

# Experimentos usando Pi

<https://github.com/guilhermeivo/computing-projects/guilhermeivo>

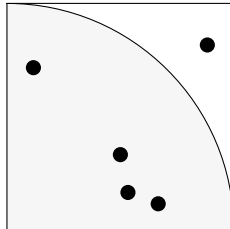
15 de abril de 2025

Queremos implementar algoritmos para calcular o valor de  $\pi$  usando métodos computacionais.

## PI USANDO MONTE CARLO

O método de Monte Carlo é uma técnica de amostragem estatística, teve seu desencadeamento após o desenvolvimento do *ENIAC*<sup>1</sup>, primeiro computador eletrônico, desenvolvido durante a Segunda Guerra Mundial, para lidar com grandes cálculos. O método baseia-se em números “aleatórios” para resolver os seus problemas, que é aplicado com sucesso em diversos problemas contemporâneos [3].

`monteCarlo()` O problema consiste em montar um quadrado de comprimento 1, e 1/4 de uma circunferência inscrita nesse quadrado e, de forma aleatória, disparar pontos dentro do espaço.



$$A = \pi \cdot R^2 \implies A = \pi$$
$$A_c = \frac{\pi}{4} \approx 4 \cdot \frac{\text{dentro}}{\text{total}} = \pi$$

```
long double monteCarlo(unsigned long long n) {
    unsigned long long ins;
    ins = 0;
    long double x, y, z, pi;

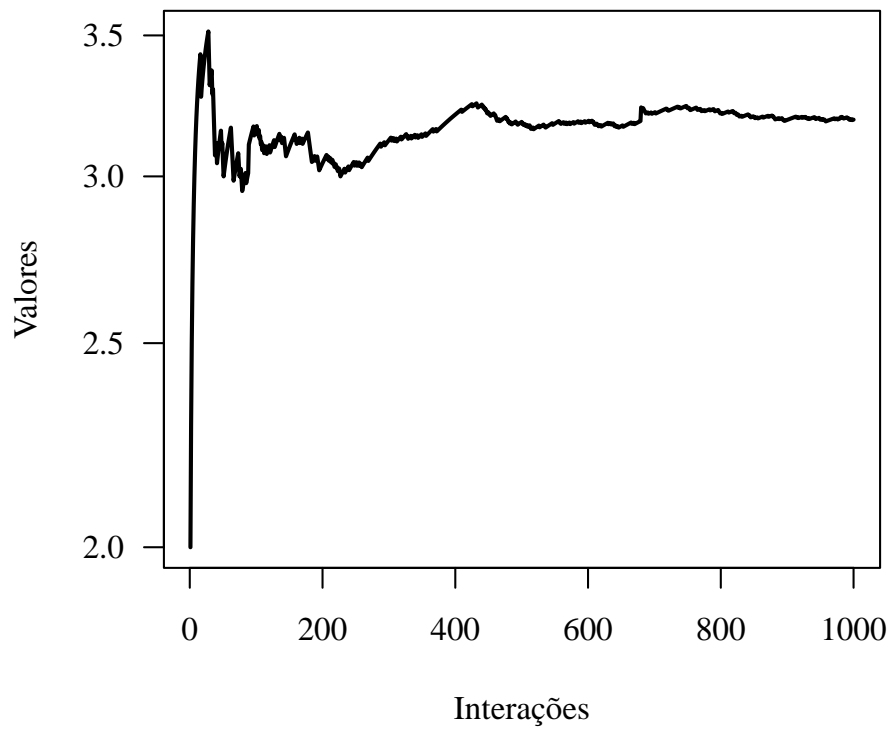
    srand(time(NULL));
    unsigned long long i;
    for (i = 0; i < n; i++) {
        x = (long double) rand() / (long double) RAND_MAX;
        y = (long double) rand() / (long double) RAND_MAX;

        z = x * x + y * y;
        if (z <= 1) {
            ins++;
        }
    }

    pi = (long double) 4 * ins;
}
```

<sup>1</sup>O ENIAC produziu 2,037 dígitos de  $\pi$  em 70 horas em 1949 usando a *Fórmula de Machin*[2, p. 592 e 627]

