2. An accurate implementation of e^x

In [8]:

```
import numpy as np
from math import *

x = 5.5
n = 30  #the number of terms to evaluate in the series

#creating array for storing the successive terms in the series
terms_array = np.zeros(n+1)
terms_array[0] = float(format(1.0,'.5g'))  #the first term is always 1

print(0,terms_array[0])

for i in range(1,31):
    num = 1.0
    denom = 1  #contains evaluated factorial

    for j in range(1,i+1):
        denom = float(format(denom*j,'.5g')) #total of 5 sig-figs always

    for j in range(i):
        num *= x  #this takes care of the left-right product convention
        num = float(format(num,'.5g'))


    #thus numerator and denominator have been calculated separately.
    terms_array[i] = float(format(num/denom,'.5g'))
    print(i,terms_array[i])
```

```
0 1.0
1 5.5
2 15.125
3 27.73
4 38.129
5 41.942
6 38.447
7 30.208
8 20.768
9 12.692
10 6.9805
11 3.4902
12 1.5997
13 0.67679
14 0.26588
15 0.097484
16 0.03351
17 0.010842
18 0.0033128
19 0.00095898
20 0.00026372
21 6.907e-05
22 1.7269e-05
23 4.1297e-06
24 9.4638e-07
25 2.0821e-07
26 4.4043e-08
27 8.9715e-09
28 1.7623e-09
29 3.3422e-10
30 6.1274e-11
```

(a) The above represent the terms of the $e^{5.5}$ till the 31st term. At each step of the calculation, we have approximated that there can be a maximum of five significant figures. Also, the multiplication in calculation of the factorial and the successive powers of x have been computed in a left-right fashion.

In [9]:

```
#left-right summation of the terms in the series

sum_ltr = 0.0 #stores summation value from left-right

for i in range(len(terms_array)):
    sum_ltr += terms_array[i]  #starts from smallest i and goes to largest. Left
-right
    sum_ltr = float(format(sum_ltr,'.5g')) #keeing 5 sig-figs
    print(i,sum_ltr)  #checking for convergence
```

```
0 1.0
1 6.5
2 21.625
3 49.355
4 87.484
5 129.43
6 167.88
7 198.09
8 218.86
9 231.55
10 238.53
11 242.02
12 243.62
13 244.3
14 244.57
15 244.67
16 244.7
17 244.71
18 244.71
19 244.71
20 244.71
21 244.71
22 244.71
23 244.71
24 244.71
25 244.71
26 244.71
27 244.71
28 244.71
29 244.71
30 244.71
```

(b) The summation converges exactly after k=17 when following the left-right summation convention.

In [9]:

```
#the actual value of exponential as given by inbuilt routine
print(exp(5.5))
```

```
244.69193226422038
```

$f_{sig-fig}$ = 244.71

f_{exact} = 244.69193226422038

Magnitude of relative error = $\frac{f_{rounded} - f_{exact}}{f_{exact}}$ the absolute value of which is = 7.383870654188337e-05

In [10]:

```
#right-left summation of the terms in the partial-sum

print(0,terms_array[0])

for i in range(1,31):
    sum_rtl = 0.0 #stores summation value from right-left

    #computing the partial sum
    for j in range(i,-1,-1):
        sum_rtl += terms_array[j]  #starts from largest j and goes to smallest.
 Right-left
        sum_rtl = float(format(sum_rtl,'.5g'))

    #adding the new term
    #sum_rtl += terms_array[i]
    sum_rtl = float(format(sum_rtl,'.5g'))
    print(i,sum_rtl)  #checking for convergence
```

```
0 1.0
1 6.5
2 21.625
3 49.355
4 87.484
5 129.42
6 167.88
7 198.09
8 218.85
9 231.54
10 238.51
11 242.01
12 243.62
13 244.28
14 244.56
15 244.66
16 244.69
17 244.69
18 244.69
19 244.69
20 244.69
21 244.71
22 244.71
23 244.71
24 244.71
25 244.71
26 244.71
27 244.71
28 244.71
29 244.71
30 244.71
```

(c) While summing fron right-left, we find that the series briefly converges after k=16 to the value 244.69. However, it then changes value and converges to 244.71 after k=21. So, yes the step at which the series converges changes when changing our summation convention form left-right to right-left.

If we consider the first instance of convergence to be the convergence value then the relative error is 7.8967222274e-06. Else, for the case of 244.71, it stays the same.

