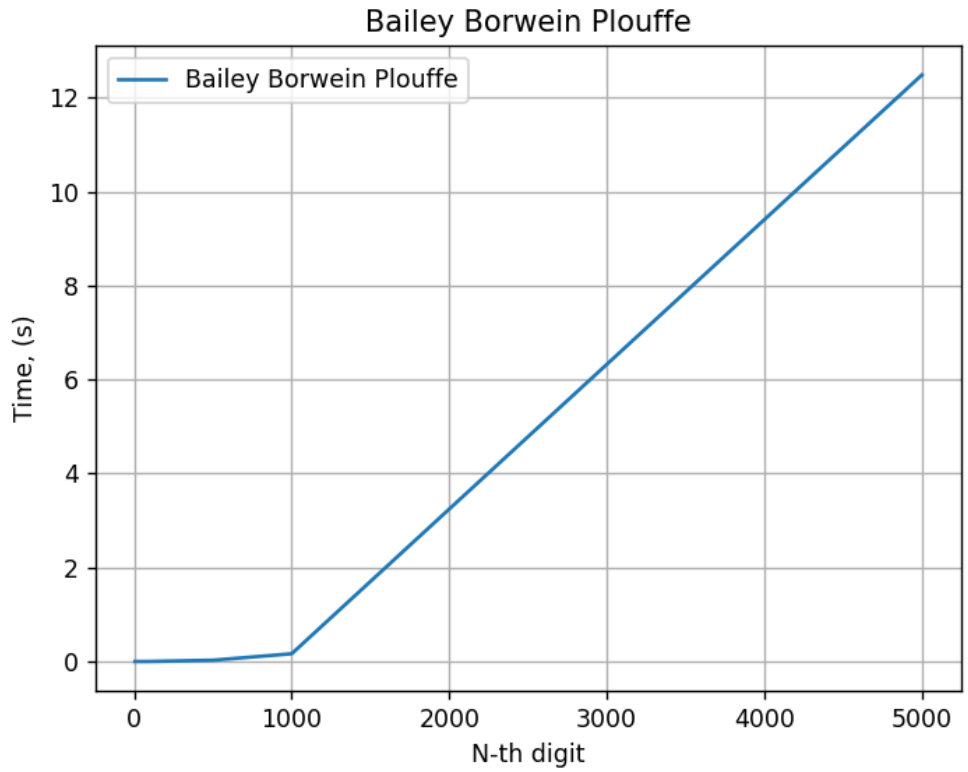
```python
def Bailey_Borwein_Plouffe(n):
    getcontext().prec = n + 10

    pi = Decimal(0)
    k = 0
    while k <= n:
        pi += (Decimal(1)/(16**k)) * ((Decimal(4)/(8*k+1)) - (Decimal(2)/(8*k+4)) - (Decimal(1)/(8*k+5)) - (Decimal(1)/(8*k+6)))
        k += 1

    return str(pi)[n]
```

Results

Algorithm / n	5	50	100	500	1000	5000
Bailey Borwein Plouffe	0.00011	0.00044	0.00113	0.02550	0.16615	12.84566
Gauss Legendre	0.00005	0.00058	0.00193	0.11580	1.03921	132.51732
Bellard	0.00006	0.00057	0.00142	0.02444	0.14489	11.32206

The time complexity for this BBB algorithm is **$O(n^2)$**, where n is the number of digits of π to be computed. This is because the algorithm uses a loop that iterates n times, and each iteration involves a modular exponentiation that takes O(n) time. Therefore, the total time is $O(n * n) = O(n^2)$.

