

Universidade de São Paulo

Instituto de Física de São Carlos

Lista 1

Pedro Calligaris Delbem 5255417

Professor: Attilio Cucchieri

Março de 2025

Sumário

1	Exercício 1	2
2	Exercício 2	3
3	Exercício 3	5
4	Exercício 4	6
5	Exercício 5	8
6	Exercício 6	11

1 Exercício 1

Tarefa: Calcular a área de um círculo. Opções: entrada e saída usando teclado e monitor; entrada e saída usando arquivos; calcular a área em uma subrotina.

Código Escrito:

```
1  ! -----
2  ! File: L1-5255417-ex-1.f90
3  !
4  ! Description:
5  !   Computes the area of a circle.
6  !
7  ! Dependencies:
8  !   - None
9  !
10 ! Since:
11 !   - 03/2025
12 !
13 ! Authors:
14 !   - Pedro C. Delbem <pedrodelbem@usp.br>
15 ! -----
16 program circle_area
17
18     !deactivate implicit typing
19     implicit none
20
21     !define variables
22     real radio, area
23
24     !request radius value to the user
25     write(*,*) 'Insert radius value:'
26
27     !read user input
28     read(*,*) radio
29
30     !call calculate_area
31     call calculate_area(radio, area)
32
33     !print result
34     write(*,*) 'Area of the circle is:', area
35
36 contains
37
38     subroutine calculate_area(radio, area)
39
40         !deactivate implicit typing
41         implicit none
42
43         !define variables
44         real, intent(in) :: radio
45         real, intent(out) :: area
46
47         !calculate area
48         area = 4*atan(1.)*radio**2
49
50     end subroutine calculate_area
```

```
51
52 end program circle_area
```

Descrição: O código - por meio de uma mensagem no terminal - pede ao usuário o raio do círculo, calcula a área do mesmo em uma subrotina, e imprime o resultado no terminal.

2 Exercício 2

Tarefa: Testar overflow e underflow em precisão simples e dupla.

Código Escrito:

```
1  ! -----
2  ! File: L1-5255417-ex-2.f90
3  !
4  ! Description:
5  !   Computes overflow and underflow
6  !
7  ! Dependencies:
8  !   - None
9  !
10 ! Since:
11 !   - 03/2025
12 !
13 ! Authors:
14 !   - Pedro C. Delbem <pedrodelbem@usp.br>
15 ! -----
16 program testing_overflow_and_underflow
17
18   implicit none
19
20   real test_overflow, real_overflow, test_underflow,
21     real_underflow
22
23   real*8 test_overflow8, real_overflow8, test_underflow8,
24     real_underflow8
25
26   test_overflow = 1.0
27   test_overflow8 = 1.0
28   call compute_overflow(test_overflow, real_overflow)
29   call compute_overflow8(test_overflow8, real_overflow8)
30
31   test_underflow = 1.0
32   test_underflow8 = 1.0
33   call compute_underflow(test_underflow, real_underflow)
34   call compute_underflow8(test_underflow8, real_underflow8)
35
36   write(*,*) 'Overflow simple real order is: ', real_overflow
37   write(*,*) 'Overflow double real order is: ', real_overflow8
38
39   write(*,*) 'Underflow simple real order is: ', real_underflow
40   write(*,*) 'Underflow double real order is: ', real_underflow8
```

```

39
40 contains
41
42     subroutine compute_overflow(test_overflow,real_overflow)
43
44         implicit none
45
46         real, intent(inout) :: test_overflow
47         real, intent(inout) :: real_overflow
48
49         infinity = huge(1.0_real(4))
50         do while (test_overflow < infinity)
51             write(*,*) test_overflow
52             test_overflow = test_overflow*10
53         end do
54
55     end subroutine compute_overflow
56
57     subroutine compute_overflow8(test_overflow8,real_overflow8)
58
59         implicit none
60
61         real*8, intent(inout) :: test_overflow8
62         real*8, intent(inout) :: real_overflow8
63         real*8 :: infinity
64
65         infinity = huge(1.0_real(8))
66         do while (test_overflow8 < infinity)
67             real_overflow8 = test_overflow8
68             write(*,*) test_overflow8
69             test_overflow8 = test_overflow8*10
70         end do
71
72     end subroutine compute_overflow8
73
74     subroutine compute_underflow(test_underflow,real_underflow)
75
76         implicit none
77
78         real, intent(inout) :: test_underflow
79         real, intent(inout) :: real_underflow
80
81         do while (test_underflow > 3.0E-38)
82             real_underflow = test_underflow
83             write(*,*) test_underflow
84             test_underflow = test_underflow/10
85         end do
86
87     end subroutine compute_underflow
88
89     subroutine compute_underflow8(test_underflow8,real_underflow8)
90
91         implicit none
92
93         real*8, intent(inout) :: test_underflow8
94         real*8, intent(inout) :: real_underflow8
95
96         do while (test_underflow8 > 3.0E-38)