In [11]:

```
#Evaluating exp(-5.5)

#We use the previous results but now change the sign of the odd terms in the ser
ies

sign_arr = [((-1)**n) for n in range(31)] #to be multiplied elementwise to terms
_array

terms_array_neg = sign_arr*terms_array
print(terms_array_neg)
```

```
[ 1.0000e+00 -5.5000e+00  1.5125e+01 -2.7730e+01  3.8129e+01 -4.194
2e+01
  3.8447e+01 -3.0208e+01  2.0768e+01 -1.2692e+01  6.9805e+00 -3.490
2e+00
  1.5997e+00 -6.7679e-01  2.6588e-01 -9.7484e-02  3.3510e-02 -1.084
2e-02
  3.3128e-03 -9.5898e-04  2.6372e-04 -6.9070e-05  1.7269e-05 -4.129
7e-06
  9.4638e-07 -2.0821e-07  4.4043e-08 -8.9715e-09  1.7623e-09 -3.342
2e-10
  6.1274e-11]
```

(d) So now that we have the terms for exp(-5.5), we use this new array to carry out subsequent operations.

In [12]:

```
#left-right summation of the terms in the series

sum_ltr = 0.0 #stores summation value from left-right

for i in range(len(terms_array_neg)):
    sum_ltr += terms_array_neg[i]  #starts from smallest i and goes to largest.
 Left-right
    sum_ltr = float(format(sum_ltr,'.5g'))
    print(i,sum_ltr)  #checking for convergence
```

```
0 1.0
1 -4.5
2 10.625
3 -17.105
4 21.024
5 -20.918
6 17.529
7 -12.679
8 8.089
9 -4.603
10 2.3775
11 -1.1127
12 0.487
13 -0.18979
14 0.07609
15 -0.021394
16 0.012116
17 0.001274
18 0.0045868
19 0.0036278
20 0.0038915
21 0.0038224
22 0.0038397
23 0.0038356
24 0.0038365
25 0.0038363
26 0.0038363
27 0.0038363
28 0.0038363
29 0.0038363
30 0.0038363
```

d(i) Therefore, on computing $e^{-5.5}$ in a left-right summation convention, we find that the value converges to 0.0038363 after k=25.

In [15]:

```
#right-left summation of the terms in the partial-sum

print(0,terms_array[0])

for i in range(1,31):
    sum_rtl = 0.0 #stores summation value from right-left

    #computing the partial sum
    for j in range(i,-1,-1):
        sum_rtl += terms_array_neg[j]  #starts from largest j and goes to smalle
st. Right-left
        sum_rtl = float(format(sum_rtl,'.5g'))

    print(i,sum_rtl)  #checking for convergence
```

```
0 1.0
1 -4.5
2 10.625
3 -17.105
4 21.024
5 -20.918
6 17.529
7 -12.679
8 8.089
9 -4.603
10 2.377
11 -1.113
12 0.487
13 -0.19
14 0.076
15 -0.021
16 0.012
17 0.001
18 0.005
19 0.004
20 0.004
21 0.004
22 0.004
23 0.004
24 0.004
25 0.004
26 0.004
27 0.004
28 0.004
29 0.004
30 0.004
```

d(ii) In computing $e^{-5.5}$ for the right-left convention, we find that the value converges to 0.004 after k=19.

In [17]:

```
#left-right summation of the terms in the series

for j in range(len(terms_array_neg)):
    sum_ltr_p = 0.0 #stores positive terms summation value from left-right
    sum_ltr_n = 0.0 #stores negative terms summation value from left-right

    #summing the positive terms left-right
    for i in range(0,j+1,2):
        sum_ltr_p += terms_array_neg[i]  #starts from smallest i and goes to lar
gest. Left-right
        sum_ltr_p = float(format(sum_ltr_p,'.5g'))

    #summing the negative terms left-right
    for i in range(1,j+1,2):
        sum_ltr_n += terms_array_neg[i]  #starts from smallest i and goes to lar
gest. Left-right
        sum_ltr_n = float(format(sum_ltr_n,'.5g'))

    print(j,float(format(sum_ltr_p + sum_ltr_n,'.5g')))  #checking for convergen
ce
```

```
0 1.0
1 -4.5
2 10.625
3 -17.105
4 21.024
5 -20.918
6 17.529
7 -12.679
8 8.09
9 -4.6
10 2.38
11 -1.11
12 0.49
13 -0.19
14 0.08
15 -0.02
16 0.01
17 0.0
18 0.0
19 0.0
20 0.0
21 0.0
22 0.0
23 0.0
24 0.0
25 0.0
26 0.0
27 0.0
28 0.0
29 0.0
30 0.0
```

d(iii) When computing the positive and negative terms separately using the left-right convention and putting them together, we find that the value converges to 0.0 after k=17.

