HW-8

Problem 1 :

In this problem , we have to find the following integrals : $ a) e^x $ $ b) sin(1/x)$ $ c) x^3 $ using simpson, adaptive trapezoid, trapezoid algorithms. a) To integrate the function : e^x . To use Simpson algorithm we use the codes used practiced in class (CP1__CalculusUtilityFunctions/integrals.py) and do the necessary imports:

```
from integrals import simpson as ss
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
import math
from scipy.integrate import quad
# %load_ext pycodestyle_magic
# %pycodestyle_on
```

First We check the function e^x with boundary [0,10] with subintervals 10.

Here I am definig the function rather than using "Lambda" because its recommended by linting tool. for accuracy I will use quad from scipy.

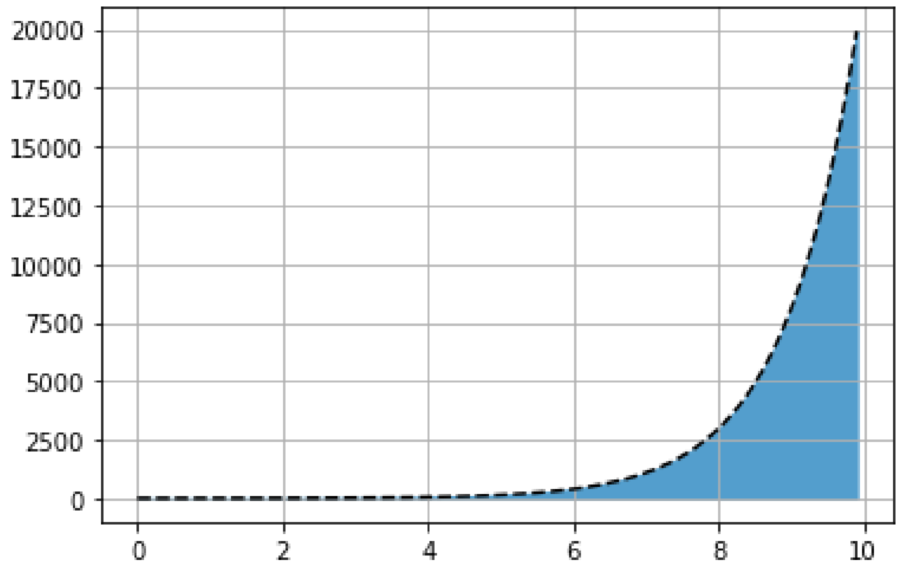
```
def f(x): return np.exp(x)

sum = simpson(f, 0, 10, 10)
print('The sum is: % d ' % sum)
x = np.arange(0, 10, 0.1)
y = np.exp(x)

# for accuracy measurement.
res, err = quad(f, 0, 10)
print("The actual numerical result is {:f} (+-{:g})"
    .format(res, err))
print("Accuracy: ",np.abs(sum-res)/res)

plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

```
The sum is:  22134
The actual numerical result is 22025.465795 (+-6.23939e-10)
Accuracy:  0.004957180659324048
```



Second: We check with the boundary $[0, 2\pi]$

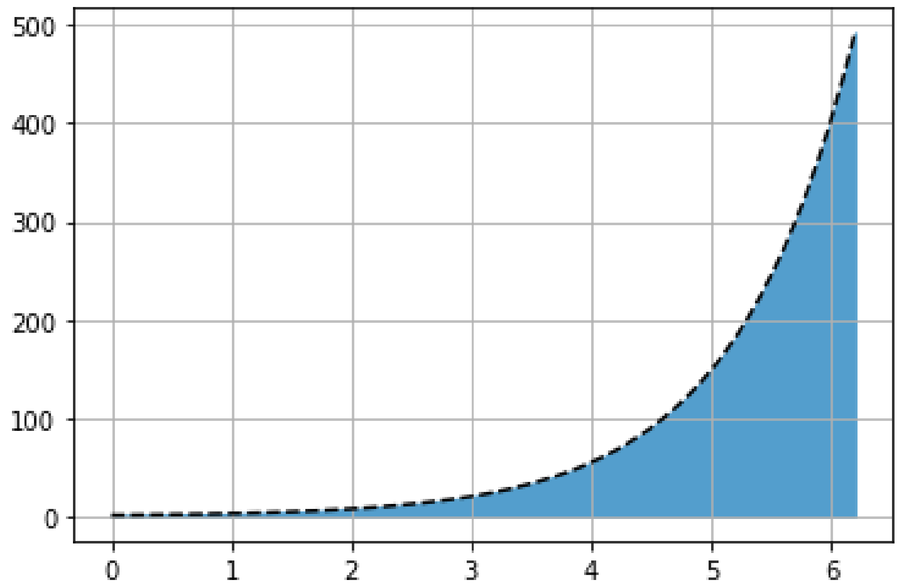
```
def f(x): return np.exp(x)


sum = simpson(f, 0, 2*(math.pi), 10)
print('The sum is: % d ' % sum)
x = np.arange(0, 2*(math.pi), 0.1)
y = np.exp(x)

