MINISTRY OF EDUCATION, CULTURE AND RESEARCH OF REPUBLIC OF MOLDOVA
TECHNICAL UNIVERSITY OF MOLDOVA
FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS
DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS

Algorithm Analysis

Laboratory work 6 : Study and empirical analysis of algorithms that determine a N decimal digit of PI.

Elaborated:
st.gr. FAF-211 Grama Alexandru
Verified:
asist.univ. Fiștic Cristofor

Chișinău, 2023

Content

Introduction

The mathematical constant (pi) has captivated the minds of mathematicians, scientists, and enthusiasts for centuries. Its irrationality, transcendence, and ubiquity in various mathematical and scientific disciplines have made it a subject of extensive exploration. Among the intriguing questions surrounding is the determination of its decimal digits, a pursuit that has attracted considerable attention and motivated the development of numerous algorithms.

In this study, we delve into the fascinating realm of algorithms that aim to calculate specific decimal digits of . These algorithms combine mathematical ingenuity, numerical methods, and computational techniques to approximate the elusive digits of this mathematical constant. While is infinitely long and its decimal representation is non-repeating, algorithms offer a way to approximate its value to an arbitrary precision.

The empirical analysis of these algorithms is crucial to understand their efficiency, accuracy, and convergence properties. By studying the performance of these algorithms, we can gain insights into their strengths, weaknesses, and computational requirements. This empirical analysis aids in selecting the most appropriate algorithm for a given precision requirement, optimizing computational resources, and identifying potential areas for algorithmic improvements.

Throughout this study, we explore a range of algorithmic approaches employed in the quest to determine N decimal digits of . From classical methods like the Machin-like formulas and series expansions, to modern techniques such as spigot algorithms and digit extraction methods, each algorithm presents a unique approach to approximating the digits of . We examine the underlying mathematical principles behind these algorithms and investigate their computational characteristics.

Furthermore, we employ empirical analysis to evaluate the performance of these algorithms. By measuring metrics such as execution time, memory usage, and accuracy, we can compare and contrast the algorithms under different scenarios. Through extensive computational experiments and numerical simulations, we aim to provide valuable insights into the behavior and efficiency of these algorithms, enabling researchers and practitioners to make informed choices.

Ultimately, this study serves as a comprehensive exploration of the study and empirical analysis of algorithms for determining N decimal digits of . By combining theoretical foundations, algorithmic innovation, and empirical investigation, we hope to shed light on the remarkable world of and contribute to the ongoing pursuit of understanding and calculating this iconic mathematical constant.

The Chudnovsky algorithm

The Chudnovsky algorithm is a remarkable algorithm developed by the Chudnovsky brothers, David and Gregory, in 1989. It is a highly efficient and rapidly converging algorithm used for the computation of (pi) to a large number of decimal places. The algorithm gained significant attention for its ability to calculate with unprecedented speed, surpassing previous algorithms in terms of efficiency.

The Chudnovsky algorithm is based on the concept of infinite series, specifically the Ramanujan series. It leverages the incredible power of series acceleration techniques to accelerate the convergence of the series and compute with remarkable efficiency. The algorithm combines the principles of mathematical analysis, number theory, and advanced computational methods to achieve its impressive results.

At the core of the Chudnovsky algorithm lies the use of the inverse of the Ramanujan series:

$1/pi = 12 * ((-1)^k * (6k)! * (13591409 + 545140134k))/((3k)! * (k!)^3 * (-640320)^(3k + 3/2))$

where k ranges from 0 to infinity, and the symbol represents summation.

To calculate using the Chudnovsky algorithm, the series is truncated after a certain number of terms. The more terms included in the calculation, the higher the precision achieved in the approximation of . However, each additional term also adds computational complexity, so there is a trade-off between precision and computational resources.

One key advantage of the Chudnovsky algorithm is its rapid convergence rate. The number of correct decimal places of roughly doubles with each iteration of the series. This remarkable property enables the algorithm to calculate billions or even trillions of decimal places of with relatively few iterations, making it highly efficient compared to alternative approaches.

