

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	6
4	Exercício 4	8
5	Exercício 5	10
6	Exercício 6	13

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 radius, area
23
24     !request radius value to the user
25     write(*,*) 'Insert radius value:'
26
27     !read user input
28     read(*,*) radius
29
30     !call calculate_area
31     call calculate_area(radius, area)
32
33     !print result
34     write(*,*) 'Area of the circle is:', area
35
36 contains
37
38     subroutine calculate_area(radius, area)
39
40         !deactivate implicit typing
41         implicit none
42
43         !define variables
44         real, intent(in) :: radius
45         real, intent(out) :: area
46
47         !calculate area (4*atan(1) = pi)
48         area = 4*atan(1.)*radius**2
49
50     end subroutine calculate_area
```

```
51
52 end program circle_area
```

Resultados:

```
pedro@Pedro-Lenovo ~/Documentos/GitHub/quanticacomp/lista1
> $ ./L1-5255417-ex-1.exe
Insert radius value:
1
Area of the circle is: 3.14159274

pedro@Pedro-Lenovo ~/Documentos/GitHub/quanticacomp/lista1
> $ ./L1-5255417-ex-1.exe
Insert radius value:
2
Area of the circle is: 12.5663710

pedro@Pedro-Lenovo ~/Documentos/GitHub/quanticacomp/lista1
> $ ./L1-5255417-ex-1.exe
Insert radius value:
4
Area of the circle is: 50.2654839

pedro@Pedro-Lenovo ~/Documentos/GitHub/quanticacomp/lista1
> $ ./L1-5255417-ex-1.exe
Insert radius value:
8
Area of the circle is: 201.061935
```

Figura 1: Valores de área para os raio de 1, 2, 4 e 8.

A área para o raio = 1 corresponde ao valor de π , de acordo com o esperado. Dobrando o valor do raio espera-se, então, que o valor da área quadruplique e isto foi o que se obteve.

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     !deactivate implicit typing
19     implicit none
20
21     !define variables
22     real test_overflow, real_overflow, test_underflow,
23     real_underflow
24     real*8 test_overflow8, real_overflow8, test_underflow8,
25     real_underflow8
26
27     !initialize overflow variables
28     test_overflow = 1.0
29     test_overflow8 = 1.0
30
31     !call overflows computations
32     call compute_overflow(test_overflow,real_overflow)
33     call compute_overflow8(test_overflow8,real_overflow8)
34
35     !initialize underflow variables
36     test_underflow = 1.0
37     test_underflow8 = 1.0
38
39     !call underflows computations
40     call compute_underflow(test_underflow,real_underflow)
41     call compute_underflow8(test_underflow8,real_underflow8)
42
43     !print results
44     write(*,*)'Overflow simple real is: ', real_overflow
45     write(*,*)'Overflow double real is: ', real_overflow8
46     write(*,*)'Underflow simple real is: ', real_underflow
47     write(*,*)'Underflow double real is: ', real_underflow8
48
49 contains
50
51     !computes single precision overflow
52     subroutine compute_overflow(test_overflow,real_overflow)
53
54         !deactivate implicit typing
55         implicit none
56
57         !define variables
58         real, intent(inout) :: test_overflow
59         real, intent(inout) :: real_overflow
60
61         !compute overflow
62         do while (test_overflow < 2.0*test_overflow) !test_overflow
63             => 2*test_overflow implies overflow
64
65             !update real_overflow variable which saves the value
66             before exceeding the overflow
67             real_overflow = test_overflow
68

```

```

65         !update test_overflow variable to test the next value
66         test_overflow = test_overflow*10
67
68     end do
69
70 end subroutine compute_overflow
71
72 !computes double precision overflow
73 subroutine compute_overflow8(test_overflow8,real_overflow8)
74
75     !deactivate implicit typing
76     implicit none
77
78     !define variables
79     real*8, intent(inout) :: test_overflow8
80     real*8, intent(inout) :: real_overflow8
81
82     !compute overflow
83     do while (test_overflow8 < 2.0*test_overflow8) !
test_overflow8 => 2*test_overflow8 implies overflow
84
85         !update real_overflow8 variable which saves the value
before exceeding the overflow
86         real_overflow8 = test_overflow8
87
88         !update test_overflow8 variable to test the next value
89         test_overflow8 = test_overflow8*10
90
91     end do
92
93 end subroutine compute_overflow8
94
95 !computes single precision underflow
96 subroutine compute_underflow(test_underflow,real_underflow)
97
98     !deactivate implicit typing
99     implicit none
100
101     !define variables
102     real, intent(inout) :: test_underflow
103     real, intent(inout) :: real_underflow
104
105     !compute underflow
106     do while (test_underflow > 0.5*test_underflow) !
test_underflow > 0.5*test_underflow implies underflow
107
108         !update real_underflow variable which saves the value
before exceeding the underflow
109         real_underflow = test_underflow
110
111         !update test_underflow variable to test the next value
112         test_underflow = test_underflow/10
113
114     end do
115
116 end subroutine compute_underflow
117
118 !computes double precision underflow