# for accuracy measurement.
res, err = quad(f, 0, 2*(math.pi))
print("The actual numerical result is {:f} (+-{:g})"
    .format(res, err))
print("Accuracy: ",np.abs(sum-res)/res)

plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

```
The sum is:  534
The actual numerical result is 534.491656 (+-5.93405e-12)
Accuracy:  0.0008267862704823464
```

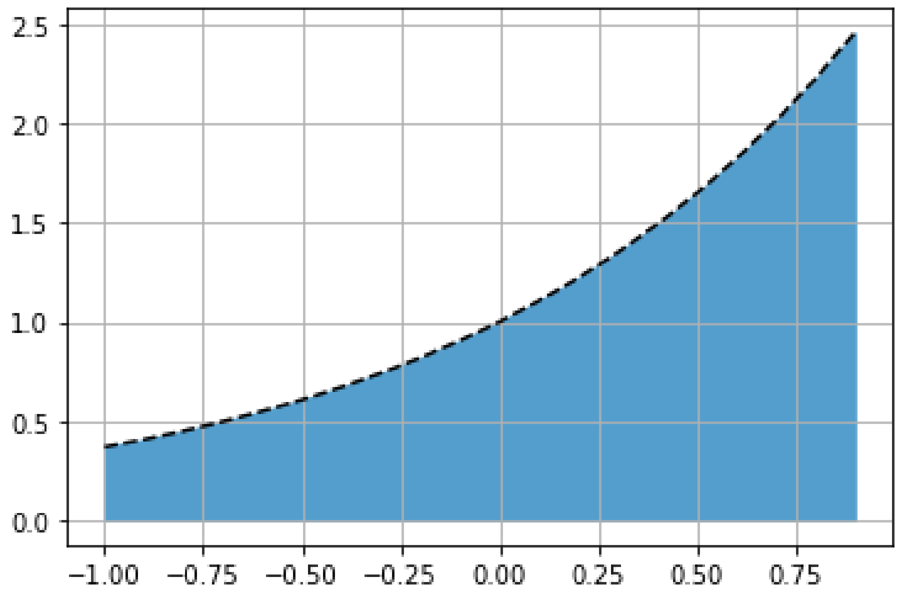


Finaly we check with boudary [-1,1]

```
def f(x): return np.exp(x)


sum = simpson(f, -1, 1, 10)
print('The sum is: % d ' % sum)
x = np.arange(-1, 1, 0.1)
y = np.exp(x)
# for accuracy measurement.
res, err = quad(f, -1, 1)
print("The actual numerical result is {:f} (+-{:g})"
    .format(res, err))
print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

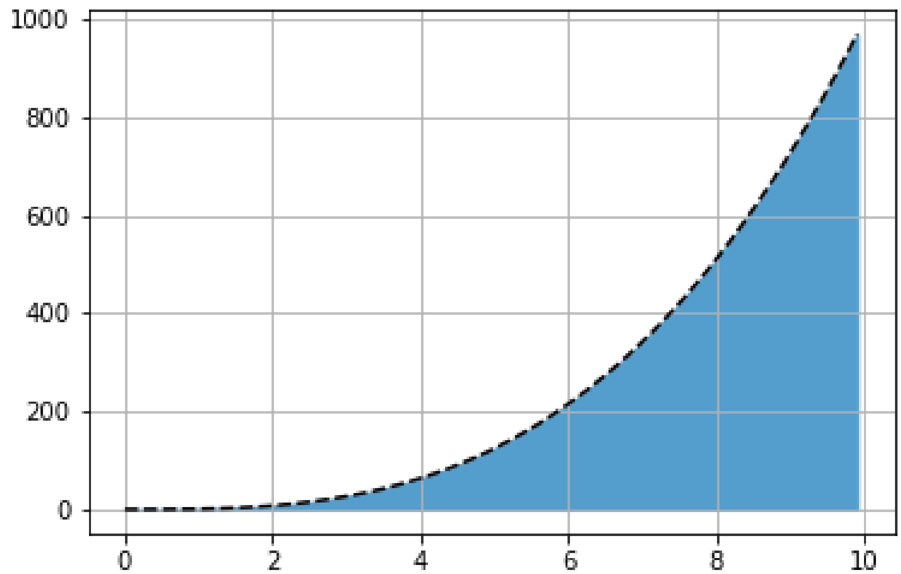
```
The sum is:  2
The actual numerical result is 2.350402 (+-2.60947e-14)
Accuracy:  8.846737900270168e-06
```



For the fucntion x^3, we check with boudary [0,10]

