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 Gazea Sandu
Verified:
asist.univ. Fiștic Cristofor

Chișinău, 2023

Content

Introduction

Pi, that magical mathematical constant, has been the apple of the eye for mathematicians, scientists, and number enthusiasts for hundreds of years. Its irrationality and transcendence, as well as its omnipresence across many areas of math and science, have made it a tantalizing topic of deep investigation. One of the many mysteries of pi is unlocking its decimal digits, a challenge that has stirred great interest and led to the creation of a plethora of algorithms.

In this study, we're diving headfirst into the world of algorithms specifically designed to calculate the elusive decimal digits of pi. These algorithms, born from mathematical brilliance, numerical methodologies, and computational strategies, give us a way to estimate the seemingly infinite, non-repeating decimals of this fascinating constant. They enable us to hone in on its value with as much precision as we desire.

Evaluating these algorithms empirically is essential to comprehend their efficiency, accuracy, and convergence behavior. By putting these algorithms under the microscope, we can glean insights into their strengths, limitations, and computational demands. Such empirical scrutiny helps us pick the best-fit algorithm for a specific precision need, optimize computational resources, and highlight areas ripe for improvement.

Throughout our research, we'll be delving into a broad spectrum of algorithmic strategies used in the quest to compute N decimal digits of pi. From time-honored techniques like Machin-like formulas and series expansions, to modern methods including spigot algorithms and digit extraction techniques, each algorithm presents a unique angle to approximate the digits of pi. We'll dissect the mathematical logic underpinning these algorithms and scrutinize their computational traits.

Moreover, we'll apply empirical analysis to assess the performance of these algorithms. By tracking metrics like execution speed, memory footprint, and precision, we can juxtapose the algorithms in different settings. We aim to provide valuable insights into the behavior and efficiency of these algorithms via comprehensive computational experiments and numerical simulations, helping researchers and practitioners make informed decisions.

Ultimately, our research is a thorough exploration and empirical examination of algorithms for determining N decimal digits of pi. By merging theory, algorithmic creativity, and empirical inquiry, we hope to illuminate the extraordinary universe of pi and add to the relentless quest to understand and compute this iconic mathematical constant.

The Bailey–Borwein–Plouffe (BBP) formula

The Bailey–Borwein–Plouffe (BBP) formula is a clever mathematical trick that allows us to calculate specific digits of the number (pi) without having to calculate all the previous digits. It was discovered by Simon Plouffe in 1995, based on earlier work by Peter Borwein and Jonathan M. Borwein. The formula states that can be expressed as an infinite series. By plugging in different values for a variable called ”k,” we can compute individual digits of in hexadecimal format. Each term of the series contributes a single digit to the final result. The beauty of the BBP formula lies in its efficiency. Unlike traditional methods that require calculating all the preceding digits, the BBP formula allows us to skip ahead and directly calculate the specific digits we're interested in. This is incredibly handy when we don't need in its entirety but only a few particular digits. The discovery of the BBP formula has revolutionized the way we approach . It has become a valuable tool for generating and verifying hexadecimal representations of this important mathematical constant. Code:

```
function calculatePiHexDigit(k):
    result = 0
    powerOf16 = 1

    for i from 0 to k:
        term1 = 4.0 / (8 * i + 1)
        term2 = 2.0 / (8 * i + 4)
        term3 = 1.0 / (8 * i + 5)
        term4 = 1.0 / (8 * i + 6)

        term = (term1 - term2 - term3 - term4) / powerOf16
        result = result + term

        powerOf16 = powerOf16 * 16

    return result
```

Spigot algorithm

The Spigot algorithm is a method used to compute the digits of certain irrational numbers, such as (pi), e (Euler's number), and 2 (square root of 2), in a digit-by-digit fashion without the need to calculate all the

preceding digits. It was developed by Stanley Rabinowitz and Stan Wagon in 1995. The algorithm gets its name from the analogy of a spigot, which refers to a tap or valve that controls the flow of water. Just as a spigot allows water to be released in a controlled manner, the Spigot algorithm enables the extraction of digits of an irrational number sequentially, one digit at a time. The Spigot algorithm utilizes a combination of mathematical techniques, such as modular arithmetic, digit extraction, and iterative calculations. It exploits the inherent patterns and properties of these irrational numbers to generate their digits in a systematic way.The key idea behind the Spigot algorithm is to maintain a ”state” or ”accumulator” that stores partial results as each digit is computed. This state is updated in each iteration, and the extracted digit is output. The algorithm continues until the desired number of digits is obtained. The advantage of the Spigot algorithm lies in its ability to generate individual digits of an irrational number without needing to compute the entire number. This is particularly useful when only specific digits are required or when memory or computational resources are limited. Different variations of the Spigot algorithm have been developed for various irrational numbers, each tailored to exploit the specific mathematical properties of the target number. These algorithms have contributed to the advancement of digit extraction techniques and have found applications in fields such as mathematics, computer science, and cryptography.

