A constante matemática π (pi) é fundamental em diversas áreas da computação e engenharia. Sua precisão é essencial em aplicações como simulações físicas, gráficos computacionais e inteligência artificial. Nesta atividade, implementei um programa em C que aproxima o valor de π utilizando a série de Leibniz, analisei a influência do número de iterações na precisão do resultado e medi o tempo de execução para diferentes quantidades de termos.

Série de Leibniz

A série de Leibniz para calcular π é definida da seguinte forma:

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

Essa fórmula expressa π como uma soma infinita de frações. No entanto, sua convergência é lenta, ou seja, é necessário um grande número de termos para obter uma aproximação precisa.

Além disso, o aumento das iterações influencia a precisão e o tempo de execução, o que cria a necessidade de balancear eficiência computacional e exatidão em aplicações reais.

Implementação

O código implementado segue a lógica da série de Leibniz e mede o tempo de execução para diferentes números de iterações:

Neste código:

- A função calcularPi implementa a série de Leibniz.
- O main executa testes com diferentes quantidades de iterações e mede o tempo de execução usando clock().
- A precisão da aproximação é avaliada comparando o resultado com M_PI, uma constante definida na biblioteca math.h representando o valor real de π.

Resultados

Os testes foram executados com diferentes números de iterações, e os seguintes resultados foram obtidos:

Iterações	π Aproximado	Erro	Tempo (s)
1.000	3.1405926538	9.999998e-004	0.000000
10.000	3.1414926536	1.000000e-004	0.000000
100.000	3.1415826536	1.000000e-005	0.000000
1.000.000	3.1415916536	1.000000e-006	0.003000
10.000.000	3.1415925536	9.999995e-008	0.026000
100.000.000	3.1415926436	1.000042e-008	0.234000

Com base nesses resultados, observamos:

- O erro reduz conforme aumentamos o número de iterações.
- O tempo de execução cresce significativamente para valores maiores de n, mostrando a relação entre precisão e custo computacional.
- Para valores pequenos de n, a aproximação é ruim, mas melhora consideravelmente conforme iteramos mais.

Isso demonstra que há uma relação entre precisão e eficiência computacional, aspecto crucial em aplicações de alto desempenho.

Reflexão em aplicações reais

O comportamento observado neste experimento reflete um desafio recorrente em diversas aplicações do mundo real: a necessidade de equilibrar precisão e desempenho computacional. Em simulações físicas, por exemplo, cálculos envolvendo constantes matemáticas, como π, são essenciais para modelar fenômenos naturais, desde a trajetória de planetas até o fluxo de fluidos em engenharia. Para garantir previsões mais acuradas, é necessário aumentar o número de cálculos, o que pode resultar em um custo computacional elevado.

Na inteligência artificial, um problema semelhante ocorre no treinamento de modelos complexos. Redes neurais profundas exigem cálculos precisos para ajustar pesos e minimizar erros, mas cada refinamento aumenta o tempo e os recursos necessários para o processamento. Assim como no cálculo de π, há um ponto em que o ganho de precisão se torna marginal em relação ao custo computacional, exigindo técnicas de otimização para balancear esses fatores.

Código usado:

```c
#include <stdio.h>
#include <stdlib.h>
#define _USE_MATH_DEFINES
#include <math.h>
#include <time.h>

double calcularPi(int n) {
    double soma = 0.0;
    for (int k = 0; k < n; k++) {
        soma += (k % 2 == 0 ? 1.0 : -1.0) / (2 * k + 1);
    }
    return soma * 4.0;
}

int main() {
    int iteracoes[] = {1000, 10000, 100000, 1000000, 10000000, 100000000};
    int num_testes = sizeof(iteracoes) / sizeof(iteracoes[0]);

    for (int i = 0; i < num_testes; i++) {
        int n = iteracoes[i];
        clock_t inicio = clock();
        double pi_aproximado = calcularPi(n);
        clock_t fim = clock();

        double erro = fabs(M_PI - pi_aproximado);
        double tempo = (double)(fim - inicio) / CLOCKS_PER_SEC;

        printf("Iteracoes: %d | pi aproximado: %.10f \n Erro: %e | Tempo: %.6f segundos\n\n",
               n, pi_aproximado, erro, tempo);
    }

    return 0;
}
```

Saída dos resultados:

```
JP@PCGilmara MINGW64 ~/Documents/AltoDesempenho/tarefa3 (tarefa3)
$ gcc tarefa3.c -o tarefa3 -lm

JP@PCGilmara MINGW64 ~/Documents/AltoDesempenho/tarefa3 (tarefa3)
$ ./tarefa3
Iteracoes: 1000 | pi aproximado: 3.1405926538
 Erro: 9.999998e-004 | Tempo: 0.000000 segundos

Iteracoes: 10000 | pi aproximado: 3.1414926536
 Erro: 1.000000e-004 | Tempo: 0.000000 segundos

Iteracoes: 100000 | pi aproximado: 3.1415826536
 Erro: 1.000000e-005 | Tempo: 0.000000 segundos

Iteracoes: 1000000 | pi aproximado: 3.1415916536
 Erro: 1.000000e-006 | Tempo: 0.003000 segundos

Iteracoes: 10000000 | pi aproximado: 3.1415925536
 Erro: 9.999995e-008 | Tempo: 0.026000 segundos

Iteracoes: 100000000 | pi aproximado: 3.1415926436
 Erro: 1.000042e-008 | Tempo: 0.234000 segundos
```