```

```

97         real_underflow8 = test_underflow8
98         write(*,*) test_underflow8
99         test_underflow8 = test_underflow8/10
100     end do
101
102     end subroutine compute_underflow8
103
104 end program testing_overflow_and_underflow

```

Descrição:

3 Exercício 3

Tarefa: Achar a precisão do computador, i.e., o maior número positivo tal que $1 + \epsilon = 1$, usando precisão simples e dupla.

Código Escrito:

```

1  !-----
2  ! File: L1-5255417-ex-3.f90
3  !
4  ! Description:
5  !   Finds computer precision
6  !
7  ! Dependencies:
8  !   - None
9  !
10 ! Since:
11 !   - 03/2025
12 !
13 ! Authors:
14 !   - Pedro C. Delbem <pedrodelbem@usp.br>
15 !-----
16 program computer_precision
17
18     !deactivate implicit typing
19     implicit none
20
21     !define variables
22     real*4 real4, sum4, aux4
23     real*8 real8, sum8, aux8
24
25     !initialize variables
26     real4 = 1.0
27     sum4 = 1.0
28     aux4 = 1.0
29     real8 = 1.0
30     sum8 = 1.0
31     aux8 = 1.0
32
33     !find machine precision
34     do while (sum4+real4 /= sum4) !while sum4+real4 is not equal to
sum4

```

```

35         !divide real4 by 2
36         real4 = aux4
37         aux4 = real4*0.5
38     end do
39
40     !find machine precision
41     do while (sum8+real8 /= sum8) !while sum4+real4 is not equal to
sum4
42         !divide real4 by 2
43         real8 = aux8
44         aux8 = real8*0.5d0
45     end do
46
47     !print results
48     write(*,*) "Machine precision for real4 is:", real4
49     write(*,*) "Machine precision for real8 is:", real8
50
51 end program computer_precision

```

Descrição: O código inicia variáveis de precisão simples e dupla, e divide-as por 2 - sucessivamente até que a soma da mesma com 1 permaneça igual a 1. Além disso utiliza-se uma variável auxiliar para armazenar o valor da precisão - sem que o mesmo se perca quando o valor for menor que a precisão da máquina.

4 Exercício 4

Tarefa: Calcular

$$e^{-x} = 1 - x + x^2/2! - x^3/3! + \dots \quad (1)$$

para $x = 0.1, 1, 10, 100$ e 1000 com um erro menor do que 10^{-8} . Problema: quando truncar a série? É preciso calcular o fatorial explicitamente? Comparar o valor obtido usando a série com o resultado exato.

Código Escrito:

```

1  ! -----
2  ! File: L1-5255417-ex-4.f90
3  !
4  ! Description:
5  !   Computes taylor series for exponential of -x
6  !
7  ! Dependencies:
8  !   - None
9  !
10 ! Since:
11 !   - 03/2025
12 !
13 ! Authors:
14 !   - Pedro C. Delbem <pedrodelbem@usp.br>

```

```

15 !-----
16 program exponential_taylor_series
17
18     !deactivate implicit typing
19     implicit none
20
21     !define variables
22     integer i
23     real*16 x(5), e_x, index, next_term, precision
24
25     !initialize variables
26     x = [0.1,1.0,10.0,100.0,1000.0]
27     precision = 1.0e-8
28
29     do i=1,5
30
31         !initialize variables
32         e_x = 1.0
33         index = 1.0
34         next_term = 1.0
35
36         !call compute_exponential
37         call compute_exponential(x(i), e_x, index, next_term,
38 precision)
39
40         !print result
41         write(*,*) "      x = ", x(i)
42         write(*,*) "      Computed e^(-", x(i), ") = ", e_x
43         write(*,*) "      Real e^(-", x(i), ") = ", exp(-x(i))
44
45     end do
46
47 contains
48
49     subroutine compute_exponential(x, e_x, index, next_term,
50 precision)
51
52         !deactivate implicit typing
53         implicit none
54
55         !define variables
56         real*16, intent(in) :: x
57         real*16, intent(in) :: precision
58         real*16, intent(inout) :: index
59         real*16, intent(inout) :: next_term
60         real*16, intent(out) :: e_x
61
62         !update e_x while the absolute value of next_term is
63         !greater than precision
64         do while (abs(next_term) > precision)
65
66             !compute next term
67             next_term = next_term*(-x)/index
68
69             !update e_x
70             e_x = e_x + next_term
71
72             !update index and factorial