```
def f(x): return x ** 3
sum = simpson(f, 0, 10, 10)
print('The sum is: % d ' % sum)
x = np.arange(0, 10, 0.1)
y = x**3
# for accuracy measurement.
res, err = quad(f, 0, 10)
print("The actual numerical result is {:f} (+-{:g})"
    .format(res, err))
print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

```
The sum is:  2500
The actual numerical result is 2500.000000 (+-2.77556e-11)
Accuracy:  1.818989403545856e-16
```

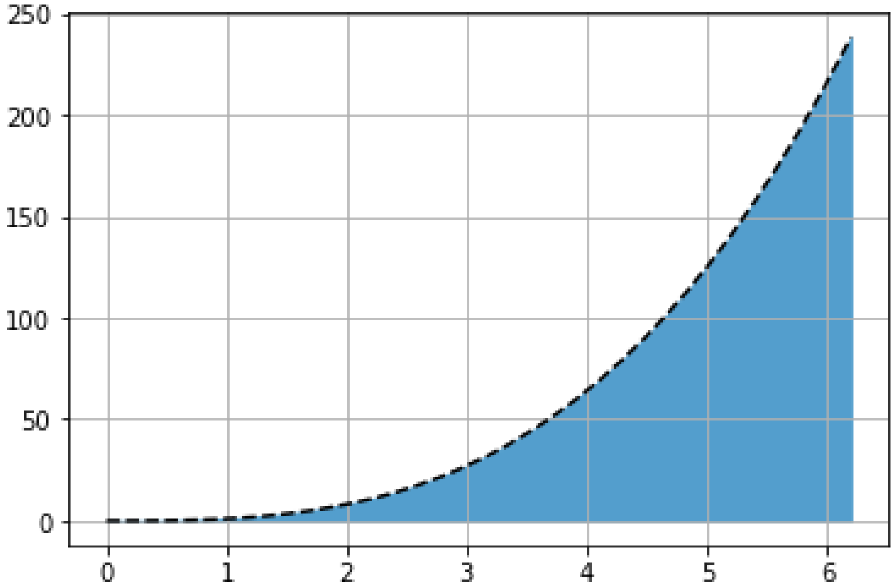


Second: We check with the boundary $[0, 2\pi]$

```
def f(x): return x ** 3


sum = simpson(f, 0, 2*(math.pi), 10)
print('The sum is: % d ' % sum)
x = np.arange(0, 2*(math.pi), 0.1)
y = x**3
# for accuracy measurement.
res, err = quad(f, 0, 2*(math.pi))
print("The actual numerical result is {:f} (+-{:g})"
     .format(res, err))
print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

```
The sum is:  389
The actual numerical result is 389.636364 (+-4.32583e-12)
Accuracy:  2.91776764660322e-16
```

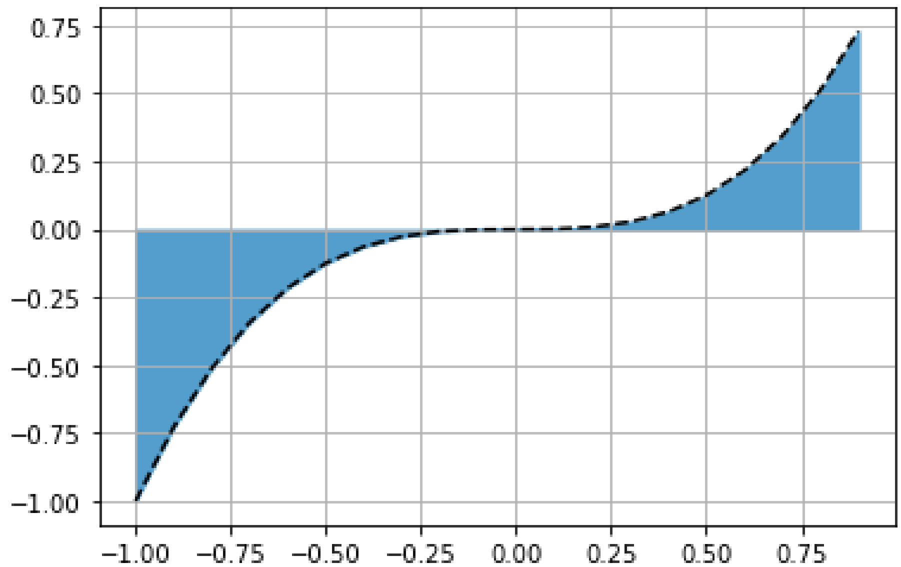


Finaly we check with boudary [-1,1]

