**Análise de Erros em Séries**

FCUP

Análise Numérica (M2018) 2018/2019

Trabalho de Grupo 1

Ângelo Gomes – 201703990 – MIERSI

Eduardo Morgado – 201706894 – MIERSI

Simão Cardoso – 201604595 – MIERSI

Sónia Rocha – 201704679 – MIERSI

1. **Considerações Iniciais:**

Na realização deste trabalho, utilizamos como linguagem de implementação Python, como tal, tendo em conta que, pontos de virgula flutuante em Python são sempre de precisão dupla, a evolução dos dados não é tão notória. Uma das soluções para este problema seria, através de bibliotecas como *numpy*, forçar o programa a trabalhar apenas em precisão simples, no entanto, todos os cálculos seriam realizados em precisão dupla e só depois convertidos/arredondados para precisão simples, o que acabaria por transmitir erros de arredondamento para os resultados. Sendo assim, durante todo este trabalho, iremos sempre ter em atenção esse fator, fator esse que influencia no cálculo do *epsilon* máquina.

Uma nota importante para utilização dos próximos métodos em Python é a necessidade de configurar a divisão inteira como sendo uma divisão de pontos flutuantes, caso contrário, operações como 1/4=0 e não 0.25, para isso é necessário importar um parâmetro de uma biblioteca: *from \_\_future\_\_ import division*

Todos estes programas podem ser encontrados no Github no repositório da equipa, <https://github.com/thejoblessducks/Trabalho1AN>

1. **Cálculo do *Epsilon* Máquina:**

Antes de qualquer cálculo, é necessário perceber que o *epsilon* máquina é o menor número que, quando somado a 1, produza um resultado diferente de 1, ou seja, é o primeiro valor que, quando somado a 1, não leve a arredondamento, representando a exatidão relativa de um computador. Este valor surge provém da precisão finita de pontos flutuantes, uma vez que, em qualquer computador existem um número limitado de *bits* para representar um número, normalmente 32 *bits* ou 64 *bits*.

Para este trabalho, procuramos diferentes formas de calcular o *epsilon* (eps), as próximas implementações foram as mais simples de realizar:

* eps= numpy.finfo(float).eps
* eps=numpy.spacing(1.0)
* eps=2\*\*(-23)

Tanto a primeira como a segunda implementação, utilizam funções pré-definidas pela biblioteca *numpy*, a primeira produz um cálculo em precisão dupla (), enquanto que a segunda produz um cálculo em precisão simples pela distância de 1 até ao primeiro número diferente de 1 (). A terceira implementação, provem da limitação de representação para pontos de vírgula flutuante. Em qualquer computador com implementação em precisão simples, dispomos apenas de 23 *bits* para a mantissa de um qualquer número, como tal, o menor valor possível a representar é. Para o resto deste trabalho e, tendo em conta, restrições de linguagem, iremos considerar como valor de *epsilon*.

1. **Resolução de Série de Termos Positivos (Exercício 2)**

Qualquer série pode ser decomposta em onde , pelo que, , logo, aproximar a série será encontrar um valor n tal que, , , uma vez que, a série tratada é uma série de termos positivos, podemos aplicar o critério de D’Alembert, o que permite concluir que , onde =L.

Para resolver este problema, primeiro é necessário verificar a convergência e determinar L:

L=1/4<1 pelo que a série converge, logo, basta encontrar n tal que, , é também importante referir que, uma vez que, este resultado irá facilitar o cálculo dos termos da série.

A *Figura 1* representa o pseudocódigo a implementar, e a *Figura 2* a sua respetiva implemenntação em Python (em inglês).



**AproximacaoSerie:**

a->1; n->0; soma->a

a->próximo(a)

Para cada erro fazer

Enquanto |a/(1-L)|>erro fazer

n->n+1

soma->soma+a

a=próximo(a)

escreve(n)

escreve(soma)

Figura 1-Pseudocódigo

A *Figura 3* apresenta a tabela com os

valores dos n’s e das somas para os erros.

