Erros

Grupo 39

October 9, 2022

Introdução

Este trabalho prático visa estudar os erros de cálculo em números reais, que tendem a ser um grande problema para computadores, e em geral, em dispositivos capazes de avaliar o campo da aritmétrica.

Durante este trabalho irá ser estudado a forma como os valores aproximados, os limites, as somas infinitas e os erros absolutos afetam os valores calculados.

Pretende-se assim analizar a eficácia do computador e dos próprios métodos usados, e consequentemente, os resultados obtidos via tais computações.

Para além disso, vale a pena frisar que todos os cálculos feitos e todos os valores foram obtidos usando a linguagem **C/C++**.

Epsilon Máquina (macheps)

O **macheps** utilizado pela máquina pode ser obtido pela função:

```
type macheps() {
    type eps = 0.5;
    type macheps;
    while ((1+eps) != 1) {
        macheps = eps;
        eps /= 2;
    }
    return macheps;
}
```

$type \in (float, double, longdouble)$

Sendo assim temos 3 valores que o macheps vai tomar (dependendo do tipo a ser usado):

Float: $macheps = 1.192093e\text{-}07$

Double: $macheps = 2.220446e\text{-}16$

Long Double: $macheps = 1.084202e\text{-}19$

Nota: C só representa 7 algarismos significativos para estes valores, o que pode já ser uma causa de erros futuros.

Exercício 2

Queremos calcular o valor aproximado de:

$$S = 18 \sum_{k=1}^{\infty} \frac{k!^2}{k^2(2k)!}$$

Podemos observar que S é uma série de termos positivos e também sabemos que $\forall n$:

$$a_n = \frac{n!^2}{n^2(2n)!}$$

Passamos a calcular o limite entre a_n e a_{n+1} para saber se a série é convergente ou não. Sendo assim:

$$\lim_{n\to\infty} \frac{a_{n+1}}{a_n}$$

Manipulando a expressão a_{n+1} sabemos que:

$$a_{n+1} = \frac{(n+1)!^2}{(n+1)^2(2(n+1))!} = \frac{n!^2(n+1)^2}{(n+1)^2(2n+2)!} = \frac{n!^2}{(2n+2)!} = \frac{n!^2}{(2n+2)(2n+1)(2n)!} =$$

$$\frac{n!^2}{n^2(2n)!}\frac{n^2}{(2n+2)(2n+1)} = a_n \frac{n^2}{(2n+2)(2n+1)}$$

Assim, o limite entre a_n e a_{n+1} é:

$$\lim_{n\to\infty}\frac{a_{n+1}}{a_n}=\lim_{n\to\infty}\frac{a_n\frac{n^2}{(2n+2)(2n+1)}}{a_n}=\lim_{n\to\infty}\frac{n^2}{(2n+2)(2n+1)}=\lim_{n\to\infty}\frac{n^2}{4n^2+6n+2}=$$

$$\lim_{n\to\infty}\frac{n^2}{n^2(4+\frac{6}{n}+\frac{2}{n^2})}=\frac{1}{4}<1$$

Como o limite entre a_n e a_{n+1} é menor do que 1, pelo critério de D'Alembert, a série é convergente e o erro é majorado por $R_n \leq a_{n+1}\frac{1}{1-L}$. Portanto, basta encontrar um n tal que $a_{n+1}\frac{1}{1-L}<\epsilon$.
Seja $S_n = 18\sum_{k=1}^{n-1}\frac{k!^2}{k^2(2k)!}$ a soma parcial dos n primeiros termos de S e seja $R_n = S - S_n$. Tem-se que:

$$|R_n| \leq a_n\frac{1}{1-\frac{1}{4}}=\frac{4}{3}a_n$$

Logo, basta encontrar um n tal que $\frac{4}{3}\frac{n!^2}{n^2(2n)!}<\epsilon$ e somar os primeiros n termos da série. O programa usado foi o seguinte:

```
// Somatório desde low até high
double calc_sum(double low, double high) {
    double sum = 0;
    while (low <= high) {
        sum += s(low);
        low++;
    }
    return sum;
}

// função responsável pelo calculo de S num determinado n
double s(double n) {
    return (18*pow(factorial(n),2))/(pow(n, 2)*factorial(2*n));
}

// Avalia valores para todos os intervalos de erros epsilon
void error_values() {
    double epsilon[8] = {8,9,10,11,12,13,14,15};
    double n = 1;
    double prev = 1;
```

```
    double sum = 0;

    for (int i = 0; i < 8; i++) {
        while (s(n+1)*(4/3) >= pow(10,-epsilon[i])) n++;
        sum += calc_sum(prev, n);
        prev = n+1;
        printf("exp:%.0f | n:%.0f | sum:%0.19f | error:%0.19f\n",
            epsilon[i], n, sum, pow(M_PI,2)-sum);
    }
}
```

Sabemos que o valor exato de S é π^2. Podemos calcular o valor efetivo do erro absoluto através da expressão $E = |\pi^2 - S|$ para cada valor aproximado de S. Para isso, tomamos o valor exato de π disponível na biblioteca math.h do C. Como esse valor tem 20 casas decimais corretas, o erro absoluto é, no máximo, $5 * 10^{-21}$.

