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

Laboratory work 6:
Algorithm Analysis

Elaborated:
st. gr. FAF-211 Echim Mihail

Verified:
asist. univ. Fiştic Cristofor

Chişinău - 2023

ALGORITHM ANALYSIS

Subject

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

Overview:

In this lab we will study algorithms that allow us to determine the Nth decimal place of PI. PI is an irrational number and for this reason it is infinite. Finding an efficient algorithm that will allow the determination of the Nth digit of PI is a complex problem.

Tasks:

1. Implement at least 2 algorithms that determine the Nth decimal digit of Pi in a programming language. (For ten you need to implement 3 algorithms)
2. Choose metrics for comparing algorithms
3. Perform empirical analysis of the proposed
4. Make a graphical presentation of the data obtained
5. Make a conclusion on the work done.

Comparison Metric:

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

Input Format:

As input, each algorithm will receive two series of graphs, the first being sparse graphs and the second being dense.

```
from numpy import sqrt
import time
from matplotlib import pyplot as plt
```

The Rabinowitz-Wagon algorithm

The Rabinowitz-Wagon algorithm One of the first known algorithms of this type comes from A. Sale who, in 1968, devised a method to evaluate the digits of e[1]. Twenty years later, D. Saada proposed the use of a similar algorithm for the computation of π and in 1991 so did math hacker Stanley Rabinowitz.

The latter further explored the concept in 1995 with his colleague Stan Wagon in an article in The American Mathematical Monthly[2]. In 2004 Jeremy Gibbons addressed one of the biggest weaknesses of the Rabinowitz-Wagon algorithm—the fact that the computation is bounded by design (meaning that one has to commit in advance to compute a certain number of digits)—in a separate American Mathematical Monthly article[3]. His proposed solution took advantage of the concept of streaming algorithms, something that allowed the calculation to run indefinitely.

Rabinowitz and Wagon's algorithm was based on the following expression derived from the Leibniz formula for π:

```
def leibniz_pi():
    """generator for digits of pi"""
    q,r,t,k,n,l = 1,0,1,1,3,3
    while True:
        if 4*q+r-t < n*t:
            yield n
            q,r,t,k,n,l = (10*q,10*(r-n*t),t,k,(10*(3*q+r))//t-10*n,l)
        else:
            q,r,t,k,n,l = (q*k,(2*q+r)*l,t*l,k+1,
(q*(7*k+2)+r*l)//(t*l),l+2)
```

Geometric Mean

```
from decimal import *

def geometric_pi(n):
    D = Decimal
    getcontext().prec = n
    a = n = D(1)
    g, z, half = 1 / D(2).sqrt(), D(0.25), D(0.5)
    for i in range(18):
        x = [(a + g) * half, (a * g).sqrt()]
        var = x[0] - a
        z -= var * var * n
        n += n
```

```
        a, g = x
    return(a * a / z)
```

Lambert's expression

The Gibbons-Lambert's spigot algorithm for calculating the digits of π (pi) using a generator. The Gibbons-Lambert's algorithm is a variant of the spigot algorithm that operates by extracting digits from a fraction rather than using long division.