Figura 2- Implementação em Python 3.7

|  |  |  |
| --- | --- | --- |
|  | n |  |
|  | 11 | **3.1415926**41 |
|  | 13 | **3.14159265**29 |
|  | 14 | **3.141592653**42 |
|  | 16 | **3.1415926535**79 |
|  | 17 | **3.14159265358**73 |
|  | 19 | **3.141592653589**64 |
|  | 21 | **3.1415926535897**85 |
|  | 22 | **3.14159265358979**18 |

Figura 3- Resultados Exercício 2

Por análise dos resultados, podemos concluir que os valores aproximam o , para cada valor da tabela, a negrito estão os valores corretos do resultado, a vermelho os valores por propagação do erro, seguido pelo erro de aproximação excluído do erro permitido. Para cada erro, , o resultado está representado com m-1 casas decimais corretas. Assumindo agora, por observação dos dados, que a série aproxima podemos calcular o erro relativo desta série, a *Figura 4* apresenta esse erro, podemos então concluir que a série aproxima com exatidão o valor de a partir do elemento n=16.

|  |  |
| --- | --- |
|  |  |
|  | 0.999999996 |
|  | 0.9999999995 |
|  | 0.9999999999 |
|  | 1 |
|  |
|  |
|  |
|  |

Figura 4-Erro relativo de série

1. **Resolução de Série Alternada (Exercício 3)**

Para séries alternadas, aproximar uma série é encontrar n tal que, isto se a série for convergente. Uma série alternada converge se convergir, isto é, se e se , tanto a primeira condição como a segunda são trivialmente verdadeiras , uma vez que, .

A *Figura 5, 6* e *7* representam o pseudocódigo a implementar, a sua implementação em Python e a sua tabela de resultados respetivamente.

**AproximacaoSerie:**

a->1; n->0; soma->a

a->próximo(a)

Para cada erro fazer

Enquanto |a|>erro fazer

n->n+1

soma->soma+a

a=próximo(a)

escreve(n)

escreve(soma)