Graph

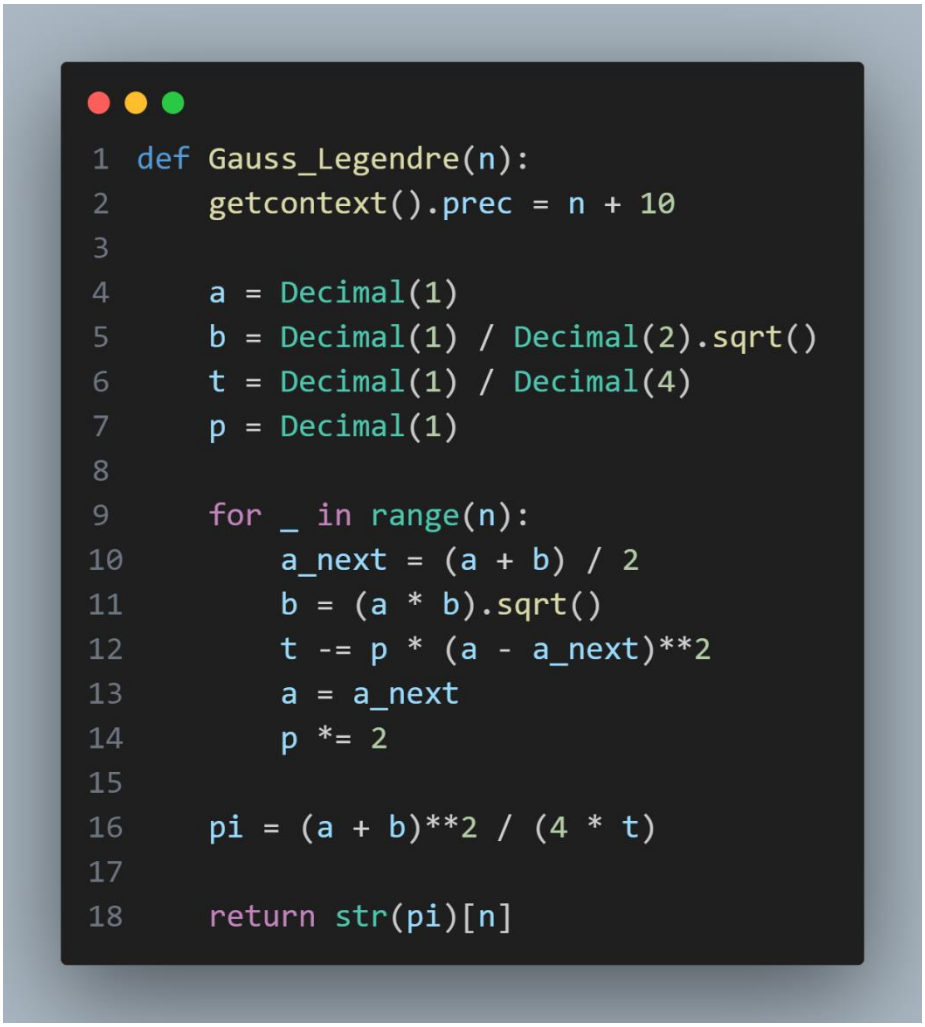


Gauss Legendre

Algorithm Description:

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.

Implementation



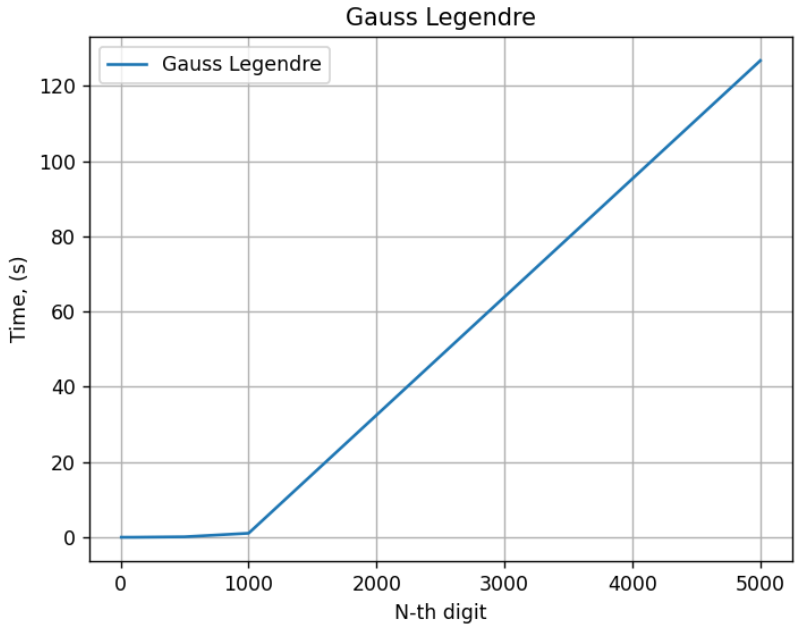
```python
def Gauss_Legendre(n):
    getcontext().prec = n + 10

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

    for _ in range(n):
        a_next = (a + b) / 2
        b = (a * b).sqrt()
        t -= p * (a - a_next)**2
        a = a_next
        p *= 2

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

    return str(pi)[n]
```

Results

Algorithm / n	5	50	100	500	1000	5000
Bailey Borwein Plouffe	0.00011	0.00044	0.00113	0.02550	0.16615	12.84566
Gauss Legendre	0.00005	0.00058	0.00193	0.11580	1.03921	132.51732
Bellard	0.00006	0.00057	0.00142	0.02444	0.14489	11.32206

The time complexity for this Gauss Legendre algorithm is **$O(n^2)$**, where n is the number of digits of π to be computed. This is because the algorithm uses a loop that iterates n times, and each iteration involves a modular exponentiation that takes O(n) time. Therefore, the total time is $O(n * n) = O(n^2)$.

