Análise Numérica
Relatório 1 - Truncamento de Séries

Grupo 29

José Dias
Luís Pinto
Samuel Neves
Bárbara Gonçalves

6 de Março de 2019

1 Introdução

Com este trabalho, pretendemos calcular a soma de certas séries com um erro inferior a um certo ϵ, dado. Dito isto, alguns pontos a ter em consideração quanto ao mesmo:

1. A linguagem utilizada foi Python e, portanto, os cálculos foram feitos em dupla precisão.
2. A estrutura das respostas foi feita da seguinte forma:
   - Resolução Teórica;
   - Código utilizado;
   - Resposta ao problema;
   - Comentários.

Exercício 1.

Código utilizado:

```
>>> epsilon = 1
>>> while 1+epsilon > 1:
      epsilon /= 2
>>> 2*epsilon

2.220446049250313e-16
```

Resposta ao problema: O valor obtido para o zero da máquina foi $\epsilon = 2.2 \times 10^{-16}$

Comentários:

1. Multiplicamos o valor obtido por 2 porque queremos descobrir o menor número, da máquina, maior que 1. Caso não o fizessemos, para a máquina, $1 + \epsilon = 1$ e nós não queremos isso.
2. Usando o valor ϵ obtido, conseguimos determinar o intervalo à direita em que a máquina identifica os números como 1: $[1; 1 + \epsilon[$. Logo, ao adicionar números que pertencem a esse intervalo, irão ocorrer erros.

Exercício 2.

Resolução Teórica:

Sejam $a_n = \frac{n!^2}{(2n+1)!}$, $S = \sum_{i=0}^{\infty} a_i$ e $S_n = \sum_{i=0}^{n-1} a_i$. Queremos $S_n : |S - S_n| < \epsilon$. Note-se que $\forall n \in \mathbb{N}, a_n > 0$.

$$\text{Como } \lim_{n\to\infty} \frac{a_{n+1}}{a_n} = \lim_{n\to\infty} \frac{\frac{(n+1)!^2}{(2(n+1)+1)!}}{\frac{n!^2}{(2n+1)!}} = \lim_{n\to\infty} \frac{(n+1)^2(2n+1)!}{(2n+2)(2n+3)(2n+1)!} = \frac{1}{4} < 1,$$

Então, pelo critério de D'Alembert, a série $S = \sum_{i=0}^{\infty} a_i$ converge e $|S - S_n| \leq \frac{1}{1-\frac{1}{4}} a_n = \frac{4}{3} a_n$. Basta encontrar $n \in \mathbb{N}$ para o qual $\frac{4}{3} a_n < \epsilon \Rightarrow |S - S_n| < \epsilon$.

Código utilizado:

```
#Definicoes
def a(n):
    return (9/(2*sqrt(3)))*(factorial(n)**2)/factorial(2*n+1)
def prog1(eps):
    n = 0
    s = 0
    while a(n)*(4/3) > eps :
        s += a(n)
        n += 1
    return (s,n)
#resultado
>>> for i in range (8,16):
     print (10**-i,prog1(10**-i))

1e-08 (3.1415926506432688, 14)
1e-09 (3.1415926528764797, 15)
1e-10 (3.141592653547753, 17)
1e-11 (3.1415926535873, 19)
1e-12 (3.1415926535891847, 20)
1e-13 (3.1415926535897567, 22)
1e-14 (3.1415926535897842, 23)
1e-15 (3.1415926535897927, 25)
```

Resposta ao problema:

ϵ	n	S_n
10^{-8}	14	3.1415926506432688
10^{-9}	15	3.1415926528764797
10^{-10}	17	3.141592653547753
10^{-11}	19	3.1415926535873
10^{-12}	20	3.1415926535891847
10^{-13}	22	3.1415926535897567
10^{-14}	23	3.1415926535897842
10^{-15}	25	$3.1415926535897927 \pm 5.5 \times 10^{-15}$

Resposta Final: $S = 3.14159265 \pm 10^{-8}$

Comentários: O valor 5.5×10^{-15} foi obtido da seguinte forma:
Seja ϵ_1 o erro da máquina e ϵ_2 o erro de arredondamento. $\epsilon_1 = 2.2 \times 10^{-16} \Rightarrow \epsilon_2 \leq 25 \times \epsilon_1 = 5.5 \times 10^{-15}$ [1]

Como $\epsilon_2 \ll \epsilon$, o erro de arredondamento é menosprezável e, portanto, apenas foi utilizado no caso $\epsilon = 10^{-15}$ (que é o caso com maior erro de arredondamento devido a ser o caso com mais iterações).