```
def f(x): return x ** 3
sum = simpson(f, -1, 1, 10)
print('The sum is: % d ' % sum)
x = np.arange(-1, 1, 0.1)
y = x**3
# for accuracy measurement.
res, err = quad(f, -1, 1)
print("The actual numerical result is {:f} (+-{:g})"
    .format(res, err))
print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

```
The sum is:  0
The actual numerical result is 0.000000 (+-5.55121e-15)
Accuracy:  inf
```

```
/tmp/ipykernel_50/1030297780.py:10: RuntimeWarning: divide by zero encountered in double_sca
  print("Accuracy: ",np.abs(sum-res)/res)
```



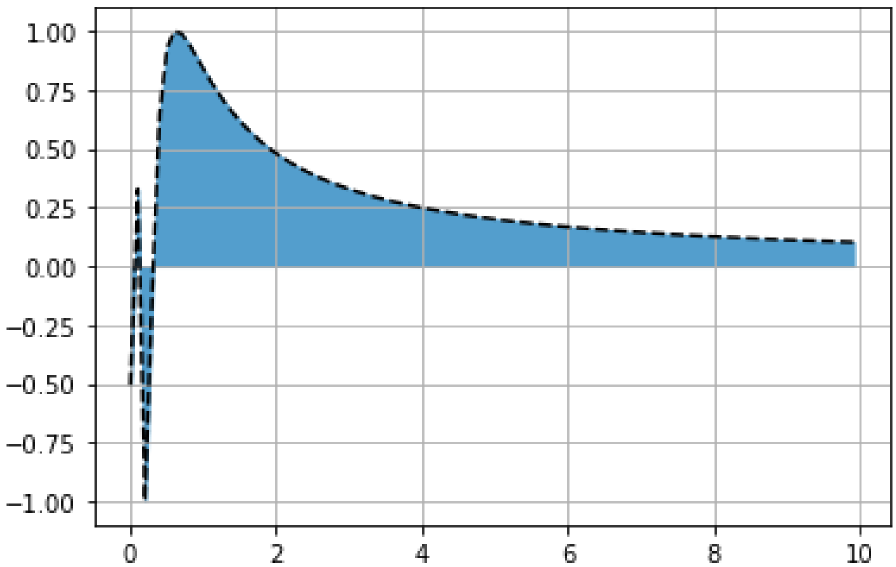
For the 3rd function sin(1/x), we check with interval [0,10]:

As our function is not defined at x = 0 , I will skip the initial value of 0 by taking a very small valeu.

```
def f(x): return np.sin(1/x)

sum = trapezoid(f, 0.1, 10, 10)
print('The sum is: % d ' % sum)
x = np.arange(0.01, 10, 0.1)  # As our function is not defined at x = 0,
y = np.sin(1/x)  # I am skipping the  value of x= 0 , starting from 0.01
# for accuracy measurement.
res, err = quad(f, 0.1, 10)
print("The actual numerical result is {:f} (+-{:g})"
    .format(res, err))
print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

```
The sum is:  2
The actual numerical result is 2.735148 (+-5.78793e-09)
Accuracy:  0.15423268889214625
```

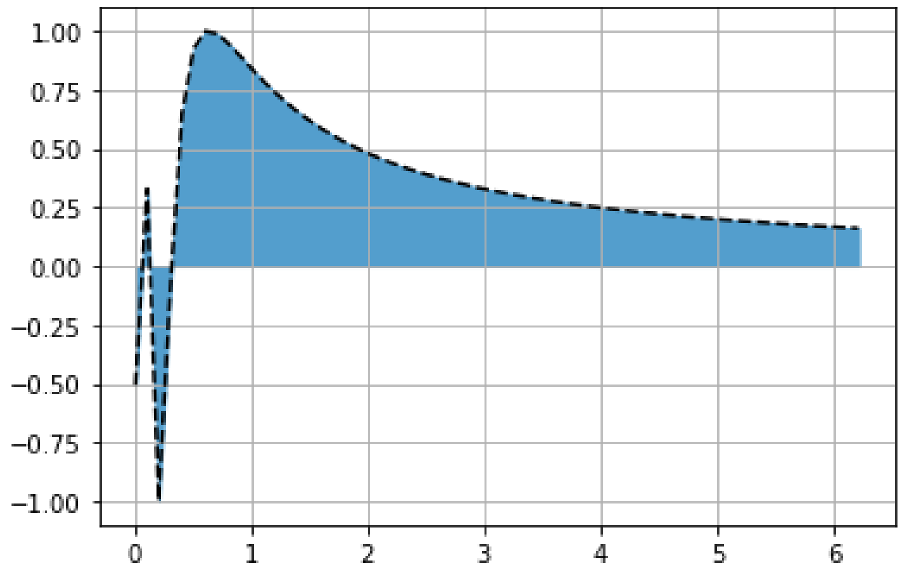


Second: We check with the boundary $[0, 2\pi]$