Graph



Bellard

Algorithm Description:

The Bellard formula is a mathematical formula that is used to calculate the value of the mathematical constant π (pi). It was discovered by Fabrice Bellard, a French software developer and mathematician, in 1997.

$$\pi = \frac{1}{2^6} \sum_{n=0}^{\infty} \frac{(-1)^n}{2^{10n}} \left(-\frac{2^5}{4n+1} - \frac{1}{4n+3} + \frac{2^8}{10n+1} - \frac{2^6}{10n+3} - \frac{2^2}{10n+5} - \frac{2^2}{10n+7} + \frac{1}{10n+9} \right)$$

Implementation

```python
def Bellard(n):
    getcontext().prec = n + 10   # Set precision to get accurate results

    digit = Decimal(0)
    k = 0
    while k <= n:
        a = Decimal(1) / Decimal(16) ** k
        b1 = Decimal(4) / (Decimal(8) * k + Decimal(1))
        b2 = Decimal(2) / (Decimal(8) * k + Decimal(4))
        b3 = Decimal(1) / (Decimal(8) * k + Decimal(5))
        b4 = Decimal(1) / (Decimal(8) * k + Decimal(6))
        term = a * (b1 - b2 - b3 - b4)

        digit += term
        k += 1

    digit = int(digit * Decimal(10)**n) % 10

    return digit
```

Results

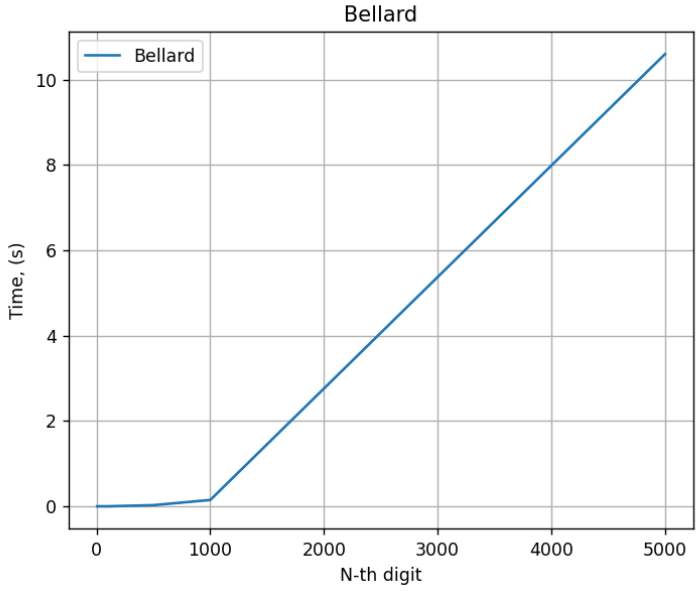
Algorithm / n	5	50	100	500	1000	5000
Bailey Borwein Plouffe	0.00011	0.00044	0.00113	0.02550	0.16615	12.84566
Gauss Legendre	0.00005	0.00058	0.00193	0.11580	1.03921	132.51732
Bellard	0.00006	0.00057	0.00142	0.02444	0.14489	11.32206

The time complexity of the given Bellard function can be analyzed as follows:

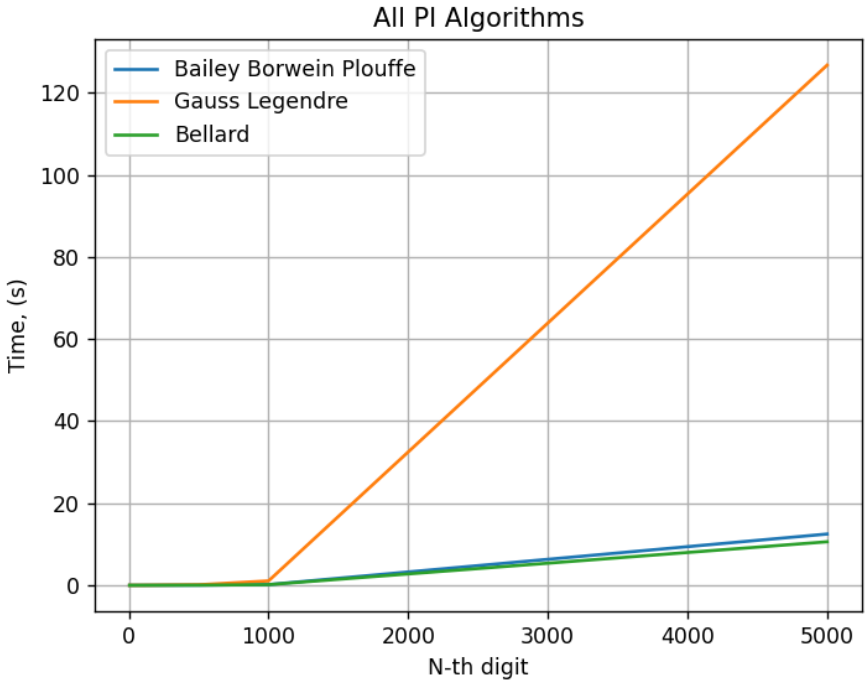
1. The loop iterates from k = 0 to k = n, performing a constant number of arithmetic operations in each iteration. Therefore, the time complexity of the loop is O(n).
2. The computation of each term within the loop involves arithmetic operations with Decimal numbers, which typically have a time complexity of O(log(N)), where N is the number of digits involved in the computation. In this case, the number of digits is limited by the precision set using **getcontext().prec = n + 10**. So the time complexity of each term computation can be approximated as O(log(n)).
3. The final operation **int(digit * Decimal(10)**n) % 10** involves exponentiation and multiplication, which can be considered to have a time complexity of O(log(n)) as well.

