TECHNICAL UNIVERSITY OF MOLDOVA

FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS

DEPARTMENT: **SOFTWARE ENGINEERING AND AUTOMATION**

CC - LABORATORY WORK NR 5

N-th Digit of π

Ciuş Iurie   FAF-203

Chişinău 2022

Contents

1 Algorithm Analysis

Algorithm analysis is an important part of computational complexity theory, which provides theoretical estimation for the required resources of an algorithm to solve a specific computational problem. Analysis of algorithms is the determination of the amount of time and space resources required to execute it.

1.1 Introduction

The current paper work represents my implementation of two algorithms to determine the N-th digit of π. The fascination of the number π by mathematicians is ancient, and numerous computations of its digits have been performed in the history. Later, computers have been used to increase the number of computed digits There are multiple ways by which we can calculate the nth digit of pi by using **Arctan formula** and **Bailey–Borwein–Plouffe formula. Chudnovsky Algorithm** is a fast way of calculating the digits of pi and is similar to the arctan's formula.

$$\pi + 3 = \sum_{k>0} \frac{k2^k}{\binom{2k}{k}}$$

This formula has no big powers of primes in denominators, and this was the key success factor in **Plouffe** approach.

Later, based on the same formula, Fabrice Bellard refined Plouffe technique with an algorithm using $O(n^2)$ elementary operations on numbers of size $O(logn)$.

1.2 Objectives

- Implement at least 2 algorithms to determine the N-th digit of π.
- Establish the properties of the input data in relation to which the analysis is made.
- Choose the metric for comparing algorithms.
- Perform empirical analysis of the proposed algorithms.
- Make a conclusion on the work done.

1.3 Theoretical Notes

Our starting point is the classical following alternating series to compute π:

$$\frac{\pi}{4} = arctan(1) = \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1}$$

In this form, the formula is well suited to nth decimal digit computation, but its convergence is too slow.

Let an alternating series of the form:

$$S = \sum_{k=0}^{\infty} (-1)^k a_k$$

where we assume that there exists a positive function weight $w(x)$ such that

$$a_k = \int_0^1 x^k w(x) dx.$$

Various acceleration convergence techniques exist for such series (finite differences Euler acceleration process for example, ...), and can be generalized with the following result from Cohen, Villegas and Zagier.

Let $P_n(x) = \sum_{k=0}^{n} p_k(-x)^k$ a degree n polynomial for which $P_n(-1) \neq 0$. We define:

$$|S_n - S| <= \frac{1}{|P_n(-1)|} \int_0^1 \frac{|P_n(x)| w(x)}{1+x} dx \leq \frac{M_n}{|P_n(-1)|} S$$

1.4 Description of the algorithm

Algorithm 1 (n-th digit computation of π with very low memory) *The following algorithm computes the fractional part of $10^n\pi$ with an error $< 10^{-n_0}$ when $n \geq 4n_0$.*

1. Define integers M and N by

$$M = 2\left[\frac{n}{log^3(n)}\right] \quad and \quad N = \left[(n + n_0 + 1)\frac{log(10)}{log(2eM)}\right]$$

2. (Computation of B) Initialize b = 0 a floating point value. For index k, $0 \leq k < (M + 1)N$ perform the following operations :

- Compute $x = 4 \times 10^n \ mod \ 2k + 1$
- Compute $b := \{b + (-1)^k x/(2k + 1)\}$

3. (Computation of C) Initialize c = 0 a floating point value. For index $k, 0 \leq k < N$ perform the following operations

- Compute $x = \sum_{j=0}^{k}(Nj)mod(2MN + 2k + 1)$

4. (Final step) Compute the value x as the fractional part of $b - c(x = b - c - [b - c])$. Then x is an approximation of $\{10n\pi\}$ with an error less than $10 - n_0$.

2 Code

The following chapter represents my implementation and results of the current laboratory work.

2.1 Implementation

```
import math

def sqrt(n,m):
    m1=10**16
    m2=float((n*m1)//m)/m1
    b=(int(m1*math.sqrt(m2))*m)//m1
    n_m=n*m
    while True:
        a=b
        b=(b+n_m//b)//2
        if b==a:
            break
    return b

def power(n):
    if n==0:
        return 1
    r=power(n//2)
    if n%2==0:
        return r*r
    return r*r*10

def pi():
     m = power(100000)
     c = (640320**3)//24
     n = 1
     Ak = m
     Asum = m
     Bsum = 0
     while Ak != 0 :
          Ak *= -(6*n-5)*(2*n-1)*(6*n-1)
          Ak //= n*n*n*c
          Asum += Ak
          Bsum += n * Ak
          n = n + 1
          result = (426880*sqrt(10005*m,m)*m)//(13591409*Asum
    +545140134*Bsum)
          return result

stringPi = str(pi())
n = int(input())
```

```python
print(stringPi[n-1])
```