In addition to the series itself, the Chudnovsky algorithm utilizes various optimization techniques to further enhance its efficiency. These optimizations include the efficient calculation of factorials, the use of binary splitting for efficient multiplication, and the utilization of multi-threading or parallel processing to exploit modern computing architectures fully.

The Chudnovsky algorithm has found applications in numerous fields, such as high-precision computations, cryptographic algorithms, and scientific research requiring precise mathematical constants. It has been used to set world records in computing the value of and has become a benchmark for evaluating the performance of computer systems and numerical libraries.

In summary, the Chudnovsky algorithm stands as a remarkable achievement in the study of algorithms for calculating . Its efficient convergence, combined with optimization techniques, enables the rapid and accurate computation of to an extraordinary number of decimal places. By pushing the boundaries of computational precision, the Chudnovsky algorithm has contributed significantly to the field of numerical computation and our understanding of this fundamental mathematical constant.

The Leibniz formula

The Leibniz formula, also known as the Leibniz series or the Gregory-Leibniz series, is a simple yet intriguing method for approximating the value of (pi). It is named after the German mathematician and philosopher Gottfried Wilhelm Leibniz, who discovered this series in the late 17th century.

The Leibniz formula for calculating is based on an alternating series. It takes advantage of the property that the arctangent function, atan(x), evaluated at certain values can be used to approximate /4. The formula is given by:

pi/4 = 1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 + ...

In this series, each term alternates between positive and negative, and the denominators increase by 2 for each subsequent term (starting from 1). By summing up more terms of this series, we can achieve a more accurate approximation of /4, and subsequently, .

To obtain an approximation for , we can multiply the sum of the series by 4:

4 * (1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 + ...)

The Leibniz formula provides a relatively straightforward way to calculate using basic arithmetic operations. However, it converges slowly, requiring a large number of terms to achieve high precision. The convergence rate of the series is quite modest, meaning that each additional term added to the calculation provides a smaller improvement in accuracy compared to other algorithms.

Despite its slow convergence, the Leibniz formula has historical significance and is often used as an introductory example in teaching and popularizing the concept of series and numerical approximations. Its simplicity and intuitive nature make it accessible for students and enthusiasts interested in exploring the world of .

While the Leibniz formula may not be the most efficient method for calculating to a high number of decimal places, it serves as a stepping stone for more advanced algorithms and series expansions. By understanding the principles behind the Leibniz formula and its alternating series structure, one can appreciate the broader concepts and techniques employed in the study of algorithms for computing .

In conclusion, the Leibniz formula offers a basic but interesting approach to approximating the value of . Its alternating series construction provides an accessible entry point into the world of numerical approximations and serves as a fundamental building block for more sophisticated algorithms. While not the most efficient method, the Leibniz formula holds historical and educational value, contributing to our understanding and appreciation of this ubiquitous mathematical constant.

The Bailey–Borwein–Plouffe (BBP) formula

The Bailey–Borwein–Plouffe (BBP) formula is a remarkable algorithm for computing the value of (pi) that was discovered by Simon Plouffe in 1995. The formula allows for the direct calculation of

individual hexadecimal (base-16) digits of without the need to compute the preceding digits. This property makes it particularly useful in applications where only specific digits of are required.

The BBP formula is based on the concept of infinite series and the digit extraction method. It can be stated as follows:

$= [(1/16^k) * (4/(8k+1) - 2/(8k+4) - 1/(8k+5) - 1/(8k+6))]$

where k ranges from 0 to infinity, and the symbol represents summation.

In this formula, each term of the series contributes to the calculation of a specific hexadecimal digit of . By summing up more terms, one can obtain increasingly accurate values for the desired digit. The BBP formula's advantage lies in its direct computation of individual digits, eliminating the need to compute the entire decimal expansion of .