```
def f(x): return np.sin(1/x)


sum = simpson(f, 0.01, 2*(np.pi), 10)
print('The sum is: % d ' % sum)
x = np.arange(0.01, 2*(np.pi), 0.1)
y = np.sin(1/x)
# for accuracy measurement.
res, err = quad(f, 0.01, 2*np.pi)
print("The actual numerical result is {:f} (+-{:g})"
    .format(res, err))
print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

```
The sum is:  2
The actual numerical result is 2.262686 (+-1.47477e-08)
Accuracy:  0.07548894881414826
```

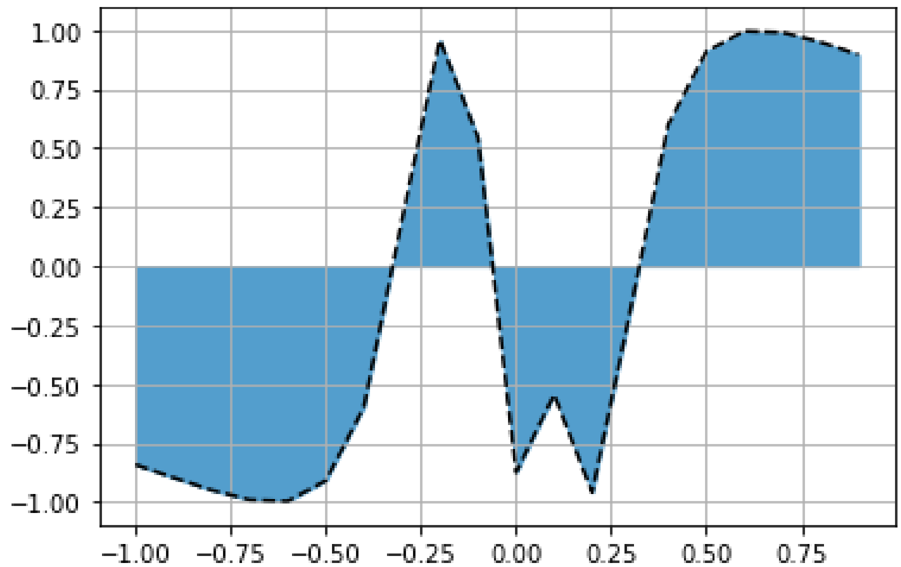


Finaly we check with boudary [-1,1] My error calculating function did not work this time

```
def f(x): return np.sin(1/x)

sum = simpson(f, -1, 1, .1)  # skipping x= 0 input by stepping as 0.1
print('The sum is: % d ' % sum)
x = np.arange(-1, 1, 0.1)
y = np.sin(1/x)
# for accuracy measurement.
#res, err = quad(f, -1, 1)
#print("The actual numerical result is {:f} (+-{:g})"
 #    .format(res, err))
#print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

```
The sum is:  0
```



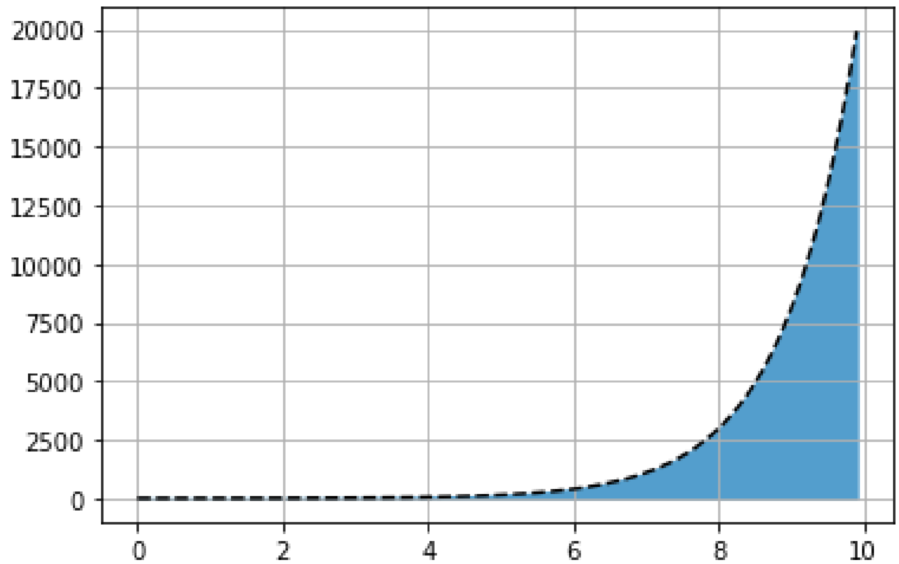
Method 2 (Adaptive Tropozoid)

```
from integrals import adaptive_trapezoid as at
```

For the fucntion e^x, we check with boudary [0,10]