Code:

```
function spigotAlgorithm(digitCount):
    result = []  // List to store the computed digits
    state = 0    // Initial state or accumulator

    for i from 1 to digitCount:
        carry = 0

        // Perform digit extraction and update the state
        for j from length(result) down to 1:
            numerator = result[j-1] * 10 + carry
            quotient = floor(numerator / i)
            remainder = numerator mod i
            result[j-1] = quotient
            carry = remainder

        // Append the extracted digit to the result
        result.append(carry)
```

```
        // Adjust the state for the next iteration
        state = carry * 10

    return result
```

The Chudnovsky algorithm

The Chudnovsky algorithm is a clever and efficient technique discovered by the Chudnovsky brothers in 1989 to calculate the decimal digits of , the famous mathematical constant. It works by using a formula that involves a series of terms that gradually converge to the value of . Instead of calculating directly, the Chudnovsky algorithm uses this special formula to compute it step by step. Each iteration of the algorithm brings us closer to the actual value of , and the good news is that it converges quickly. This means that even with a relatively small number of iterations, we can get a significant number of correct decimal places of . The Chudnovsky algorithm has some important advantages compared to other methods. It is known for its speed and efficiency, which makes it ideal for heavy-duty computations and large-scale calculations. It takes advantage of efficient ways to compute factorials and uses fractions, which helps simplify the calculations. This algorithm has had a profound impact on the field of numerical computation, enabling accurate and efficient estimation of . It has been used to break records in calculating the most decimal places of , and it continues to be a valuable tool for various mathematical and scientific applications. Code:

```
function chudnovskyAlgorithm(decimalPlaces):
    iterations = ceil(decimalPlaces / 14.181647462725477)  // Estimate number of itera
    sum = 0
    k = 0

    for k from 0 to iterations:
        numerator = factorial(6 * k) * (545140134 * k + 13591409)
        denominator = factorial(3 * k) * (factorial(k) ^ 3) * (-640320) ^ (3 * k)
        term = numerator / denominator
        sum = sum + term

    pi = (426880 * sqrt(10005)) / sum
    return pi rounded to the desired decimal places
```

