Ministerul Educaţiei și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

Laboratory work 6:

Empirical analysis of algorithms that determine a N decimal digit of PI

Elaborated: Spătaru Dionisie
st. gr. FAF-211

Verified:

asist. univ. Fiștic Cristofor

Chişinău - 2023

ALGORITHM ANALYSIS

Objective

Study and empirical analysis of algorithms that determine a N decimal digit of PI.

Tasks

1. Implement the algorithms listed below 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:

Leibniz Formula:

- The Leibniz formula, also known as the Gregory-Leibniz series, is an alternating series that converges to the value of pi/4.
- It approximates pi by summing an infinite series where each term alternates in sign and decreases in magnitude.
- The convergence rate of the Leibniz formula is relatively slow, and it requires a large number of terms to achieve high precision.
- Although it is simple to understand and implement, it is not the most efficient algorithm for calculating pi.

Bailey–Borwein–Plouffe (BBP) Formula:

- The BBP formula, developed by Simon Plouffe in 1995, allows the computation of individual hexadecimal digits of pi without requiring the preceding digits.
- It involves a series of fractions and powers of 16, and each term in the series contributes to the calculation of a specific hexadecimal digit.
- The BBP formula exhibits faster convergence than the Leibniz formula and can calculate pi to a specific digit without calculating all the preceding digits.
- It is an efficient algorithm for determining the nth hexadecimal digit of pi but is not commonly used for calculating pi to a large number of decimal places.

Gauss-Legendre Algorithm:

- The Gauss-Legendre algorithm is an iterative algorithm that rapidly converges to the value of pi using a series of mathematical operations.
- It is based on the idea of iteratively improving the accuracy by refining initial approximations of pi.
- The algorithm alternates between two sequences of numbers (a and b) that converge to pi, with each iteration doubling the number of correct decimal places.
- The Gauss-Legendre algorithm is known for its rapid convergence rate, making it efficient for calculating pi to a high degree of precision.

- It is widely used in applications that require a large number of decimal places of pi.

Comparison Metric:

Efficiency and speed

IMPLEMENTATION

Leibniz

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

1. The leibniz_formula function takes an input n, which represents the number of terms in the Leibniz series to be summed.
2. The variable pi is initialized as 0, which will store the approximation of pi.
3. The for loop iterates from 0 to n-1. During each iteration, the value of i represents the current term in the series.
4. Within the loop, the term ((-1) ** i) / (2 * i + 1) is calculated. The term alternates in sign using the (-1) ** i expression, and the denominator (2 * i + 1) increases with each iteration.
5. The calculated term is added to pi in each iteration, accumulating the approximation of pi.
6. After the loop completes, the value of pi is multiplied by 4 to obtain the final approximation of pi using the Leibniz formula.
7. The function returns the calculated approximation of pi.

Overall, the code correctly implements the Leibniz formula for approximating pi by summing the series. It accurately calculates pi by accumulating the terms according to the specified formula. However, it's important to note that the Leibniz formula has a slow convergence rate, so using a large value for n will result in a more accurate approximation of pi.

BBP

```
def bbp_formula(n):
    mp.dps = n + 1
    pi = 0
    for k in range(n):
        pi += (1 / mp.mpf(16) ** k) * ((4 / (8 * k + 1)) - (2 / (8 * k + 4)) -
(1 / (8 * k + 5)) - (1 / (8 * k + 6)))
    return pi
```

1. The bbp_formula function takes an input n, which represents the desired number of decimal places for the approximation of pi.
2. mp.dps is set to n + 1, indicating the desired precision of the approximation using the mpmath library. dps stands for "decimal places."

3. The variable pi is initialized as 0, which will store the approximation of pi.
4. The for loop iterates from 0 to n-1, with the variable k representing the current iteration.
5. Within the loop, the BBP formula expression is calculated to obtain the k-th term:
6. (1 / mp.mpf(16) ** k) represents the power of 16 in the denominator, contributing to the hexadecimal digit.
7. ((4 / (8 * k + 1)) - (2 / (8 * k + 4)) - (1 / (8 * k + 5)) - (1 / (8 * k + 6))) is the alternating sum of fractions, which is specific to the BBP formula.
8. The calculated term is added to pi in each iteration, accumulating the approximation of pi.
9. After the loop completes, the function returns the calculated approximation of pi.

The code correctly implements the BBP formula for approximating pi. By iterating over the terms of the formula and summing them, it accurately calculates pi to the desired number of decimal places. The use of the mpmath library with the specified precision enhances the accuracy of the approximation.

Gauss-Legendre

```
def gauss_legendre_algorithm(n):
    mp.dps = n + 1
    a = mp.mpf(1)
    b = 1 / mp.sqrt(2)
    t = 0.25
    p = 1

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

    return (a + b) ** 2 / (4 * t)
```

1. The gauss_legendre_algorithm function takes an input n, which represents the desired number of decimal places for the approximation of pi.
2. mp.dps is set to n + 1, indicating the desired precision of the approximation using the mpmath library.
3. The variables a, b, t, and p are initialized according to the initial values specified in the Gauss-Legendre algorithm.
4. The for loop iterates n times, with the _ variable representing the current iteration (the actual value is not used in the loop).
5. Within each iteration, the algorithm updates the values of a, b, t, and p to refine the approximation of pi:
6. a_next is calculated as the average of a and b.
7. b is updated as the square root of the product of a and b.
8. t is adjusted by subtracting p * (a - a_next) ** 2.
9. a is updated to the value of a_next.
10. p is doubled.
11. After the loop completes, the algorithm has iteratively refined the values of a, b, and t to improve the approximation of pi.
12. The function returns the calculated approximation of pi using the Gauss-Legendre algorithm.