```
def f(x): return np.exp(x)
sum = at(f, 0, 10, 10)
print('The sum is: % d ' % sum)
x = np.arange(0, 10, 0.1)
y = np.exp(x)
# for accuracy measurement.
res, err = quad(f, 0, 10)
print("The actual numerical result is {:f} (+-{:g})"
    .format(res, err))
print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

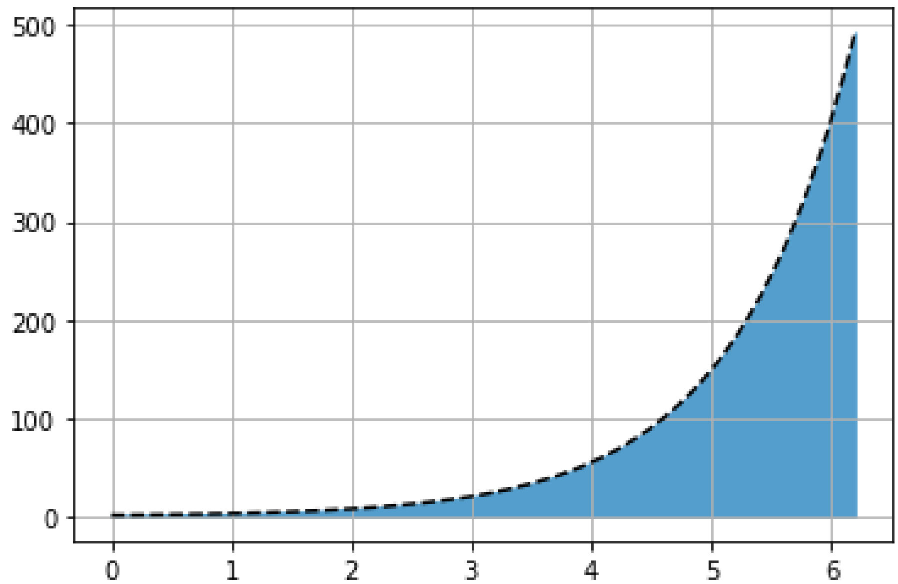
```
The sum is:  22028
The actual numerical result is 22025.465795 (+-6.23939e-10)
Accuracy:  0.00012715334187861084
```



Second: We check with the boundary $[0, 2\pi]$

```
def f(x): return np.exp(x)
sum = at(f, 0, 2*(math.pi), 10)
print('The sum is: % d ' % sum)
x = np.arange(0, 2*(math.pi), 0.1)
y = np.exp(x)
# for accuracy measurement.
res, err = quad(f, 0, 2*(math.pi))
print("The actual numerical result is {:f} (+-{:g})"
    .format(res, err))
print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

```
The sum is:  536
The actual numerical result is 534.491656 (+-5.93405e-12)
Accuracy:  0.003210699374697461
```

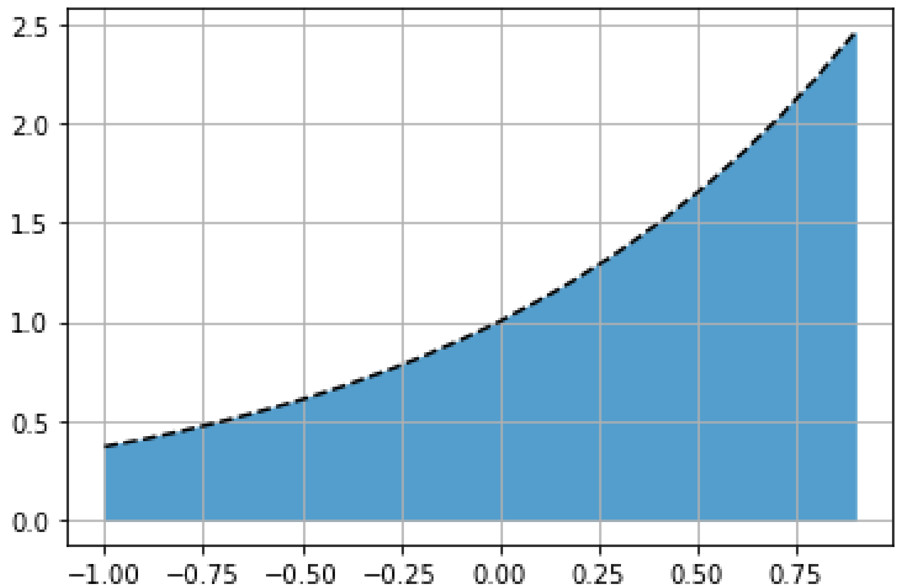


Finaly we check with boudary [-1,1]

```
def f(x): return np.exp(x)


sum = at(f, -1, 1, 10)
print('The sum is: % d ' % sum)
x = np.arange(-1, 1, 0.1)
y = np.exp(x)
# for accuracy measurement.
res, err = quad(f, -1, 1)
print("The actual numerical result is {:f} (+-{:g})"
    .format(res, err))
print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