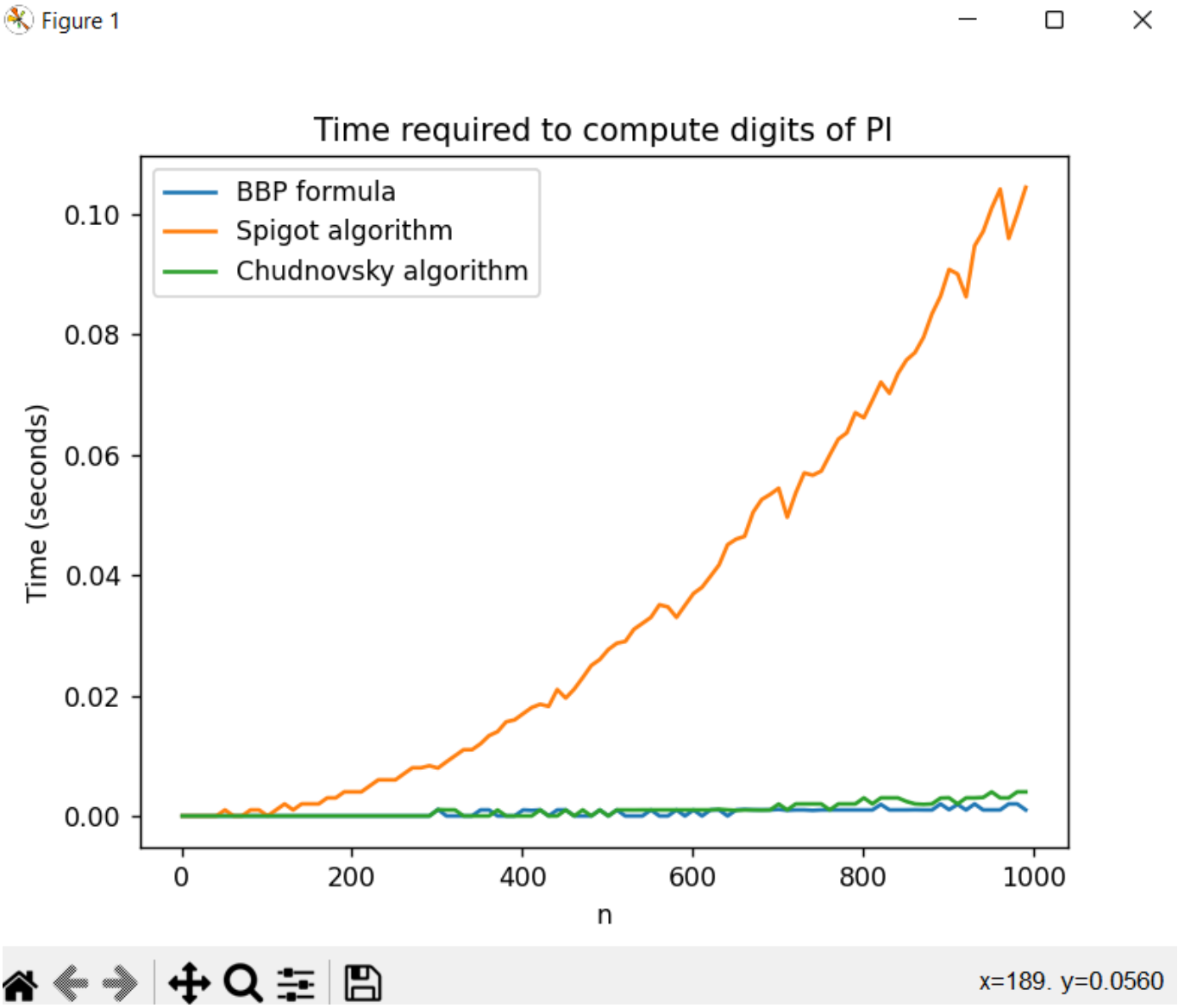
Implementation

```
import time
from matplotlib import pyplot as plt
from math import factorial, sqrt
from decimal import Decimal, getcontext

# BBP Algorithm to compute nth digit of PI
def calc_bbp(n):
    pi_approx = 0
    # Iteratively compute each term of the series
    for k in range(n):
        pi_approx += (1/16**k) * ((4/(8*k+1)) - (2/(8*k+4)) - (1/(8*k+5)) - (1/(8*
    return int(pi_approx * 16) % 16

# Spigot Algorithm to compute nth digit of PI
def calc_spigot(n):
    pi_digits = [2]
    # Iteratively compute each digit of PI
    for i in range(1, n+1):
        carry_over = 0
        # Adjust each digit in place and carry over the remainder
        for j in reversed(range(len(pi_digits))):
            val = 10 * pi_digits[j] + carry_over
            pi_digits[j] = val // (2*i - 1)
            carry_over = val % (2*i - 1)
        # Handle any remaining carry
        while carry_over > 0:
            pi_digits.insert(0, carry_over % 10)
            carry_over //= 10
    return pi_digits[-1]
```

```
# Chudnovsky Algorithm to compute nth digit of PI
def calc_chudnovsky(n):
    getcontext().prec = n+1
    k_val = n // 14
    chud_sum = Decimal(0)
    # Iteratively compute each term of the series
    for i in range(k_val+1):
        numer = (-1)**i * factorial(6*i) * (13591409 + 545140134*i)
        denom = factorial(3*i) * factorial(i)**3 * 640320**(3*i)
        chud_sum += Decimal(numer) / Decimal(denom)
    chud_sum *= Decimal(12)
    result = Decimal(sqrt(10005)) * chud_sum
    return int(result * 10**n) // 10 % 10

# Array of n-values for which we compute PI's digits
n_values = list(range(1, 1001, 10))
bbp_times = []
spigot_times = []
chudnovsky_times = []

# Time the execution of each algorithm for each n
for n in n_values:
    start_t = time.time()
    calc_bbp(n)
    bbp_times.append(time.time() - start_t)

    start_t = time.time()
    calc_spigot(n)
    spigot_times.append(time.time() - start_t)

    start_t = time.time()
    calc_chudnovsky(n)
    chudnovsky_times.append(time.time() - start_t)
```

```
# Plot the time taken by each algorithm
plt.plot(n_values, bbp_times, label='BBP formula')
plt.plot(n_values, spigot_times, label='Spigot algorithm')
plt.plot(n_values, chudnovsky_times, label='Chudnovsky algorithm')
plt.xlabel('n')
plt.ylabel('Time (seconds)')
plt.title('Time required to compute digits of PI')
plt.legend()
plt.show()
```

Screenshot:



Conclusion

In conclusion, we have explored three fascinating formulas for calculating mathematical constants. Each formula has its own unique characteristics and applications. The Bailey–Borwein–Plouffe (BBP) formula provides a method for calculating specific hexadecimal digits of without the need to compute all the preceding digits. It offers efficiency and is particularly useful when only certain digits are required. The Spigot algorithm is a general approach that allows for the computation of the decimal digits of various irrational numbers, including , in a digit-by-digit fashion. It leverages the concept of rapidly converging series and is adaptable to different numbers. The Chudnovsky algorithm is a powerful and efficient technique specifically developed for computing the decimal digits of . It relies on a series formula and incorporates advanced mathematical concepts such as factorials and infinite series. The algorithm demonstrates remarkable speed and accuracy, making it well-suited for high-performance computing and large-scale calculations. While all three formulas contribute to the field of numerical computation and allow for the calculation of mathematical constants, they differ in their specific approaches and mathematical properties. The BBP formula excels at hexadecimal digit extraction, the Spigot algorithm provides a versatile digit-by-digit computation method, and the Chudnovsky algorithm stands out for its speed and efficiency in computing the decimal digits of . The choice of which formula to use depends on the requirements of the specific application and the desired level of precision. These formulas continue to play a significant role in mathematical research, computer science, and various fields where accurate and efficient estimation of mathematical constants is needed.

$https://github.com/GSandu1/LabsAA.git$