```



```

70         index = index + 1
71
72     end do
73
74     end subroutine compute_exponential
75
76 end program exponential_taylor_series

```

Resultados:

```

pedro@Pedro-Lenovo ~/Documentos/GitHub/quantumcomp/lista1
> $ ./L1-5255417-ex-4.exe
x = 0.100000001490116119384765625000000000
Computed e^(- 0.100000001490116119384765625000000000 ) = 0.904837416707242736562657396953711393
Real e^(- 0.100000001490116119384765625000000000 ) = 0.904837416687646752130945377714961247
x = 1.000000000000000000000000000000000000
Computed e^(- 1.000000000000000000000000000000000000 ) = 0.367879441321281599059376837154614950
Real e^(- 1.000000000000000000000000000000000000 ) = 0.367879441171442321595523770161460873
x = 10.000000000000000000000000000000000000
Computed e^(- 10.000000000000000000000000000000000000 ) = 4.53993527866054627678629605973418051E-0005
Real e^(- 10.000000000000000000000000000000000000 ) = 4.53999297624848515355915155605506104E-0005
x = 100.000000000000000000000000000000000000
Computed e^(- 100.000000000000000000000000000000000000 ) = 95487007.4938040642843786144415534919
Real e^(- 100.000000000000000000000000000000000000 ) = 3.72007597602083596295969580386311846E-0044
x = 1000.000000000000000000000000000000000000
Computed e^(- 1000.000000000000000000000000000000000000 ) = 1.06968282564912128496633499853761240E+0399
Real e^(- 1000.000000000000000000000000000000000000 ) = 5.07595889754945676529180947957433714E-0435

```

Figura 1: Valores de e^{-x} para $x = 0.1, 1, 10, 100$ e 1000 .

Descrição: O código faz um loop onde, em cada iteração, calcula o valor da série para um valor de x diferente. Na subrotina "compute-exponencial" calcula-se a série definindo o próximo termo como a multiplicação do termo anterior por $-x/n$, pois - deste modo - não se faz necessário calcular o fatorial explicitamente. Ademais, interrompe-se a soma quando o termo atual for menor que 10^{-8} garantindo a precisão desejada, uma vez que cada termo da série é menor - em módulo - que o anterior.

Por fim, percebe-se que o resultado esperado foi obtido para todos os casos testados - com exceção dos casos onde o resultado é menor que a precisão.

5 Exercício 5

Tarefa: Considerar a somatória

$$\Sigma(N) = \sum_{n=1}^{2N} (-1)^n \frac{n}{n+1} = - \sum_{n=1}^N \frac{2n-1}{2n} + \sum_{n=1}^N \frac{2n}{2n+1} = \sum_{n=1}^N \frac{1}{2n(2n+1)} \quad (2)$$

e calcular $\Sigma(N)$ para $N = 1, 2, \dots, 10^6$ usando as três fórmulas acima. Comparar os resultados usando precisão simples.

Código Escrito:

```

1 ! -----
2 ! File: L1-5255417-ex-5.f90

```

```

3 !
4 ! Description:
5 !   Computes sums of series
6 !
7 ! Dependencies:
8 !   - None
9 !
10 ! Since:
11 !   - 03/2025
12 !
13 ! Authors:
14 !   - Pedro C. Delbem <pedrodelbem@usp.br>
15 ! -----
16 program exponential_taylor_series
17
18     !deactivate implicit typing
19     implicit none
20
21     !define variables
22     integer N(14), i
23     real sum1, sum2, sum3
24
25     !initialize variables
26     N = [1,2,3,4,5,6,7,9,10**1,10**2,10**3,10**4,10**5,10**6]
27
28     do i=1,14
29
30         !initialize variables
31         sum1 = 0.0
32         sum2 = 0.0
33         sum3 = 0.0
34
35         !compute series
36         call compute_series(sum1, sum2, sum3, N(i))
37
38         !print result
39         write(*,*) N(i), sum1, sum2, sum3
40
41     end do
42
43 contains
44
45     subroutine compute_series(sum1, sum2, sum3, N)
46
47         !deactivate implicit typing
48         implicit none
49
50         !define variables
51         integer i
52         integer, intent(in) :: N
53         real x, sum2a, sum2b
54         real, intent(inout) :: sum1
55         real, intent(inout) :: sum2
56         real, intent(inout) :: sum3
57
58         !initialize partial sums
59         sum2a = 0.0
60         sum2b = 0.0