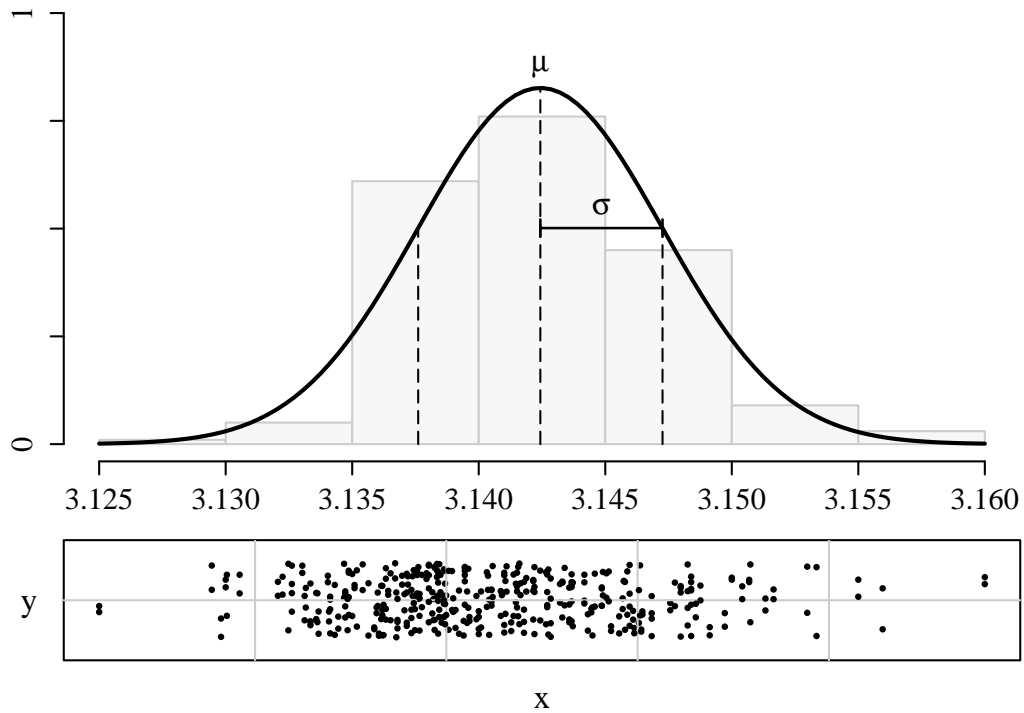
```
pi = (long double) pi / (long double) n;  
  
return pi;  
}
```

Gráfico 1: Convergência para o método de *Monte Carlo*

Fonte: Elaborado pelo próprio autor

Observando que os valores coletados na Gráfico 1 convergem com flutuações nos valores, pois os valores são obtidos de forma aleatória. Então, podemos interpretar estatisticamente este resultado realizando uma hipótese de normalidade na amostra.

Figura 1: Histograma dos valores obtidos no método de *Monte Carlo*, com curva de densidade, aplicado à coleta de  $\pi$



Fonte: Elaborado pelo próprio autor

Devido às propriedades da função, temos:

- O ponto  $x = \mu$  é um ponto de máximo absoluto, sendo o único ponto crítico do gráfico, representado pela linha pontilhada central da Figura 1.
- Possui dois pontos de inflexão  $x = \mu - \sigma$  e  $x = \mu + \sigma$ , representado pelas linhas pontilhadas nas extremidades do gráfico.
- A área entre  $\mu \pm \sigma$  representa 68% da área total do gráfico, ou seja, 68% dos valores de  $\pi$  estão presentes nesse intervalo.