Resultados obtidos:

ϵ	n	S	$\lvert\pi^2 - S\rvert$
10^{-8}	13	9.8696043981327523653	0.0000000029566056270
10^{-9}	14	9.8696044004219967150	0.0000000006673612774
10^{-10}	16	9.8696044010547119285	0.0000000000346460638
10^{-11}	17	9.8696044010814016900	0.0000000000079563023
10^{-12}	19	9.8696044010889334430	0.0000000000004245493
10^{-13}	20	9.8696044010892602927	0.0000000000000976996
10^{-14}	22	9.8696044010893544396	0.0000000000000035527
10^{-15}	23	9.8696044010893579923	0.0000000000000000000

Analisando os valores da tabela, os termos a somar foram-se mantendo relativamente baixos, mesmo para valores de ϵ pequenos. As aproximações obtidas pelo programa mostram que a série converge monotonamente para o valor exato de π^2, logo estão de acordo com o esperado. Como é possível ver na última coluna da tabela, os valores dos erros absolutos são todos menores do que o valor de ϵ dado em cada caso. Logo, foram conseguidas aproximações corretas de S para cada valor de ϵ. Além disso, podemos ver que o erro absoluto é 10^{-16} e, portanto, menor que todos os ϵ dados.

Exercício 3

Queremos estudar o comportamento das seguintes fórmulas para um cálculo aproximado da constante de Euler e:

$$e = \lim_{n\to\infty}(1+\frac{1}{n})^n,\ e = \sum_{k=0}^{\infty}\frac{1}{k!}$$

Para a primeira fórmula **A.** temos que n toma valores: $n = 10^j$, j = 1,2,...,16
Para a segunda fórmula **B.** temos que m toma valores: m = 1,2,...20

De forma a ser mais conviniente, em ambas fórmulas foram calculados 20 parcelas de valores, sendo a seguinte tabela o registo dos resultados obtidos:

j/m	**A.**	**B.**
0	2.0000000000000000000	1.0000000000000000000
1	2.5937424601000018676	2.0000000000000000000
2	2.7048138294215289257	2.5000000000000000000
3	2.7181459268243561844	2.6666666666666665186
4	2.7181459268243561844	2.7083333333333330373
5	2.7182682371975284141	2.7166666666666663410
6	2.7182804691564275146	2.7180555555555554470
7	2.7182816939803724487	2.7182539682539683668
8	2.7182817863957975391	2.7182787698412700372
9	2.7182820308145094756	2.7182815255731922477
10	2.7182820532347875542	2.7182818011463845131
11	2.7182820533571101507	2.7182818261984929009
12	2.7185234960372377522	2.7182818282861687109
13	2.7161100340869008818	2.7182818284467593628
14	2.7161100340870230063	2.7182818284582301871
15	3.0350352065492618436	2.7182818284589949087
16	1.0000000000000000000	2.7182818284590428703
17	1.0000000000000000000	2.7182818284590455349
18	1.0000000000000000000	2.7182818284590455349
19	1.0000000000000000000	2.7182818284590455349
20	1.0000000000000000000	2.7182818284590455349

A partir da tabela acima podemos observar três pontos importantes nos valores apresentados:

1. Geralmente para valores "pequenos" de n as aproximações tendem a ter valores muito longínquos dos valores que queremos obter (**A.** tende a aproximar mais rápido para valores mais baixos, mas vem com um defeito mais à frente falado.);

2. Em geral a fórmula **B.** mostra-se ser mais consistente no aproximamento do valor de Euler;

3. Para $n >= 13$ a fórmula **A.** começa a ter comportamento atípico, chegando mesmo a deixar de ser uma fórmula de cálculo viável a partir de $n >= 16$, onde passa apenas a gerar 1.0 como valor de cálculo, independente do valor de n. Isto é causado pelo facto de que no **C/C++**, é primeiro arredondado o valor de $1+\frac{1}{n}$ antes de se fazer a exponencial, pelo que para $n >= 10^{16}$ esse arredondamento torna-se 1.0000000000000000000, e assim sabendo, $1^n = 1$ independente do valor de n, sabemos também o porquê da estranheza dos valores obtidos.

Conclusão

Erros e arredondamentos irão ser sempre um poblema elementar em dispositvos capazes de fazer aritmétrica, pois por exemplo, um computador nunca será capaz de representar um número infinito (como o π) por mais memória que tenha, sendo apenas capaz de representar o valor como uma aproximação do valor real. Em suma, este trabalho estudou os possíveis problemas que podem ser causados por tais erros e arredondamentos, tendo em conta um ponto de vista computacional.

Repositório com todo o material produzido para o trabalho: Github