Considering the dominant operations, the time complexity of the given Bellard function is approximately O(n * log(n)).

Graph



Comparison



The Gauss-Legendre algorithm, the Bailey-Borwein-Plouffe (BBP) algorithm, and the Bellard formula are all methods for approximating the value of π. While Gauss-Legendre is a popular iterative algorithm, it generally performs worse than the BBP and Bellard methods for computing π. Here are a few reasons why Gauss-Legendre may be less efficient:

1. Convergence Speed: The Gauss-Legendre algorithm has a relatively slower convergence rate compared to the BBP and Bellard formulas. The convergence rate refers to how quickly the algorithm approaches the true value of π. The BBP and Bellard methods have been specifically designed to rapidly converge to the desired precision, allowing for faster computation of π.
2. Iterative Nature: The Gauss-Legendre algorithm relies on iterative calculations, repeatedly refining the approximation of π. Each iteration requires a series of arithmetic operations, which can become computationally expensive for high precision calculations. In contrast, the BBP and Bellard formulas are direct formulas that can calculate specific digits of π without the need for iterative refinement, making them more efficient for high-precision computations.
3. Complexity of Arithmetic Operations: The Gauss-Legendre algorithm involves various arithmetic operations, such as square roots, divisions, and multiplications, which can be time-consuming. In contrast, the BBP and Bellard formulas primarily involve simpler operations like exponentiation, addition, and subtraction, which can be more efficiently implemented and optimized for speed.

CONCLUSION

In conclusion, we have examined three different algorithms for approximating the value of π: the Gauss-Legendre algorithm, the Bailey-Borwein-Plouffe (BBP) algorithm, and the Bellard formula. Each algorithm offers its own advantages and considerations when it comes to efficiency and precision.

The Gauss-Legendre algorithm, although not as efficient as the BBP and Bellard methods, still holds value due to its simplicity and ease of implementation. It is suitable for scenarios where a moderate level of precision is required, and computational resources are not a limiting factor. Additionally, the iterative nature of the Gauss-Legendre algorithm allows for flexibility in refining the approximation iteratively, making it useful in situations where precision needs to be gradually increased.

On the other hand, when precision is a priority and computational efficiency is crucial, the BBP algorithm and the Bellard formula outperform the Gauss-Legendre algorithm. The BBP algorithm is particularly notable for its ability to compute individual hexadecimal digits of π, making it valuable for

applications that require specific digit extraction or when the result needs to be presented in a hexadecimal format.

The Bellard formula, derived from the BBP algorithm, offers an optimized approach for calculating π in binary or hexadecimal form. With its direct formula and rapid convergence, the Bellard formula is especially well-suited for high-precision calculations and has been used to set world records for computing π to trillions of decimal places.

In summary, consider the following guidelines when choosing between these algorithms:

1. Gauss-Legendre algorithm: Use when simplicity and ease of implementation are important, and moderate precision is sufficient. It allows for iterative refinement and flexibility in precision adjustment.
2. BBP algorithm: Utilize when specific hexadecimal digits of π are required or when hexadecimal representation is desired. It performs well for moderate to high precision and provides fast convergence.
3. Bellard formula: Employ when high precision is needed, especially for calculations in binary or hexadecimal form. It is highly efficient, has rapid convergence, and excels in computing π to a large number of decimal places.

By understanding the strengths and weaknesses of these algorithms, you can make an informed decision on which method to use based on your specific requirements and computational constraints.

GITHUB

Sufferal/Algorithms: Analysis of algorithms (github.com)