Defina:  $mean(var)$ , média aritmética dos valores de  $\pi$  obtidos.

$$var = \{x_1, x_2, \dots, x_n\}$$

$$\mu = \bar{var} = \frac{1}{n} \sum_{i=1}^n x_i = \frac{x_1 + x_2 + \dots + x_n}{n}$$

$sd(var)$ , desvio padrão dos valores obtidos, ou seja, medida de dispersão com as mesmas unidades que a amostra.

$$\sigma = sd = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{var})^2}$$

$sd(var)/\sqrt{m}$ , desvio padrão da média ( $\bar{var}$ ), no limite para  $m$  grande. O desvio padrão da média ( $\xi$ )

caracteriza as flutuações dos valores  $\bar{v}ar_i$  em torno da média população  $\mu$ .

$$\xi = \frac{\sigma}{\sqrt{m}}$$

E realizando testes estatísticos nos dados para determinar se os dados afastam ou não da distribuição normal.

```
shapiro.test(var)
```

```
[1] Shapiro-Wilk normality test
```

```
data: var
```

```
W = 0.9987, p-value = 0.6886
```

Observando que o valor-p retornado pelo teste é maior que 0,05, fortalecendo a hipótese que os dados são normalmente distribuídos.

Além disso, tem-se:

```
library(nortest)
```

```
ad.test(var)
```

```
[1] Anderson-Darling normality test
```

```
data: var
```

```
A = 0.2726, p-value = 0.6684
```

```
ks.test(var, "pnorm", mean(var), sd(var))
```

```
[1] Asymptotic one-sample Kolmogorov-Smirnov test
```

```
data: var
```

```
D = 0.015002, p-value = 0.978
```

```
alternative hypothesis: two-sided
```

O valor-p em todos os testes anteriores foi superior a 0,05, fortalecendo a nossa hipótese inicial de que a normalidade não é rejeitada, como demonstrado na Figura 1. Então, apresentando erros de medição aleatórios.

## DESENVOLVENDO UM CASO GERAL

Para facilitar o desenvolvimento dos algoritmos para calcular  $\arctg$ , vamos realizar um caso geral, onde iremos adotar o valor para  $x$  sendo igual a  $1/a$ , pois em todas as fórmulas utilizadas é usado  $\arctg$  do tipo  $\arctg 1/a$ . Então, podemos desenvolver a série da função  $\arctg$  da seguinte maneira:

$$\begin{array}{rcl} S_n & = & 1 + r + r^2 + \dots + r^n \\ - r \cdot S_n & = & r + r^2 + \dots + r^n + r^{n+1} \\ \hline S_n - rS_n & = & 1 - r^{n+1} \end{array}$$

$$\Rightarrow S_n = \frac{1 - r^{n+1}}{1 - r}$$

$$\lim_{n \rightarrow \infty} S_n = \begin{cases} \frac{1}{1-r}, & |r| < 1 \\ \text{diverge, caso contrário} \end{cases}$$

$$S_n = \sum_{k=0}^n r^k = 1 + r + r^2 + \dots + r^n$$

(Série Geométrica)

$$\therefore \{r^n\}_{n=0}^{\infty} = \lim_{n \rightarrow \infty} \sum_{k=0}^n r^k = \sum_{n=0}^{\infty} r^n = \frac{1}{1-r}, \quad |r| < 1 \quad \forall r \in \mathbb{R}$$

Considerando  $r = -x^2$  e desenvolvendo pelo teorema de diferenciação e integração para séries de potências, temos

$$\begin{aligned} \sum_{n=0}^{\infty} (-1)^n x^{2n} &= \frac{1}{1+x^2} \\ \Rightarrow \sum_{n=0}^{\infty} \left( (-1)^n \int x^{2n} dx \right) + \text{cte} &= \int \frac{1}{1+x^2} dx \\ \Rightarrow \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{2n+1} + \text{cte} &= \arctg x \end{aligned}$$

Para  $x = 0$ , temos  $\arctg x = 0$ ,

$$\therefore \arctg x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{2n+1}, \quad |x| < 1 \quad \forall x \in \mathbb{R} \quad (1)$$