The BBP formula is particularly useful in digital computing systems, where base-16 or hexadecimal representation is common. By using the formula, one can efficiently compute any desired hexadecimal digit of without having to calculate the preceding digits. This makes it suitable for applications such as computer graphics, cryptography, and digital signal processing.

It is worth noting that the BBP formula is not as widely known or commonly used as other algorithms for calculating , such as the Chudnovsky algorithm or the Machin-like formulas. However, its unique property of directly computing individual hexadecimal digits has made it a subject of interest and exploration among mathematicians and computer scientists.

The discovery of the BBP formula showcases the beauty and versatility of mathematical algorithms for computing . By leveraging infinite series and digit extraction methods, researchers like Simon Plouffe have made significant contributions to our understanding and practical applications of this fundamental mathematical constant.

Implementation

Code in python:

```
import time
import matplotlib.pyplot as plt
import numpy as np
from mpmath import mp



# Algorithm 1: Chudnovsky Algorithm
def chudnovsky_algorithm(n):
    mp.dps = n + 1
    C = 426880 * mp.sqrt(10005)
    M = 1
    L = 13591409
    X = 1
    K = 6
    S = L
    for _ in range(1, n):
        M = (K ** 3 - 16 * K) * M // K ** 3
        L += 545140134
        X *= -262537412640768000
        S += (M * L) // X
        K += 12
    return C / S



# Algorithm 2: Leibniz Formula
def leibniz_formula(n):
    pi = 0
    for i in range(n):
        pi += ((-1) ** i) / (2 * i + 1)
    return 4 * pi
```