```

```

61
62     !compute sum1
63     do i=1,N
64
65         !update x
66         x = i
67
68         !compute sums
69         sum1 = sum1 + (-1)**(x)*x/(x+1)
70         sum2a = sum2a + (2.0*x-1)/(2.0*x)
71         sum2b = sum2b + (2.0*x)/(2.0*x+1)
72         sum3 = sum3 + 1/(2.0*x*(2.0*x+1))
73
74     end do
75
76     !!! first serie is actually 1 to 2N? !!!
77
78     !!update sum1
79     !do i=N,2*N
80
81         !!update x
82         !x = i
83
84         !!compute sums
85         !sum1 = sum1 + (-1)**(x)*x/(x+1)
86
87     !end do
88
89     !update sum2
90     sum2 = -sum2a + sum2b
91
92 end subroutine compute_series
93
94 end program exponential_taylor_series

```

Resultados:

```

pedro@Pedro-Lenovo ~/Documentos/GitHub/quanticacomp/lista1
> $ ./L1-5255417-ex-5.exe

```

1	-0.500000000	0.166666687	0.166666672
2	0.166666687	0.216666698	0.216666669
3	-0.583333313	0.240476370	0.240476191
4	0.216666698	0.254365206	0.254365087
5	-0.616666615	0.263456345	0.263455987
6	0.240476251	0.269866943	0.269866258
7	-0.634523749	0.274629116	0.274628162
9	-0.645634830	0.281229973	0.281228602
10	0.263456106	0.283611298	0.283609569
100	0.301927209	0.304382324	0.304371417
1000	0.306355298	0.306640625	0.306603163
10000	0.306808531	0.306640625	0.306800693
100000	0.306856215	0.304687500	0.306800693
1000000	0.306861997	0.312500000	0.306800693

Figura 2: Valores de $\Sigma(N)$ para $N = 1, 2, \dots, 10^6$ usando precisão simples.

Descrição: O código executa um loop que calcula o valor de $\Sigma(N)$ para $N = 1, 2, \dots, 10^6$ usando as três fórmulas - ao chamar a subrotina "compute-sigma" que, por sua vez, inicia um loop onde soma os termos de cada série. Além disso, a segunda soma é calculada com os termos separados que são somados após o fim do loop.

Por fim, nota-se que a primeira série demora mais para convergir do que a demais. Além disso, a terceira série se mostra mais estável que a segunda - sendo então a melhor versão.

6 Exercício 6

Tarefa: Estudar numericamente o erro da aproximação

$$e^{-x} \approx \sum_{n=0}^N \frac{(-x)^n}{n!} \quad (3)$$

em função de N , para diferentes valores de x . Sugestão: faça um gráfico do erro em função de N . O que acontece quando e^{-x} é calculado usando a série $e^x = \sum_{n=0}^N \frac{x^n}{n!}$ e, depois, calculando $1/e^x$?

Código Escrito:

```

1  ! -----
2  ! File: L1-5255417-ex-6.f90
3  !
4  ! Description:
5  !   Computes taylor series for exponential of -x with function of N
6  !
7  ! Dependencies:
8  !   - None
9  !
10 ! Since:
11 !   - 03/2025
12 !
13 ! Authors:
14 !   - Pedro C. Delbem <pedrodelbem@usp.br>
15 ! -----
16 program exponential_taylor_series
17
18     !deactivate implicit typing
19     implicit none
20
21     !define variables
22     integer N(14), i, j
23     real exponential_of_minus_x, exponential_of_x, x(5)
24
25     !initialize variables
26     N = [1,2,3,4,5,6,7,9,10**1,10**2,10**3,10**4,10**5,10**6]
27     x = [0.1,1.0,10.0,100.0,1000.0]
28