Sendo uma série alternada, podemos reescrever a série com estimativa do erro

$$\begin{aligned} \arctg x &\approx S_n = \sum_{k=0}^n (-1)^k \frac{x^{2k+1}}{2k+1} + R_n(x), \text{ substituindo } x = 1/a \\ &= \underbrace{\sum_{k=0}^n \left( (-1)^k \frac{1}{a^{2k+1}} \cdot \frac{1}{2k+1} \right)}_{P_n(1/a)} + R_n(1/a) \end{aligned}$$

$$P_n\left(\frac{1}{a}\right) = \frac{1}{a} - \frac{1}{3a^3} + \frac{1}{5a^5} - \frac{1}{7a^7} + \frac{1}{9a^9} - \frac{1}{11a^{11}} + \dots + \underbrace{\frac{1}{(2n+1) \cdot a^{2n+1}}}_{b_n} - \underbrace{\frac{1}{(2n+3) \cdot a^{2n+3}}}_{b_{n+1}}$$

Observando o aparecimento de uma série telescópica e otimizando para o desenvolvimento computacional, temos

$$\begin{aligned}
 b_n - b_{n+1} &= \frac{1}{(2n+1) \cdot a^{2n+1}} - \frac{1}{(2n+3) \cdot a^{2n+3}} \\
 &= \frac{(2n+3) \cdot a^{2n+3} - (2n+1) \cdot a^{2n+1}}{(2n+1) \cdot a^{2n+1} \cdot (2n+3) \cdot a^{2n+3}} \\
 &= \frac{(2n+3) \cdot a^2 - (2n+1)}{(2n+1) \cdot (2n+3) \cdot a^{2n+3}}, \text{ usando que } n = 2k \\
 &= \frac{(4k+3) \cdot a^2 - (4k+1)}{(4k+1) \cdot (4k+3) \cdot a^{4k+3}}
 \end{aligned}$$

$$\begin{aligned}
 P_n(1/a) &= \sum_{k=0}^n \frac{(4k+3) \cdot a^2 - (4k+1)}{(4k+1) \cdot (4k+3) \cdot a^{4k+3}} \\
 &= \sum_{\substack{i=0 \\ i+=4}}^n \frac{(i+3) \cdot a^2 - (i+1)}{(i+1) \cdot (i+3) \cdot a^{i+3}}
 \end{aligned}$$

```

generalArctg()
1  long double generalArctg(unsigned long int n, unsigned long int a) {
2      unsigned long int pow, aux_pow, numerator, quotient;
3      long double res;
4
5      pow = a * a * a;
6      aux_pow = a * a * a * a;
7      res = 0;
8
9      int i;
10     for (i = 0; i < n; i+=4) {
11         numerator = (unsigned long int) (i + 3) * (a * a) - (i + 1);
12         quotient = (unsigned long int) (i + 1) * (i + 3) * pow;
13         res += (long double) numerator / (long double) quotient;
14
15         pow *= aux_pow;
16     }
17
18     return res;
19 }
20

```

Desenvolvendo o resto

$$\begin{aligned}
 |R_n| &\leq \frac{1}{a^{2n+3}} \cdot \frac{1}{2n+3} \\
 \frac{1}{a^{2n+3}} \cdot \frac{1}{2n+3} &< \frac{1}{a^{2 \cdot (n+1)}} \cdot \frac{1}{2 \cdot (n+1)} \stackrel{?}{<} \epsilon \\
 \frac{1}{\epsilon} \cdot \frac{1}{2a} &< a^{2 \cdot (n+1)} \cdot (n+1)
 \end{aligned}$$

Como  $\epsilon = 10^{-d}$ , tem-se:  $10^d \cdot \frac{A}{2a} < a^{2 \cdot (n+1)} \cdot (n+1)$ , sendo  $n$  então a quantidade de iterações necessárias e  $A$  o valor de contração/expansão no eixo  $y$  da função.

```

generalErrorHandle()
1  unsigned long int generalErrorHandle(unsigned long int d, unsigned long int A,
2      unsigned long int a) {
3      unsigned long int n, err, small, pow, aux_pow;
4
5      n = 0;
6      err = 1;
7
8      aux_pow = a * a;
9      pow = aux_pow;
10
11     int i;
12     for (i = 0; i < d; i++) {
13         err *= 10;
14     }
15
16     small = err * A / (2 * a);
17
18     while (small >= (unsigned long int) pow * (n + 1)) {
19         pow *= aux_pow;
20         n++;
21     }
22
23     return n;
24 }