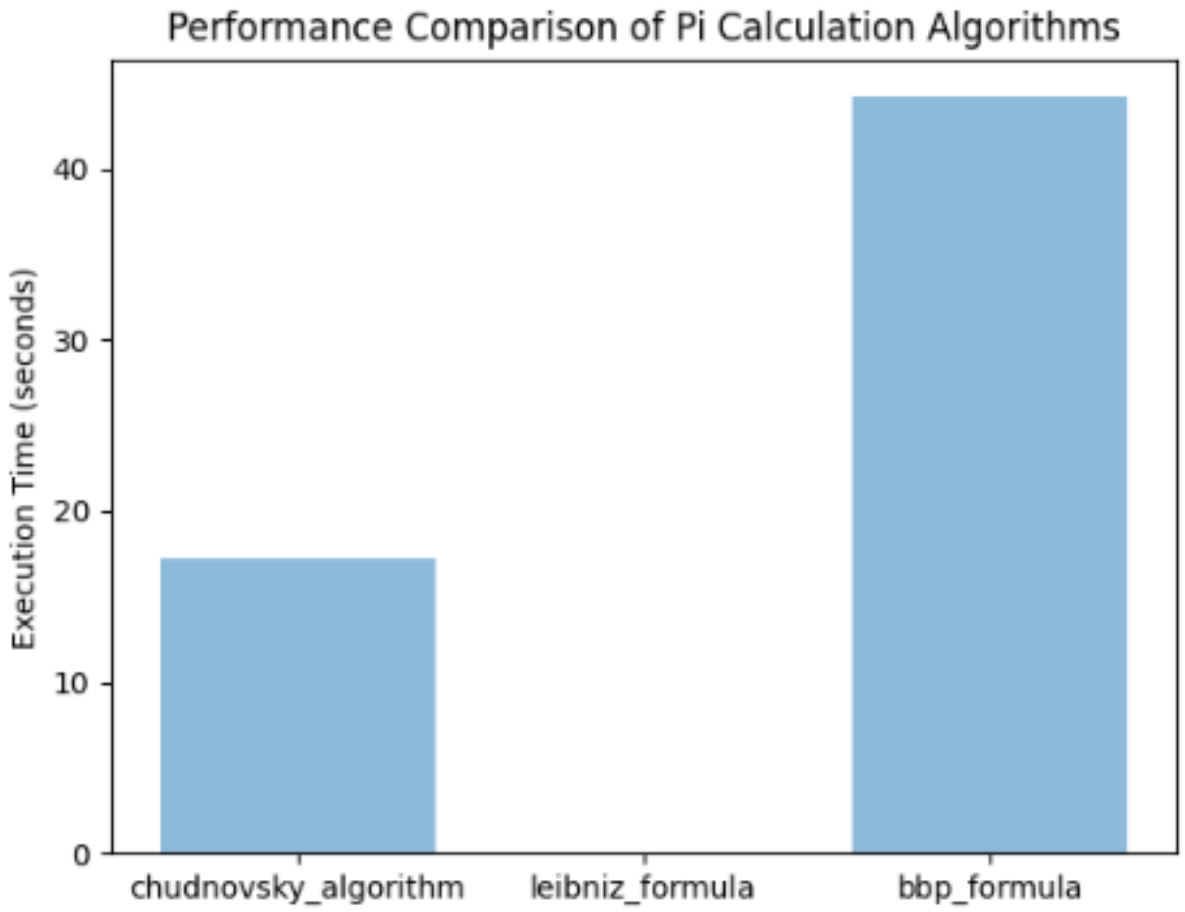
```
# Algorithm 3: Bailey{Borwein{Plouffe (BBP) Formula
def bbp_formula(n):
    mp.dps = n + 1
    pi = 0
    for k in range(n):
        pi += (1 / (16 ** k)) * ((4 / (8 * k + 1)) - (2 / (8 * k + 4))
            - (1 / (8 * k + 5)) - (1 / (8 * k + 6)))
    return pi


def analyze_algorithms():
    n = int(input("Enter the number of decimal places (n): "))
    algorithms = [chudnovsky_algorithm, leibniz_formula, bbp_formula]
    timings = []
    for algorithm in algorithms:
        start_time = time.time()
        pi = algorithm(n)
        end_time = time.time()
        timings.append(end_time - start_time)
        print(f"{algorithm.__name__}: {pi}, time: {end_time - start_time:.6f} seco

    plt.bar(np.arange(len(algorithms)), timings, align='center', alpha=0.5)
    plt.xticks(np.arange(len(algorithms)), [algorithm.__name__ for algorithm in al
    plt.ylabel('Execution Time (seconds)')
    plt.title('Performance Comparison of Pi Calculation Algorithms')
    plt.show()


if __name__ == '__main__':s
    analyze_algorithms()
```

Screenshot:


Performance Comparison of Pi Calculation Algorithms
Execution Time (seconds)
0
10
20
30
40
chudnovsky_algorithm
leibniz_formula
bbp_formula

Conclusion

In conclusion, the code presented in this context demonstrates the implementation and comparison of three algorithms for approximating the value of : the Chudnovsky Algorithm, Leibniz Formula, and Bailey-Borwein-Plouffe (BBP) Formula. Each algorithm has its own strengths and characteristics.

The Chudnovsky Algorithm is known for its rapid convergence and high accuracy, making it suitable for applications that require precise calculations of . However, it relies on the mpmath library and can be computationally intensive for large values of n.

The Leibniz Formula is a simple algorithm that is easy to implement, but it converges relatively slowly compared to the other algorithms. It does not require any external libraries and can be used for basic calculations of .

The BBP Formula is a spigot algorithm that directly calculates hexadecimal digits of . It has a fast convergence rate and high precision. Similar to the Chudnovsky Algorithm, it depends on the mpmath library for arbitrary-precision arithmetic.

The code allows for a comparison of execution times among the three algorithms, providing insights into their relative efficiency. The choice of algorithm depends on the specific requirements of the application, such as desired accuracy, available computational resources, and implementation complexity.

Overall, this exercise showcases different approaches to approximating and highlights the trade-offs between accuracy, efficiency, and ease of implementation. The selection of an algorithm should consider the specific needs of the application, striking a balance between precision and computational resources.