In [23]:

```
#right-left summation of the terms in the series

for j in range(len(terms_array_neg)):
    sum_rtl_p = 0.0 #stores positive terms summation value from right-left
    sum_rtl_n = 0.0 #stores negative terms summation value from right-left

    if(j%2 == 0):
        j_even = j
        j_odd = j-1
    else:
        j_even = j-1
        j_odd = j

    #summing the positive terms right-left
    for i in range(j_even,-1,-2):
        sum_rtl_p += terms_array_neg[i]  #starts from smallest i and goes to lar
gest. Left-right
        sum_rtl_p = float(format(sum_rtl_p,'.5g'))
        #print('e',j)

    #summing the negative terms right-left
    for i in range(j_odd,-1,-2):
        sum_rtl_n += terms_array_neg[i]  #starts from smallest i and goes to lar
gest. Left-right
        sum_rtl_n = float(format(sum_rtl_n,'.5g'))
        #print('o',i)


    print(j,float(format(sum_rtl_p + sum_rtl_n,'.5g')))  #checking for convergen
ce
```

```
0 1.0
1 -4.5
2 10.625
3 -17.105
4 21.024
5 -20.918
6 17.529
7 -12.679
8 8.09
9 -4.6
10 2.37
11 -1.12
12 0.49
13 -0.19
14 0.07
15 -0.03
16 0.0
17 -0.01
18 0.01
19 0.01
20 0.01
21 0.01
22 0.01
23 0.01
24 0.01
25 0.01
26 0.01
27 0.01
28 0.01
29 0.01
30 0.01
```

d(iv) On adopting the right-left convention in computing the positive and negative terms separately, we find that the value of $e^{-5.5}$ converges to 0.01 after k=18.

Comparing the magnitude of relative errors:

Method (i): Summation from left to right of all the terms : 0.0612883

Method (ii): Summation from right to left of all the terms : 0.02123227

Method (iii): Summation from left to right of positive and negative terms separately : 1.0

Method (iv): Summation from right to left of positive and negative terms separately : 1.446919

This shows that adding all the terms from right to left incurs smaller error. However, when handling positive and negative terms separately, we find that adding right to left results in larger error. This can be attributed to the fact that in the latter case the final subtraction occured between numbers that were large (by orders of magnitude) than the case where we did not treat positive and negative numbers separately.

(e) In the previous cases, the poor values of $e^{-5.5}$ we due to the fact that there were repeated instances of catastrophic cancellations due to the excessive number of subtraction operations being carried out. Thus, the goal of the new alternate algorithm must be to minimize or avoid the implementation of subtraction operations.

Now, we know that division does not result in any bombardment of errors (infact a rigorous analysis shows that for division of numbers with same sign, it leads to subtraction of the errors. Thus, making way for a division operation instead of subtractions would be a nice cure.

Therefore, we try the following algorithm: (a) Compute $e^{5.5}$ in the left-right or right-left way (as they are not too different in terms of the values they converge to, (b) Divide 1 by the computed value of $e^{5.5}$, i.e, $e^{-5.5} = \frac{1.0}{e^{5.5}}$.

In [31]:

```
#right-left summation of the terms in the partial-sum

print(0,terms_array[0])

for i in range(1,31):
    sum_rtl = 0.0 #stores summation value from right-left

    #computing the partial sum
    for j in range(i,-1,-1):
        sum_rtl += terms_array[j]  #starts from largest j and goes to smallest.
 Right-left
        sum_rtl = float(format(sum_rtl,'.5g'))

    #adding the new term
    #sum_rtl += terms_array[i]
    sum_rtl = float(format(sum_rtl,'.5g'))
    print(i,float(format(1.0/sum_rtl,'.5g')))  #checking for convergence
```

```
0 1.0
1 0.15385
2 0.046243
3 0.020261
4 0.011431
5 0.0077268
6 0.0059566
7 0.0050482
8 0.0045693
9 0.0043189
10 0.0041927
11 0.0041321
12 0.0041048
13 0.0040937
14 0.004089
15 0.0040873
16 0.0040868
17 0.0040868
18 0.0040868
19 0.0040868
20 0.0040868
21 0.0040865
22 0.0040865
23 0.0040865
24 0.0040865
25 0.0040865
26 0.0040865
27 0.0040865
28 0.0040865
29 0.0040865
30 0.0040865
```

The value of $e^{-5.5}$ computed from inbuilt function = 0.004086771438464067, and the value obtained from this alternate algorithm is = 0.0040865. The relative error = 6.64188022633233e-05, which is much better than the previous cases that we explored.

Therefore, this is indeed a better algorithm to get a very accurate value of $e^{-5.5}$.