```

Porém, utilizando essas funções para  $\pi/4 = \text{arctg}(1)$  alcançamos as limitações das representações utilizadas (`unsigned long int` e `long double`) ao tentar calcular para um  $d$  grande, que no caso dos pontos flutuantes segue a norma específica para formatos e operações para aritmética em sistemas de computadores, IEEE 754<sup>2</sup>[4]. Então é necessário utilizar funções para cálculos de precisão arbitrária, sendo, nesse caso, utilizada a biblioteca *GNU MPFR*, uma biblioteca *C* para cálculos de ponto flutuante de precisão múltipla com arredondamento correto, e baseada na biblioteca *GNU Multiple Precision Arithmetic (GMP)*.

## USANDO GNU MPFR E GNU MP

Usaremos como prefixo ou sufixo nos nomes das funções e das variáveis `fp` para caso use pontos flutuantes de valor arbitrário em sua composição e o prefixo ou sufixo `gmp` para caso use valores inteiros de valor arbitrário.

`generalArctg_mpfr()` Substituindo os tipos tradicionais pelos utilizados nas bibliotecas, temos:

```

1  void generalArctg_fp(const mpz_t n, unsigned long int a, fp_t res) {
2      fp_set_ui(res, 0);
3
4      mpz_t i, i_1, i_3, aux_pow_4, aux_pow_2, pow, a_gmp, numerator, quotient;
5      mpz_inits(i, i_1, i_3, aux_pow_4, aux_pow_2, pow, a_gmp, numerator, quotient,
6          NULL);
7
8      mpz_set_ui(a_gmp, a);
9      mpz_pow_ui(aux_pow_4, a_gmp, 4); // aux = a^4
10     mpz_pow_ui(aux_pow_2, a_gmp, 2); // aux = a^4
11     mpz_pow_ui(pow, a_gmp, 3); // pow = a^3

```

<sup>2</sup>Representado na norma por binary64 dependendo da arquitetura da máquina.

```

11
12     fp_t frac, numerator_fp, quotient_fp;
13     fp_inits(frac, numerator_fp, quotient_fp, NULL);
14
15     for (mpz_set_ui(i, 0); mpz_cmp(i, n) < 0; mpz_add_ui(i, i, 4)) { // for (int
        i = 0; i < n; i+=4) {...}
16         mpz_add_ui(i_1, i, 1); // i + 1
17         mpz_add_ui(i_3, i, 3); // i + 3
18
19         // numerator
20         mpz_mul(numerator, i_3, aux_pow_2); // (i + 3) * a^2
21         mpz_sub(numerator, numerator, i_1); // (i + 3) * a^2 - (i + 1)
22
23         // quotient
24         mpz_mul(quotient, i_1, i_3); // (i + 1) * (i + 3)
25         mpz_mul(quotient, quotient, pow); // (i + 1) * (i + 3) * a^{i+3}
26
27         fp_set_z(numerator_fp, numerator);
28         fp_set_z(quotient_fp, quotient);
29
30         fp_div(frac, numerator_fp, quotient_fp); // frac = numerator / quotient
31         fp_add(res, res, frac); // res += frac
32
33         mpz_mul(pow, pow, aux_pow_4); // pow *= a^4
34     }
35     mpz_clears(i, i_1, i_3, aux_pow_4, aux_pow_2, pow, a_gmp, numerator, quotient
        , NULL);
36     fp_clears(frac, numerator_fp, quotient_fp, NULL);
37 }
38

```

```

generalErrorHandle_gmp()
1 void generalErrorHandle_gmp(unsigned long int d, unsigned long int A, unsigned
    long int a, mpz_t n) {
2     mpz_set_ui(n, 0);
3
4     mpz_t err, aux_pow, pow, aux_cmp;
5     mpz_inits(err, aux_pow, pow, aux_cmp, NULL);
6     mpz_set_ui(err, 1);
7
8     for (int i = 0; i < d; i++) {
9         mpz_mul_ui(err, err, 10);
10    }
11
12    mpz_mul_ui(err, err, A);
13    mpz_div_ui(err, err, 2 * a);
14
15    mpz_set_ui(aux_pow, a);
16    mpz_set(pow, aux_pow);
17
18    mpz_set(aux_cmp, pow);
19    mpz_mul(aux_cmp, aux_cmp, n);
20    while (mpz_cmp(err, aux_cmp) > 0) { // err >= aux_cmp
21        mpz_mul(pow, pow, aux_pow);
22        mpz_set(aux_cmp, pow);
23        mpz_add_ui(n, n, 1);