```

```

119  subroutine compute_underflow8(test_underflow8,real_underflow8)
120
121      !deactivate implicit typing
122      implicit none
123
124      !define variables
125      real*8, intent(inout) :: test_underflow8
126      real*8, intent(inout) :: real_underflow8
127
128      !compute underflow
129      do while (test_underflow8 > 0.5*test_underflow8) !
test_underflow8 > 0.5*test_underflow8 implies underflow
130
131          !update real_underflow8 variable which saves the value
before exceeding the underflow
132          real_underflow8 = test_underflow8
133
134          !update test_underflow8 variable to test the next value
135          test_underflow8 = test_underflow8/10
136
137      end do
138
139  end subroutine compute_underflow8
140
141 end program testing_overflow_and_underflow

```

Resultados:

```

pedro@Pedro-Lenovo ~/Documentos/GitHub/quanticaomp/lista1
> $ ./L1-5255417-ex-2.exe
Overflow simple real is:      1.000000007E+38
Overflow double real is:     9.999999999999981E+307
Underflow simple real is:    1.40129846E-45
Underflow double real is:    9.8813129168249309E-324

```

Figura 2: Valores de overflow e underflow para precisão simples e dupla.

Nota-se que os valores obtidos são coerentes com o esperado.

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
35         sum4
36
37         !define aux4 to save the value before exceeding machine
38         precision
39         real4 = aux4
40
41         !divide real4 by 2 and save the value in aux4
42         aux4 = real4*0.5
43
44     end do
45
46     !find machine precision
47     do while (sum8+real8 /= sum8) !while sum4+real4 is not equal to
48         sum4
49
50         !define aux8 to save the value before exceeding machine
51         precision
52         real8 = aux8
53
54         !divide real4 by 2 and save the value in aux8
55         aux8 = real8*0.5d0
56
57     end do
58
59     !print results
60     write(*,*) "Machine precision for simple precision is:", real4
61     write(*,*) "Machine precision for double precision is:", real8
62
63 end program computer_precision

```


Resultados:

```
pedro@Pedro-Lenovo ~/Documentos/GitHub/quanticacomp/lista1
> $ ./L1-5255417-ex-3.exe
Machine precision for simple precision is: 5.96046448E-08
Machine precision for double precision is: 1.1102230246251565E-016
```

Figura 3: Valores de ϵ para precisão simples e dupla.

Nota-se que os valores obtidos correspondem aos valores esperados.

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, old_term, next_term, precision
24
25     !initialize variables
26     x = [0.1, 1.0, 10.0, 100.0, 1000.0]
27     precision = 1_16*10**(-8)
```

```

28
29     do i=1,5
30
31         !initialize variables
32         e_x = 1_16
33         index = 1_16
34         next_term = 1_16
35         old_term = 1_16
36
37         !call compute_exponential
38         call compute_exponential(x(i), e_x, index, old_term,
next_term, 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, old_term,
next_term, precision)
50
51         !deactivate implicit typing
52         implicit none
53
54         !define variables
55         real*16, intent(in) :: x
56         real*16, intent(in) :: precision
57         real*16, intent(inout) :: index
58         real*16, intent(inout) :: next_term
59         real*16, intent(inout) :: old_term
60         real*16, intent(out) :: e_x
61
62         !compute next term
63         old_term = next_term
64         next_term = next_term*(-x)/index
65
66         !update e_x
67         e_x = e_x + next_term
68
69         !update index and factorial
70         index = index + 1_16
71
72         !update e_x while the absolute value of next_term is
greater than precision
73         do while (abs(next_term-old_term) > abs(precision*e_x))
74
75             !compute next term
76             old_term = next_term
77             next_term = next_term*(-x)/index
78
79             !update e_x
80             e_x = e_x + next_term
81
82             !update index and factorial