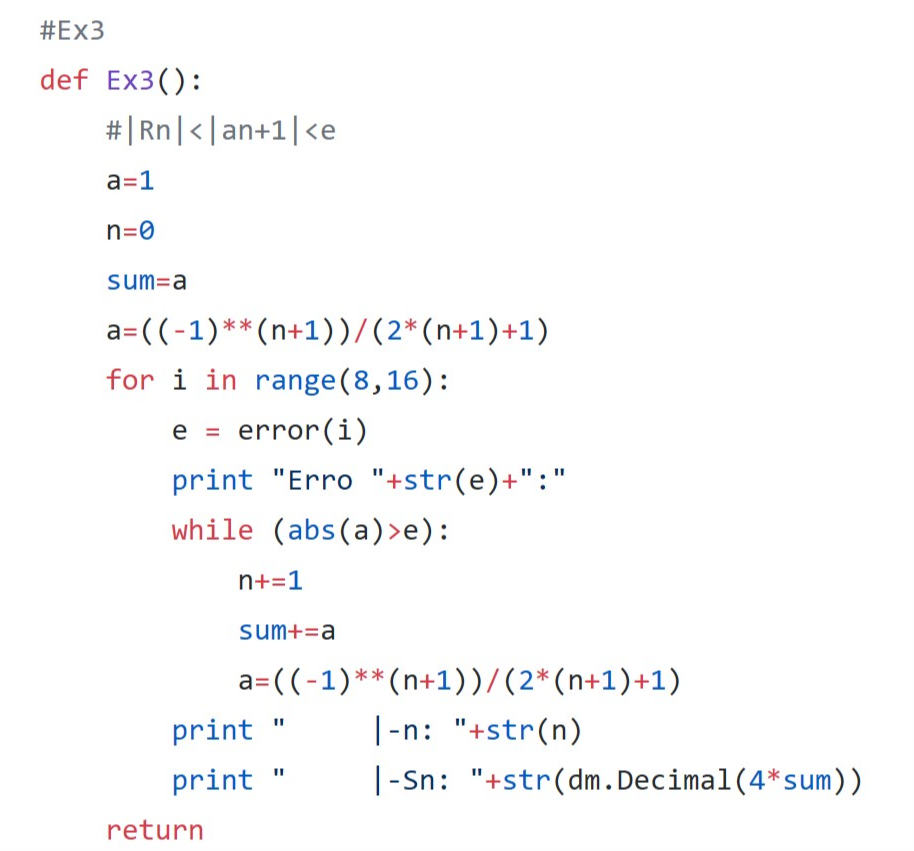


Figura 6-Implementação em Python 3.7

Figura 5-Pseudocódigo Alternada

Algo que notamos ao implementar

este programa, é o facto da série convergir muito

lentamente, o que leva a um cálculo muito lento dos

resultados. Uma das razões poderá ser o facto da

implementação não ser a mais eficiente, ou o facto

dos termos que estamos a somar, serem de tal

forma pequenos que, são menores que o *epsilon* máquina o que não conduz a um aumento “real” da soma. Como tal, a *Figura 7* apenas apresenta alguns resultados.

|  |  |  |
| --- | --- | --- |
|  | n |  |
|  | 49999999 | **3.1415926**33 |
|  | 499999999 | **3.14159265**16 |
|  | Não conseguimos obter resultados por excesso de tempo | |

Figura 7-Tabela de Resultados

Por uma primeira observação dos dados, verificamos que esta implementação converge muito lentamente, quando comparada à anterior, o que conduz a um elevado valor de n, para um valor aproximado menos correto. Quando comparado à tabela do exercício 2, verificamos que a casa de propagação do erro (vermelho) no 2, encontra-se muito mais próxima da sua versão correta para o próximo erro, em comparação com esta implementação. Logo, esta série não é uma boa série para aproximar a S, quando comparada à série anterior.

**5. Aplicação para valores Exatos (Exercício 4)**

Para esta secção, já sabemos *a priori* o valor exato da soma S=π, ou seja, a condição de dos ciclos “Enquanto” para cada série será .

**5.1. Implementação de Exercício 2 para Valor de Série Conhecido**

A *Figura 8* apresenta o código para o problema e a *Figura 9* a tabela dos resultados. Para este problema, consideramos .