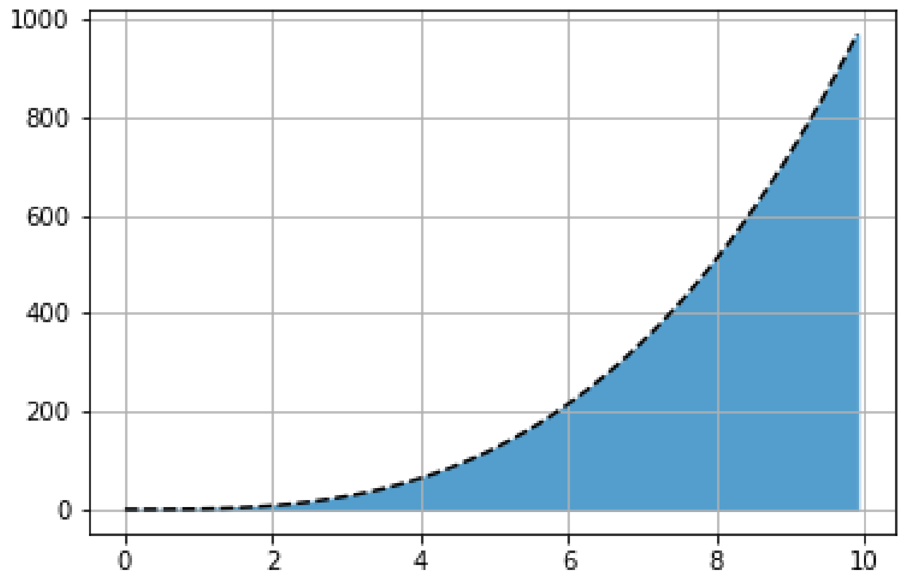
```
The sum is:  2
The actual numerical result is 2.350402 (+-2.60947e-14)
Accuracy:  0.08197670686932626
```



For the 2nd function x^3, we check with interval [0,10]:

```
def f(x): return x ** 3
sum = at(f, 0, 10, 10)
print('The sum is: % d ' % sum)
x = np.arange(0, 10, 0.1)
y = x**3
# for accuracy measurement.
res, err = quad(f, 0, 10)
print("The actual numerical result is {:f} (+-{:g})"
    .format(res, err))
print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

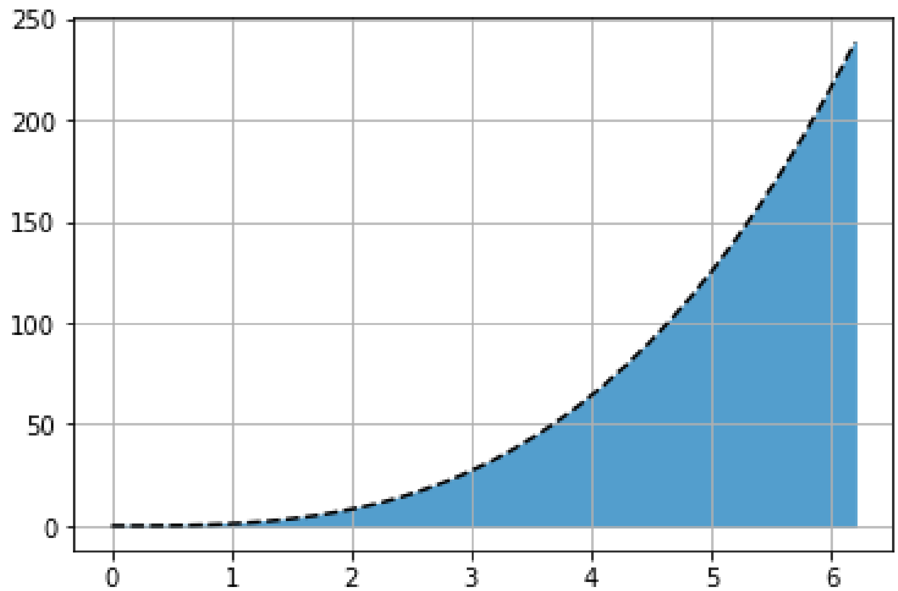
```
The sum is:  2502
The actual numerical result is 2500.000000 (+-2.77556e-11)
Accuracy:  0.000976562499999818
```



Second: We check with the boundary $[0, 2\pi]$

```
def f(x): return x ** 3
sum = at(f, 0, 2*(math.pi), 10)
print('The sum is: % d ' % sum)
x = np.arange(0, 2*(math.pi), 0.1)
y = x**3
# for accuracy measurement.
res, err = quad(f, 0, 2*(math.pi))
print("The actual numerical result is {:f} (+-{:g})"
    .format(res, err))
print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