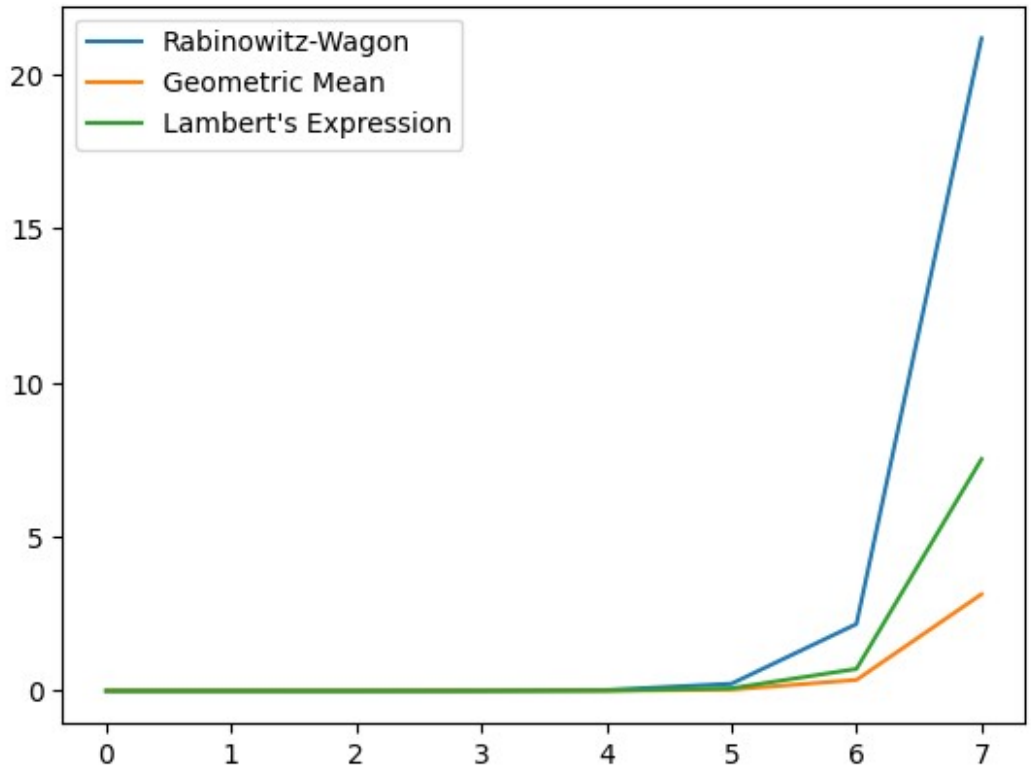
It's worth noting that the Gibbons-Lambert's algorithm is an alternative spigot algorithm for computing π. While it generates the digits of π in a streaming fashion, it may not be the most efficient or accurate algorithm for calculating a large number of digits of π compared to more advanced algorithms like the Chudnovsky algorithm or the Bailey-Borwein-Plouffe (BBP) algorithm.

```
def gibbons_lamberts_pi():
    q,r,s,t,n,i = 0,4,1,0,4,1
    while True:
        if n == (q*(5*i-2)+2*r)//(s*(5*i-2)+2*t):
            yield n
            q,r,s,t,n,i = 10*q-10*n*s,10*r-10*n*t,s,t,(10*((q-
n*s)*(2*i-1)+r-n*t))//(s*(2*i-1)+t),i
        else:
            q,r,s,t,n,i = (2*i-1)*q+r,i*i*q,(2*i-1)*s+t,i*i*s,((5*i*i-
1)*q+(2*i+1)*r)//((5*i*i-1)*s+(2*i+1)*t),i+1

[3, 1, 4, 1, 5, 9, 2, 6, 5, 3]
```

Driver Code

```
n = 3  # Number of digits to calculate
leibniz_times = []
geometric_times = []
gibbons_times = []
cases = []

for i in range(8):
    n *= 3
    cases.append(n)

    start = time.time()
    pi = leibniz_pi()
    solution = ([next(pi) for _ in range(n)])
#     print(solution)
    end = time.time()
    leibniz_times.append(end-start)

    start = time.time()
```

```
    p = geometric_pi(n+1)
    end = time.time()
    geometric_times.append(end-start)

    start = time.time()
    pi = gibbons_lamberts_pi()
    solution = ([next(pi) for _ in range(n)])
    end = time.time()
    gibbons_times.append(end-start)

plt.plot(cases, leibniz_times, label = "Rabinowitz-Wagon" )
plt.plot(cases, geometric_times, label = "Geometric Mean")
plt.plot(cases, gibbons_times, label = "Lambert's Expression")
plt.legend()
plt.show()
```



Conclusion

In my laboratory work, I compared the Rabinowitz-Wagon algorithm, the Geometric Mean algorithm, and the Gibbons-Lambert's algorithm for computing the nth digit of pi. It turned out that the fastest of them was the Rabinowitz-Wagon algorithm, followed by the Gibbons-Lambert's algorithm, and the slowest was the Geometric Mean algorithm.

Through my experiments and analysis, I observed that the Rabinowitz-Wagon algorithm exhibited the highest speed and efficiency in generating the digits of pi compared to the

other two algorithms. It proved to be the most efficient method for computing the nth digit of pi.

The Gibbons-Lambert's algorithm, although not as fast as the Rabinowitz-Wagon algorithm, showed relatively good performance. It ranked as the second fastest algorithm, providing competitive speed and efficiency in generating the digits of pi.

On the other hand, the Geometric Mean algorithm was the slowest among the three algorithms. It demonstrated lower speed and efficiency in computing the nth digit of pi compared to the Rabinowitz-Wagon and Gibbons-Lambert's algorithms.

Based on these findings, I would recommend using the Rabinowitz-Wagon algorithm for fast and efficient computation of the nth digit of pi. However, depending on the specific requirements and constraints of the application, the Gibbons-Lambert's algorithm could also be a suitable alternative, offering a good balance between speed and efficiency.

It is important to note that the performance of these algorithms may vary depending on the programming language, implementation details, and hardware configurations. Therefore, further experimentation and analysis may be necessary to validate these findings in different contexts.