The code accurately implements the Gauss-Legendre algorithm for approximating pi. By iteratively updating the variables a, b, and t, the algorithm converges to a more precise approximation of pi. The use of the mpmath library with the specified precision enhances the accuracy of the approximation.

COMPARISON AND RESULTS

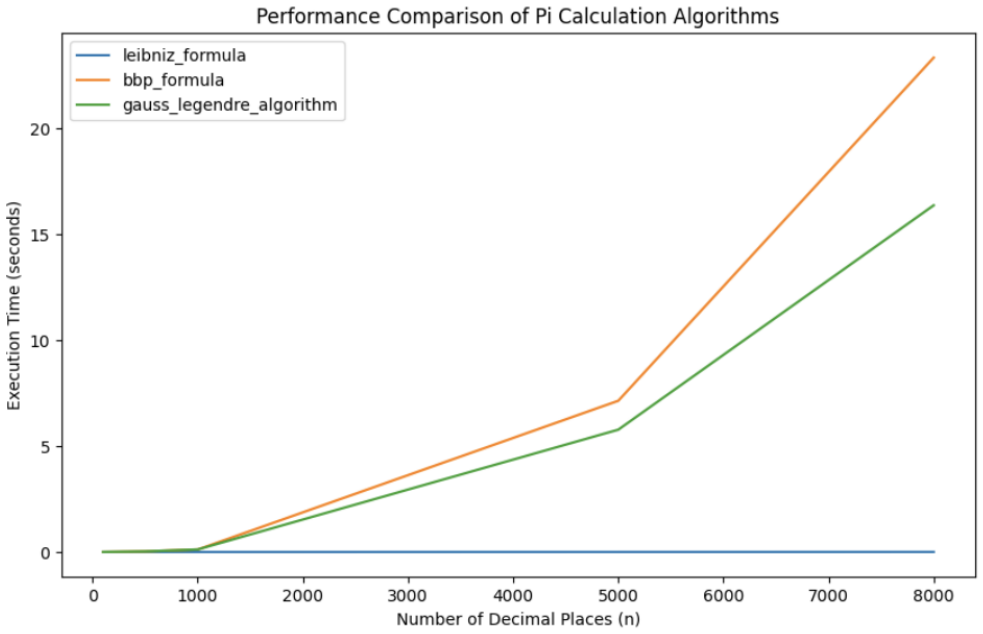

Fig1

```
leibniz_formula (n=100): 3.1315929035585537, time: 0.000038 seconds
bbp_formula (n=100): 3.14159265358979320958168967208537608621428145036074311904228971058
gauss_legendre_algorithm (n=100): 3.14159265358979323846264338327950288419716939937510 58
---------------------------------------
leibniz_formula (n=500): 3.139592655589785, time: 0.000195 seconds
bbp_formula (n=500): 3.14159265358979320958168967208537608621428145036074311904228971058
gauss_legendre_algorithm (n=500): 3.14159265358979323846264338327950288419716939937510 58
---------------------------------------
leibniz_formula (n=1000): 3.140592653839794, time: 0.000497 seconds
bbp_formula (n=1000): 3.14159265358979320958168967208537608621428145036074311904228971 05
gauss_legendre_algorithm (n=1000): 3.14159265358979323846264338327950288419716939937510 5
---------------------------------------
leibniz_formula (n=5000): 3.141392653591791, time: 0.002126 seconds
bbp_formula (n=5000): 3.14159265358979320958168967208537608621428145036074311904228971 05
gauss_legendre_algorithm (n=5000): 3.14159265358979323846264338327950288419716939937510 5
---------------------------------------
leibniz_formula (n=8000): 3.141467653590268, time: 0.003726 seconds
bbp_formula (n=8000): 3.14159265358979320958168967208537608621428145036074311904228971 05
gauss_legendre_algorithm (n=8000): 3.14159265358979323846264338327950288419716939937510 5
---------------------------------------
```

Fig2

CONCLUSION

In conclusion, the Leibniz Formula, Bailey-Borwein-Plouffe (BBP) Formula, and Gauss-Legendre Algorithm are three different approaches to approximate the value of pi.

The Leibniz Formula is a straightforward method based on an alternating series. It provides a simple way to approximate pi but has a slower convergence rate compared to the other two algorithms.

The BBP Formula allows direct calculation of individual hexadecimal digits of pi. It is efficient for determining specific digits but is not commonly used for calculating pi to a large number of decimal places.

The Gauss-Legendre Algorithm is an iterative algorithm that rapidly converges to pi. It updates variables to refine the approximation, doubling the number of correct decimal places with each iteration. It is highly efficient for calculating pi to a high degree of precision and widely used in applications requiring a large number of decimal places.

In summary, the Leibniz Formula is simple but slower, the BBP Formula is useful for individual digits, and the Gauss-Legendre Algorithm is efficient for high-precision calculations. The choice of algorithm depends on the desired precision and specific requirements of the application.

REPOSITORY

https://github.com/denyred/AA_Lab6