```



```

24     mpz_mul(aux_cmp, aux_cmp, n);
25 }
26
27     mpz_clears(err, aux_pow, pow, aux_cmp, NULL);
28 }
29

```

## FÓRMULA DE *MACHIN* ORIGINAL

`machin()` Função `machin` resolvendo a *Fórmula de Machin Original*:

$$\frac{\pi}{4} = 4 \cdot \arctg \frac{1}{5} - \arctg \frac{1}{239}$$

```

1 void machin(unsigned long long err, fp_t pi) {
2     fp_t arctg_5;
3     generalArctgWithError_fp(arctg_5, err, 4, 5);
4
5     fp_t arctg_239;
6     generalArctgWithError_fp(arctg_239, err, 1, 239);
7
8     fp_sub(pi, arctg_5, arctg_239);
9     fp_mul_ui(pi, pi, 4);
10
11     fp_clears(arctg_5, arctg_239, NULL);
12 }
13

```

## FÓRMULA DE *KIKUO TAKANO*

`takano()` Função `takano` resolvendo a *Fórmula de Kikuo Takano*:

$$\frac{\pi}{4} = 12 \cdot \arctg \frac{1}{49} + 32 \cdot \frac{1}{57} - 5 \cdot \frac{1}{239} + 12 \cdot \arctg \frac{1}{110.443} \quad (\text{K. Takano, 1982})$$

```

1 void takano(unsigned long long err, fp_t pi) {
2     fp_t arctg_49;
3     generalArctgWithError_fp(arctg_49, err, 12, 49);
4
5     fp_t arctg_57;
6     generalArctgWithError_fp(arctg_57, err, 32, 57);
7
8     fp_t arctg_239;
9     generalArctgWithError_fp(arctg_239, err, 5, 239);
10
11     fp_t arctg_110443;
12     generalArctgWithError_fp(arctg_110443, err, 12, 110443);
13
14     fp_add(pi, arctg_49, arctg_57);
15     fp_sub(pi, pi, arctg_239);
16     fp_add(pi, pi, arctg_110443);
17     fp_mul_ui(pi, pi, 4);

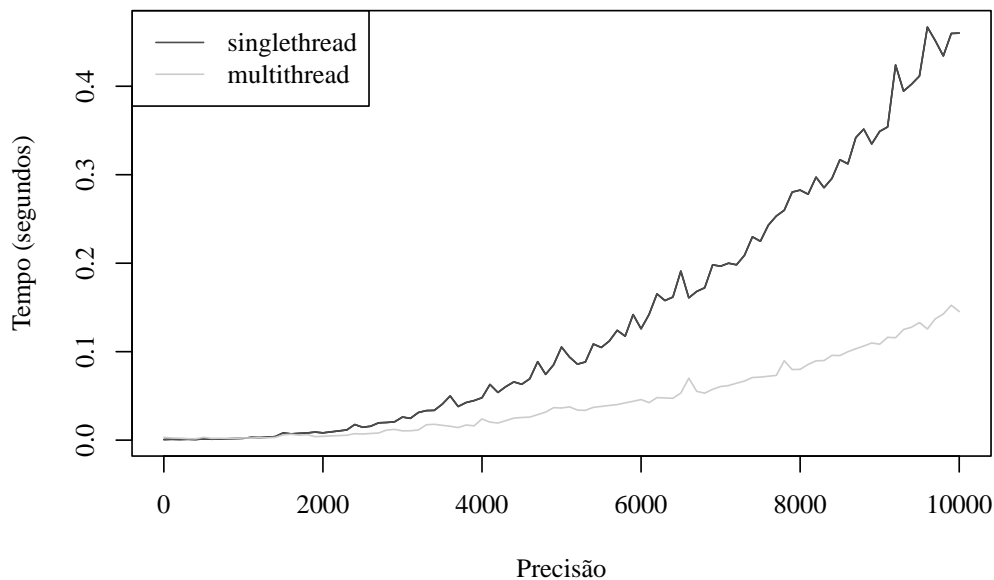
```

```

18
19     fp_clears(arctg_49, arctg_57, arctg_239, arctg_110443, NULL);
20 }
21