1. Foram utilizadas até 16 casas décimais para observar melhor a evolução da soma com alteração de ϵ;
2. Como podemos observar, S_{14} tem 8 casas decimais coincidentes com $S_i, \forall i > n$, S_{15} tem 9, e por aí adiante, que era o esperado calculando S com os respetivos erros, ϵ.

[1] A resolução teórica do porquê de multiplicarmos por 25 encontra-se no final do trabalho, na secção 2.

Exercício 3.

Resolução Teórica:

Sejam $a_n = 4\frac{(-1)^n}{2n+1}$, $S_n = \sum_{k=0}^{n-1} a_k$. Como $\lim_{n\to\infty} |a_n| = 0, |a_{n+1}| < |a_n|$ e $a_n = (-1)^n|a_n|$, $S = \sum_{n=0}^{\infty} a_n$ converge.

$$\text{Para } |S - S_n| < \epsilon, \text{ basta encontrar } n \in \mathbb{N} : |a_n| < \epsilon \Leftrightarrow |4\tfrac{(-1)^n}{2n+1}| < \epsilon \Leftrightarrow n > \tfrac{2}{\epsilon} - \tfrac{1}{2}.$$

Ou seja, para $k \in \mathbb{N}$, se $\epsilon < 10^{-k} \Rightarrow n > 2 \times 10^k - \frac{1}{2} \Leftrightarrow n \geq 2 \times 10^k$. No ponto de vista computacional, será necessário verificar 2×10^k casos, no mínimo.

Supondo que a máquina calcula 10^7 casos por segundo (experimentalmente, verificámos que apenas calcula cerca de 10^6, em Python, por segundo), obtemos a tabela seguinte:

ϵ	n	Tempo de execução
10^{-8}	2×10^8	20s
10^{-9}	2×10^9	200s
10^{-10}	2×10^{10}	33.3min
10^{-11}	2×10^{11}	333min
10^{-12}	2×10^{12}	55.5h
10^{-13}	2×10^{13}	555h
10^{-14}	2×10^{14}	231.25dias
10^{-15}	2×10^{15}	6 anos

Visto que o tempo de execução é excessivamente demorado, só foi possível verificar os primeiros dois casos.

Código utilizado:

```
#Definicoes
def b(n):
    return 4*(-1)**n/(2*n+1)
def soma_2(eps):
    start=time.time()
    soma=0
    i=20
    while abs(b(i)) > eps:
        i*=10
    for j in range(0,i):
        soma+=b(j)
    return (i,soma,'Tempo execucao:',time.time()-start)

#resultado
>>> for i in [8,9,10]:
      print(soma_2(10**-i))

(200000000, 3.1415926485894077, 'Tempo execucao:', 191.85141611099243)
(2000000000, 3.1415926530880767, 'Tempo execucao:', 2475.890358686447)
```

Resposta ao problema:

ϵ	n	S_n
10^{-8}	2×10^8	$3.1415926485894077 \pm 4.4 \times 10^{-8}$
10^{-9}	2×10^9	$3.1415926530880767 \pm 4.4 \times 10^{-7}$

Resposta Final: $S = 3.1415926 \pm 10^{-7}$

Comentário: O valor 4.4×10^{-k} foi obtido da seguinte forma:
Seja ϵ_1 o ϵ máquina e ϵ_2 o erro de arredondamento. $\epsilon_1 = 2.2 \times 10^{-16} \Rightarrow \epsilon_2 = 2 \times 10^k \times \epsilon_1 = 4.4 \times 10^{k-16}$. [2]
Como ϵ_2 está próximo de ϵ, o erro de arredondamento vai começar a afetar o valor da soma.

Foram utilizadas até 16 casas décimais para observar melhor a evolução da soma com alteração de ϵ. Com esta informação podemos retirar várias conclusões:

1. O erro de arredondamento começa a ser significativo quando tentamos calcular S com um erro inferior a 10^{-8}, pois, a partir das 2×10^8 iterações, o erro de arrondamento começa a ser superior a 10^{-8}.