```

```

83         index = index + 1_16
84
85     end do
86
87     end subroutine compute_exponential
88
89 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 4: 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     !update sum1
77     do i=N,2*N
78
79         !update x
80         x = i
81
82         !compute sums
83         sum1 = sum1 + (-1)**(x)*x/(x+1)
84
85     end do
86
87     !update sum2
88     sum2 = -sum2a + sum2b
89
90 end subroutine compute_series
91
92 end program exponential_taylor_series

```

Resultados:

```

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

```

1	-0.333333313	0.166666687	0.166666672
2	0.883333385	0.216666698	0.216666669
3	-0.509523690	0.240476370	0.240476191
4	1.05436516	0.254365206	0.254365087
5	-0.569877207	0.263456345	0.263455987
6	1.12700903	0.269866943	0.269866258
7	-0.600371778	0.274629116	0.274628162
9	-0.618771255	0.281229973	0.281228602
10	1.19270074	0.283611298	0.283609569
100	1.29447055	0.304382324	0.304371417
1000	1.30560207	0.306640625	0.306603163
10000	1.30670857	0.306640625	0.306800693
100000	1.30684614	0.304687500	0.306800693
1000000	1.30686092	0.312500000	0.306800693

Figura 5: Valores de $\Sigma(N)$ para $N = 1, 2, \dots, 10^6$ usando precisão simples.

Nota-se que a primeira série demora é mais instável do que as 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,
42 exponential_of_x, N(j), x(i))
43
44         !compute exponential of -x
45         exponential_of_x = 1/exponential_of_x
46
47         !print result
48         write(1,*) 'N:', N(j), exponential_of_minus_x,
49 exponential_of_x, exp(-x(i))
50
51     end do
52
53 end do
54
55 close(1)
56
57 contains
58
59     subroutine compute_series(exponential_of_minus_x,
60 exponential_of_x, N, x)
61
62     !deactivate implicit typing
63     implicit none
64
65     !define variables
66     integer i
67     integer, intent(in) :: N
68     real j, next_term_minus_x, next_term_x
69     real, intent(in) :: x
70     real, intent(inout) :: exponential_of_minus_x
71     real, intent(inout) :: exponential_of_x
72
73     !initialize next terms
74     next_term_minus_x = 1.0
75     next_term_x = 1.0
76
77     !compute sum1
78     do i=1,N
79
80         !update j
81         j = i
82
83         !compute next terms
84         next_term_minus_x = next_term_minus_x*(-x)/j
85         next_term_x = next_term_x*x/j
86
87         !compute sums
88         exponential_of_minus_x = exponential_of_minus_x +
89 next_term_minus_x
90         exponential_of_x = exponential_of_x + next_term_x
91
92     end do

```