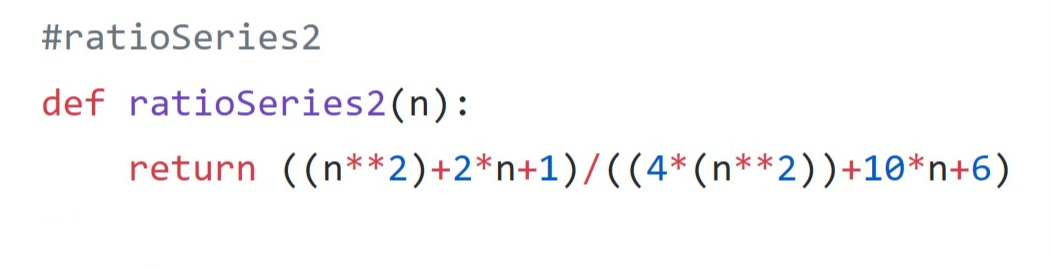
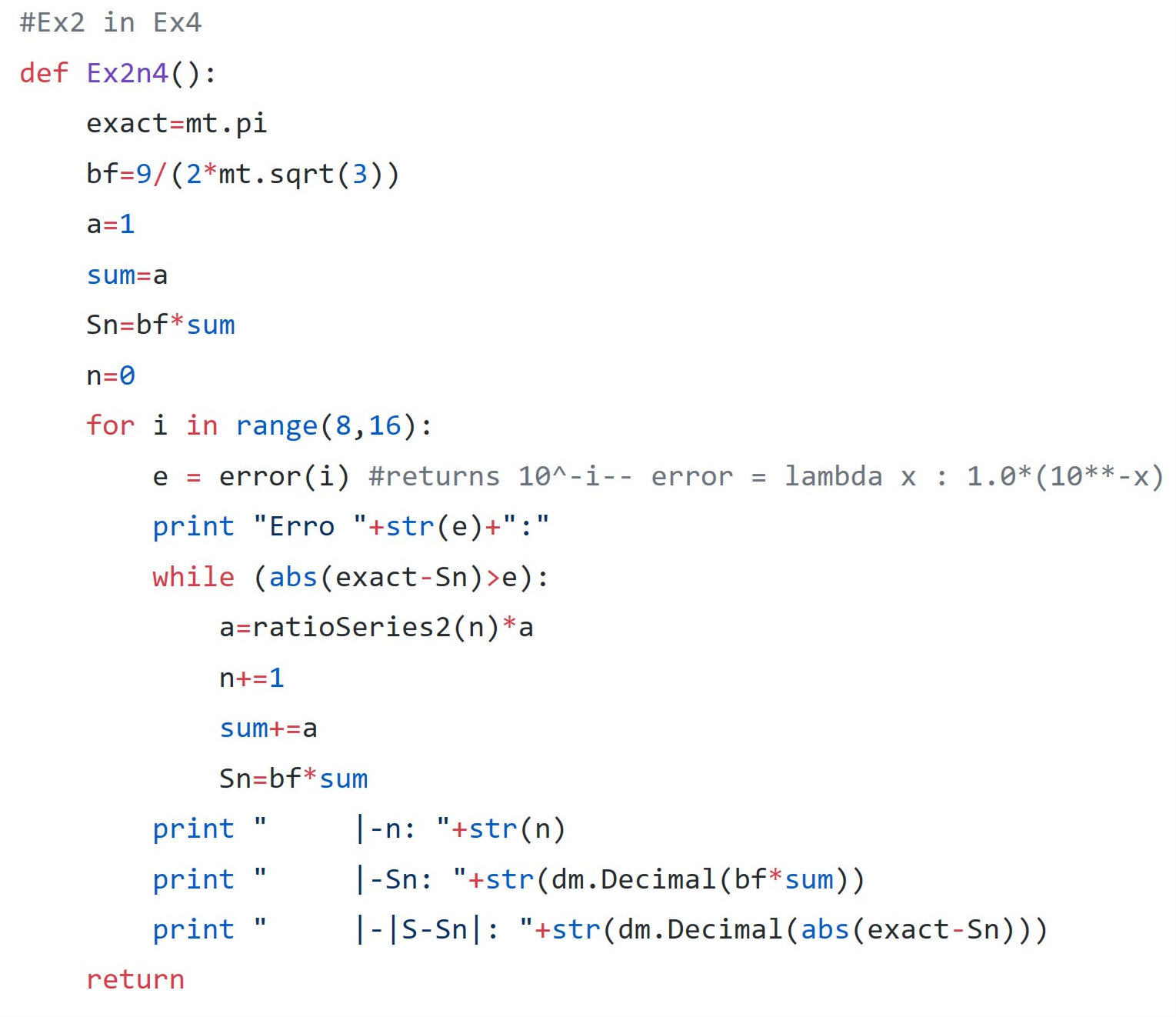
****

Figura 8-Implementação em Python 3.7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | n |  |  |  |
|  | 13 | **3.14159265**06 | **2.94652346**9x | 0.9999999989 |
|  | 14 | **3.141592652**88 | **7.133125201**6x | 0.9999999997 |
|  | 16 | **3.1415926535**48 | **4.2039705050**5x | 1 |
|  | 18 | **3.14159265358**73 | **2.49267273488**8x |
|  | 19 | **3.141592653589**19 | **6.079581282847**4 x |
|  | 21 | **3.1415926535897**57 | **3.5971225997855**1 x |
|  | 22 | **3.14159265358978**51 | **7.99360577730112**7 x |
|  | 24 | **3.141592653589793**56 | **4.440892098500626**2 x |

Figura 9-Tabela de Resultados

Por observação dos dados, esta nova implementação, necessita de mais passos para chegar a uma aproximação inferior ao erro estipulado, no entanto, é mais fidedigna (apresenta mais casas decimais corretas) que a primeira implementação do exercício 2. Através do cálculo do erro relativo, uma vez que sabemos que a série aproxima , podemos concluir que para o valor que consideramos ser exato, a nossa aproximação, atinge um rigor máximo para erros inferiores a , daí podemos assumir que o valor de calculado pela biblioteca *numpy* de Python, é um valor com erro , ou seja, a partir do elemento n=16, a nossa série aproxima com exatidão o valor de representado pelo computador.

**5.2. Implementação de Exercício 3 para Valor de Série Conhecido**

Para este exercício voltamos a deparar-nos com o mesmo problema, o programa não

termina, não chegando à solução para o erro . A *Figura 10* e *Figura 11* apresentam os código e a tabela respetivamente.