```

```

29  open(unit=1, file='exponential_taylor_series.txt', status='
replace')
30
31  do i=1,5
32
33      !initialize variables
34      exponential_of_minus_x = 0.0
35      exponential_of_x = 0.0
36
37      write(1,*) 'x:', x(i)
38
39      do j=1,14
40          !compute series
41          call compute_series(exponential_of_minus_x,
exponential_of_x, N(j), x(i))
42
43          !compute exponential of -x
44          exponential_of_x = 1/exponential_of_x
45
46          !print result
47          write(1,*) 'N:', N(j), exponential_of_minus_x,
exponential_of_x, exp(-x(i))
48
49      end do
50
51  end do
52
53  close(1)
54
55  contains
56
57  subroutine compute_series(exponential_of_minus_x,
exponential_of_x, N, x)
58
59      !deactivate implicit typing
60      implicit none
61
62      !define variables
63      integer i
64      integer, intent(in) :: N
65      real j, next_term_minus_x, next_term_x
66      real, intent(in) :: x
67      real, intent(inout) :: exponential_of_minus_x
68      real, intent(inout) :: exponential_of_x
69
70      !initialize next terms
71      next_term_minus_x = 1.0
72      next_term_x = 1.0
73
74      !compute sum1
75      do i=1,N
76
77          !update j
78          j = i
79
80          !compute next terms
81          next_term_minus_x = next_term_minus_x*(-x)/j
82          next_term_x = next_term_x*x/j

```

```

83
84     !compute sums
85     exponential_of_minus_x = exponential_of_minus_x +
next_term_minus_x
86     exponential_of_x = exponential_of_x + next_term_x
87
88     end do
89
90     end subroutine compute_series
91
92 end program exponential_taylor_series

```

Resultados:

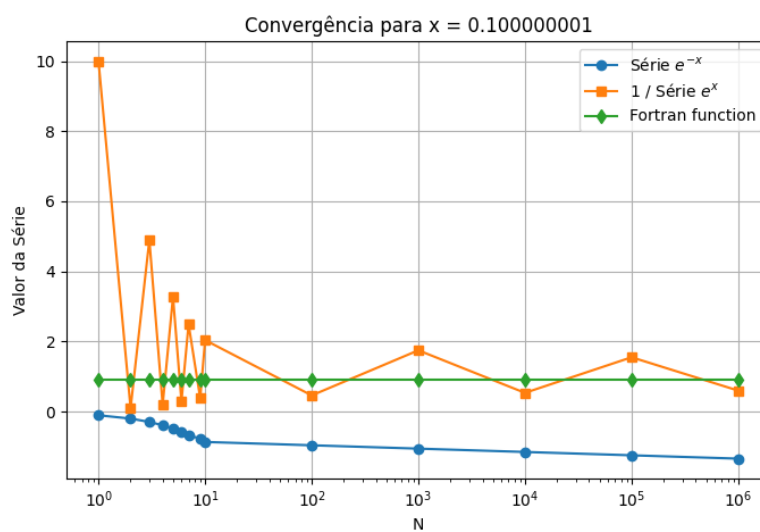


Figura 3: Gráfico do erro em função de N para $x = 0.100000001$.

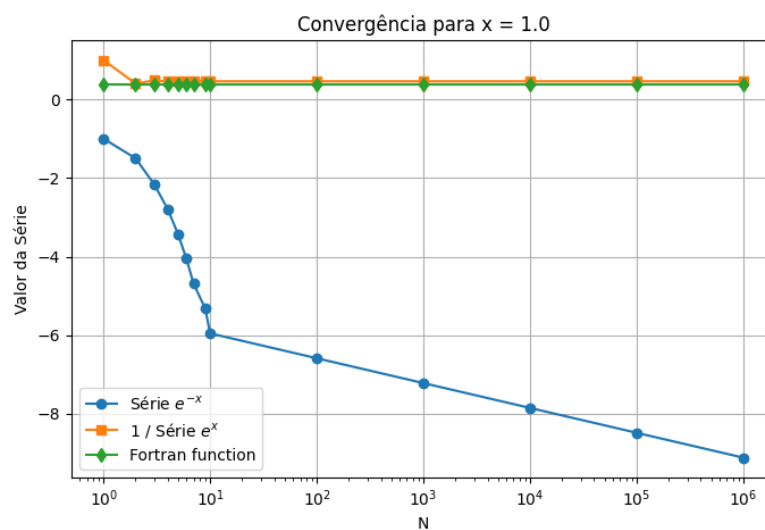


Figura 4: Gráfico do erro em função de N para $x = 1$.