2. Os valores de $S_i, \forall i > 2 \times 10^8$, podem começar a afastar-se do resultado desejado e, portanto, podemos não ser capazes de encontrar $n \in \mathbb{N} : |S - S_n| < 10^{-8}$.

Exercício 4.

Código utilizado:[2]

```
#Definicoes
def newprog1(eps):
    n = 0
    s = 0
    while a(n)*(4/3) > eps :
        s += a(n)
        n += 1
    return (abs(pi-s),s,n
def newprog2(eps):
    n = 0
    s = 0
    while abs(b(n)) > eps :
        s += b(n)
        n += 1
    return (abs(pi-s),s,n)

#resultado
>>> for i in range(8,16):
      print(newprog1(10**-i))

(2.946524357838598e-09, 3.1415926506432688, 14)
(7.133134083403547e-10, 3.1415926528764797, 15)
(4.204014913966603e-11, 3.141592653547753, 17)
(2.4931168240982515e-12, 3.1415926535873, 19)
(6.084022174945858e-13, 3.1415926535891847, 20)
```

[2]A resolução teórica do porquê de multiplicarmos por 25 encontra-se no final do trabalho, na secção 2.

[2]Não foi feita nenhuma resposta teórica pois tudo o que é necessário resolver analiticamente já foi feito nos exercícios anteriores.

```
(3.6415315207705135e-14, 3.1415926535897567, 22)
(8.881784197001252e-15, 3.1415926535897842, 23)
(4.440892098500626e-16, 3.1415926535897927, 25)

>>> for i in range(8,10):
        print(newprog2(10**-i))

(5.000385439046795e-09, 3.1415926485894077, 200000000)
(5.01716446166256e-10, 3.1415926530880767, 2000000000)
```

Resposta ao problema:

$$a_n = \frac{n!^2}{(2n+1)!}, S_n = \sum_{i=0}^{n} a_i$$

ϵ	n	S_n	$\lvert\pi - S_n\rvert$
10^{-8}	14	$3.1415926506432688 \pm 3.1 \times 10^{-15}$	3×10^{-9}
10^{-9}	15	3.1415926528764797	8×10^{-10}
10^{-10}	17	3.141592653547753	5×10^{-11}
10^{-11}	19	3.1415926535873	3×10^{-12}
10^{-12}	20	3.1415926535891847	7×10^{-13}
10^{-13}	22	3.1415926535897567	4×10^{-14}
10^{-14}	23	3.1415926535897842	9×10^{-15}
10^{-15}	25	3.1415926535897927	5×10^{-16}

$$a_n = 4\frac{(-1)^n}{2n+1}, S_n = \sum_{i=0}^{n} a_i$$

ϵ	n	S_n	$\lvert\pi - S_n\rvert$
10^{-8}	14	$3.1415926485894077 \pm 4.4 \times 10^{-8}$	6×10^{-9}
10^{-9}	15	3.1415926530880767	6×10^{-10}

Comentário: Neste último exercício, era esperado obter um erro para $|\pi - S|$ inferior ao erro ϵ dado, com exceção no segundo caso (Tabela 2).

1. Tal como observado previamente, os dados obtidos foram os esperados o que era de prever visto que o erro de arredondamento não afeta estes valores nas respetivas casas decimais calculadas;
2. Como os resultados foram melhores do que os especulados, na Tabela 2, ou se deve ao facto dos cálculos serem feitos com dupla precisão, em Python, ou simplesmente não houve um impacto tão grande do erro de arredondamento.

2 Resolução teórica do cálculo de arredondamento:

Queremos majorar o erro de arredondamento, $|S_n - \hat{S}_n|$. Seja a'_i o valor que obtemos para a_i na máquina. Sabemos que $\forall i \in \mathbb{N}, |a_i - a'_i| \leq \epsilon = 2.2 \times 10^{-16}$ (epsilon máquina).

$$|S_n - \hat{S}| = \left|\sum_{i=0}^{n-1} a_i - \sum_{i=0}^{n-1} a'_i\right| = \left|\sum_{i=0}^{n-1} a_i - a'_i\right| \leq \sum_{i=0}^{n-1} |a_i - a'_i| \leq \sum_{i=0}^{n-1} \epsilon = n\epsilon \quad \square$$