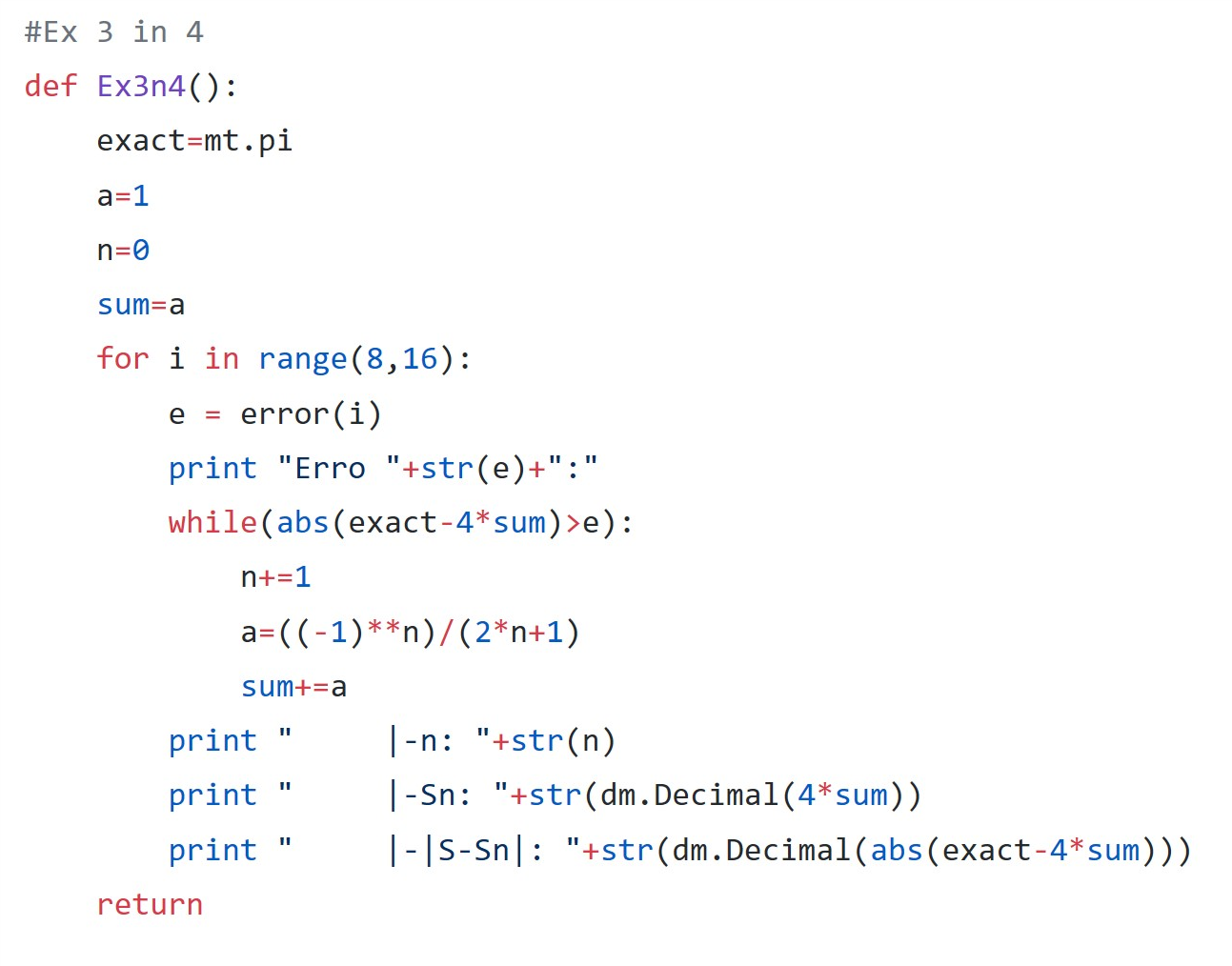


Figura 10-Implementação em Python 3.7

|  |  |  |  |
| --- | --- | --- | --- |
|  | n |  |  |
|  | Não conseguimos obter resultados por excesso de tempo | | |

Figura 10-Tabela