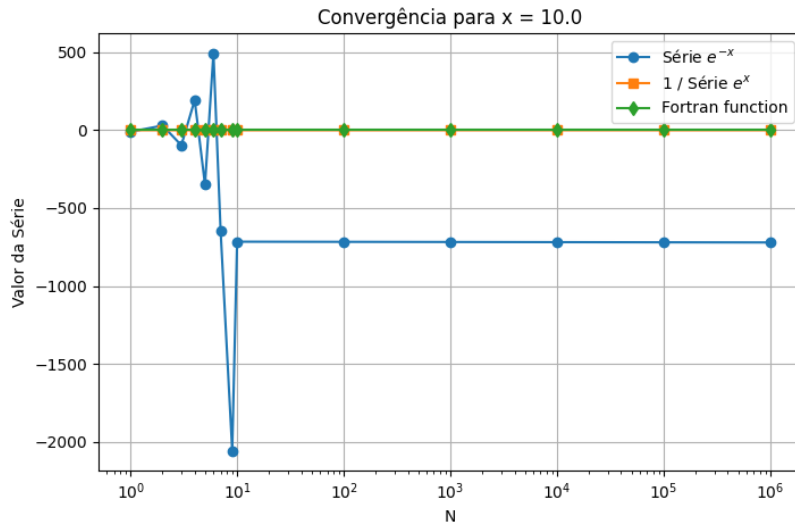


Figura 5: Gráfico do erro em função de N para $x = 10$.

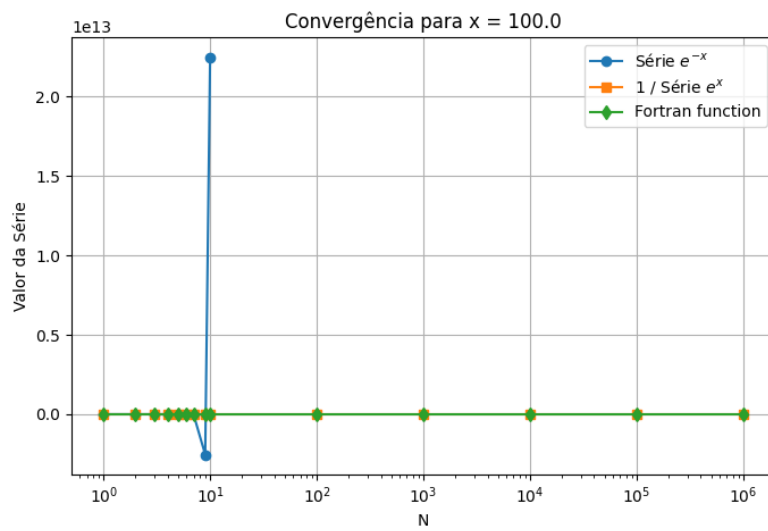


Figura 6: Gráfico do erro em função de N para $x = 100$.

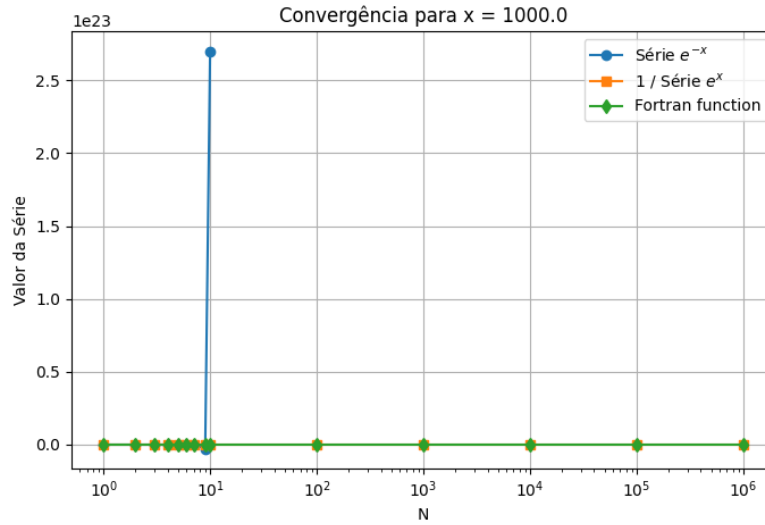


Figura 7: Gráfico do erro em função de N para $x = 1000$.

Descrição: O código inicia um loop onde - para cada valor de - inicia um subloop para cada valor de N que então inicia uma subrotina que computa as séries computando os termos por uma multiplicação do termo anterior.

Por fim, percebe-se - claramente - que calcular a série de e^x e fazer $1/e^x$ é mais preciso do que calcular a série, diretamente, de $1/e^x$.