```

Gráfico 2: Tempo de evolução com o aumento da precisão do valor de  $\pi$  usando fórmula de Kikuo Takano



Fonte: Elaborado pelo próprio autor

Nota: Usado 8 threads para gerar os dados.

## FÓRMULA DE FREDRIK CARL MÜLERTZ STØRMER

stormer() Função stormer resolvendo a Fórmula de Fredrik Carl Mülertz Størmer:

$$\frac{\pi}{4} = 44 \cdot \arctg \frac{1}{57} + 7 \cdot \frac{1}{239} - 12 \cdot \frac{1}{682} + 24 \cdot \arctg \frac{1}{12.943}$$

(F. C. M. Størmer, 1896)

```

1 void stormer(unsigned long long err, fp_t pi) {
2     fp_t arctg_57;
3     generalArctgWithError_fp(arctg_57, err, 44, 57);
4
5     fp_t arctg_239;
6     generalArctgWithError_fp(arctg_239, err, 7, 239);
7
8     fp_t arctg_682;
9     generalArctgWithError_fp(arctg_682, err, 12, 682);
10
11    fp_t arctg_12943;
12    generalArctgWithError_fp(arctg_12943, err, 24, 12943);
13
14    fp_add(pi, arctg_57, arctg_239);
15    fp_sub(pi, pi, arctg_682);
16    fp_add(pi, pi, arctg_12943);

```

```
17     fp_mul_ui(pi, pi, 4);
18
19     fp_clears(arctg_57, arctg_239, arctg_682, arctg_12943, NULL);
20 }
21
```

## REFERÊNCIAS

- [1] ADVANCED MICRO DEVICES. **AOCC User Guide**. 5.0 ed. 57222, [S. l.]: [S. n.], 2024. Disponível em: <https://docs.amd.com/r/en-US/57222-AOCC-user-guide>. Acesso em: 14 de mar. de 2025.
- [2] BERGGREN, L.; BORWEIN, J.; BORWEIN, P. **Pi: a source book**. Nova Iorque: Springer Science, 1997. ISBN 978-1-4757-2738-8.
- [3] ULAM, S. **Special Issue**. 15, [S. l.]: Los Alamos Science, 1987.
- [4] IEEE STANDARD FOR FLOATING-POINT ARITHMETIC. **IEEE Padrão 754-2019 (Revisão de IEEE 754-2008)**, [S. l.], 2019. DOI 10.1109/IEEESTD.2019.8766229.

## **APÊNDICES**

## A VALOR DE PI

Sendo,  $\pi_n$ ,  $n$  a quantidade de casas decimais, temos, usando fórmula de *Kikuo Takano*:

$\pi_{100} = 3.14159\ 26535\ 89793\ 23846\ 26433\ 83279\ 50288\ 41971\ 69399\ 37510\ 58209\ 74944\ 59230\ 78164\ 06286\ 20899\ 86280\ 34825\ 34211\ 70679$

## B PROBLEMAS COM *MULTITHREADING* E *GNU MPFR*

O uso das bibliotecas OpenMP e Pthreads tiveram incompatibilidades com a função `mpfr_set_z` da biblioteca MPFR por isso o uso da biblioteca GMP para caso deseje usar *multithreading* na aplicação.

...

```
mpfr_set_z(mpfr_number, mpf_number, MPFR_RNDF);  
  
gmp_printf("%Zd\n", mpf_number);  
mpfr_printf("%Rf\n", mpfr_number);
```

```
195453628425706224187742401424363484911678749488065314418725120127483317234...  
195453628425706258249554087436846402840899079169776236471133223956502053110...
```