```

90     end subroutine compute_series
91
92 end program exponential_taylor_series

```

Resultados:

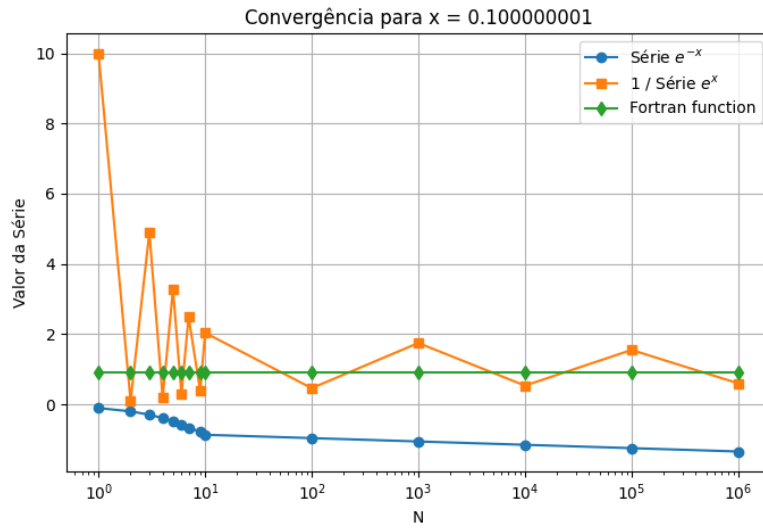


Figura 6: Gráfico do erro em função de N para $x = 0.100000001$.

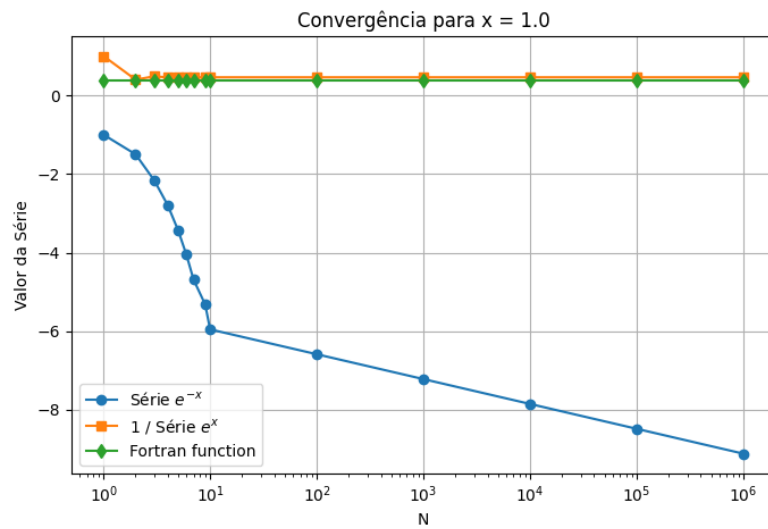


Figura 7: Gráfico do erro em função de N para $x = 1$.

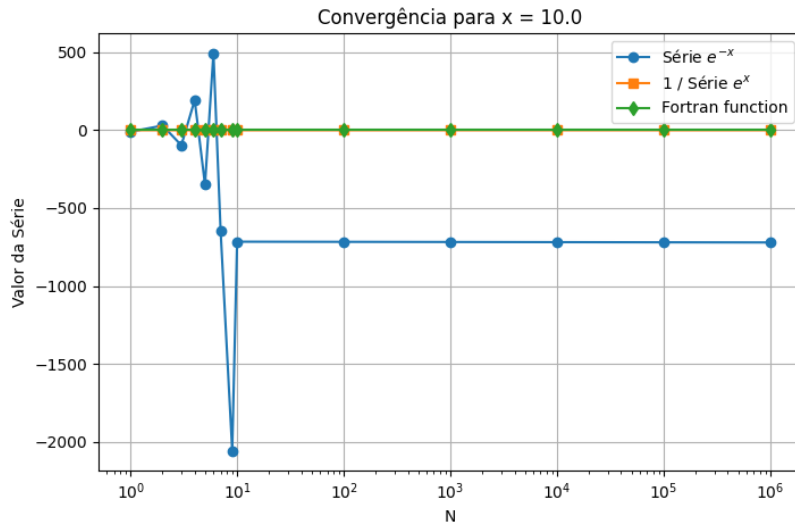


Figura 8: Gráfico do erro em função de N para $x = 10$.

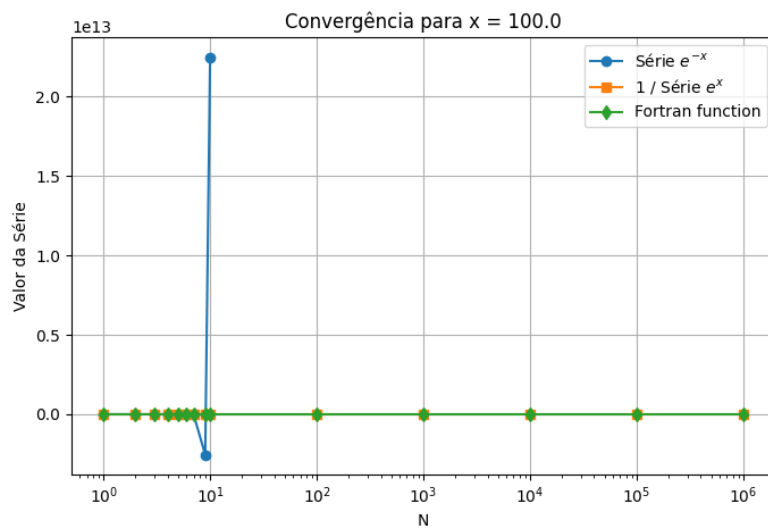


Figura 9: Gráfico do erro em função de N para $x = 100$.



Figura 10: Gráfico do erro em função de N para x = 1000.

Percebe-se - claramente - que calcular a série de e^x e fazer $1/e^x$ o resultado é mais preciso do que calcular a série, diretamente, de e^{-x} . Além disso, é notório que ao calcular a série de e^{-x} diretamente para valores de N muito grande o resultado apresenta uma divergência muito grande do resultado real o que se deve ao fato de que os termos somados se tornam muito pequenos ao ponto de serem menores que o Machine Precision.