script.py

```javascript
/*Return pi to the nth digit using Javascript and JQ */

function calculatePi(n) {
  //Input too long or not a number, default to 30
  if (n === undefined || n > 30) {
    n = 30;
  }

  //Machin's formula for pi:
  return (16 * Math.atan(1 / 5) - 4 * Math.atan(1 / 239)).
   toFixed(n);
}
```

script.js

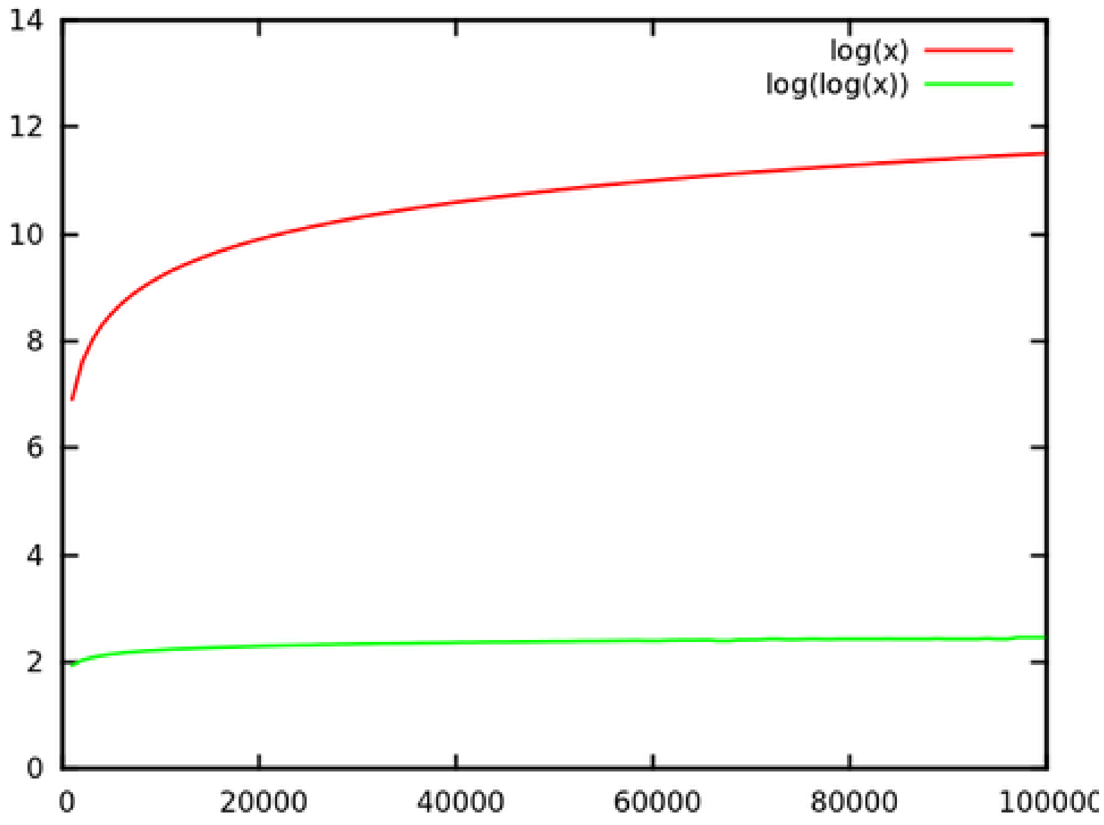
2.2 Complexity of the algorithm

Lemma 1 *For $k < N < m$, algorithm 2 has a memory need of $O(log^2 m)$ bits and time complexity of*

$$Cost_2(m) = O(log(m)) + O(k) + O(\lambda_k(m)k)$$

elementary operations on numbers of size $O(log(m))$, where $\lambda_k(m)$ is the number of distinct prime factors p of m such that $p \leq k$.

2.3 Graphs



3 Conclusion

During this laboratory work, I studied and analyzed two of the most popular algorithms to find the n-th digit of π, each with it's upsides and downsides.

On the one hand, we have been able to obtain an algorithm to compute directly the n-th decimal digit of π in nearly quadratic time and using only O(log2 n) memory; on the other hand, computing all the first n digits of π is possible in quasi-linear time and with memory O(n).

3.1 References

- https://github.com/IuraCPersonal/FAF203-CC