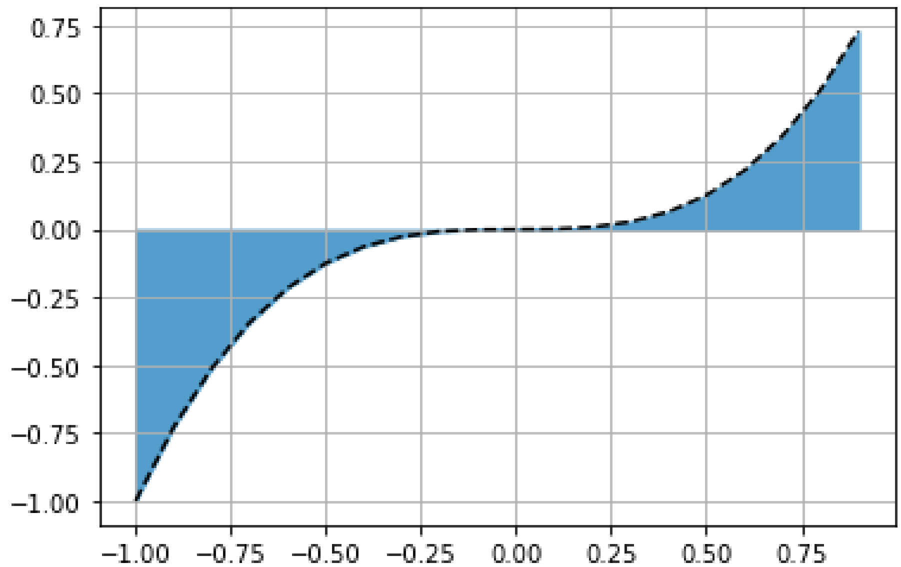
```
The sum is:  395
The actual numerical result is 389.636364 (+-4.32583e-12)
Accuracy:  0.015624999999999915
```



Finaly we check with boudary [-1,1]

```
def f(x): return np.exp(x)
sum = at(f, -1, 1, 10)
print('The sum is: % d ' % sum)
x = np.arange(-1, 1, 0.1)
y = x**3
# for accuracy measurement.
res, err = quad(f, -1, 1)
print("The actual numerical result is {:f} (+-{:g})"
    .format(res, err))
print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

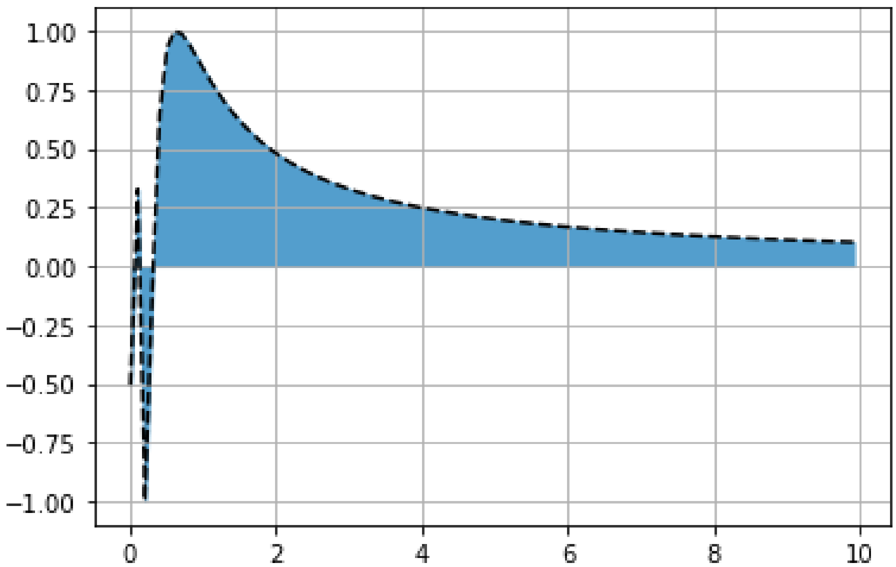
```
The sum is:  2
The actual numerical result is 2.350402 (+-2.60947e-14)
Accuracy:  0.08197670686932626
```



For the 3rd function $sin(1/x)$, first , we check with interval [0,10]:

```
def f(x): return np.sin(1/x)
sum = at(f, 0.01, 10, 10)
print('The sum is: % d ' % sum)
x = np.arange(0.01, 10, 0.1)  # As our function is not defined at x = 0,
y = np.sin(1/x)  # I am skipping the initial value of 0 , starting from 0.01
# for accuracy measurement.
res, err = quad(f, 0.01, 10)
print("The actual numerical result is {:f} (+-{:g})"
    .format(res, err))
print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

```
The sum is:  0
The actual numerical result is 2.726117 (+-3.25786e-08)
Accuracy:  1.0087813722239785
```



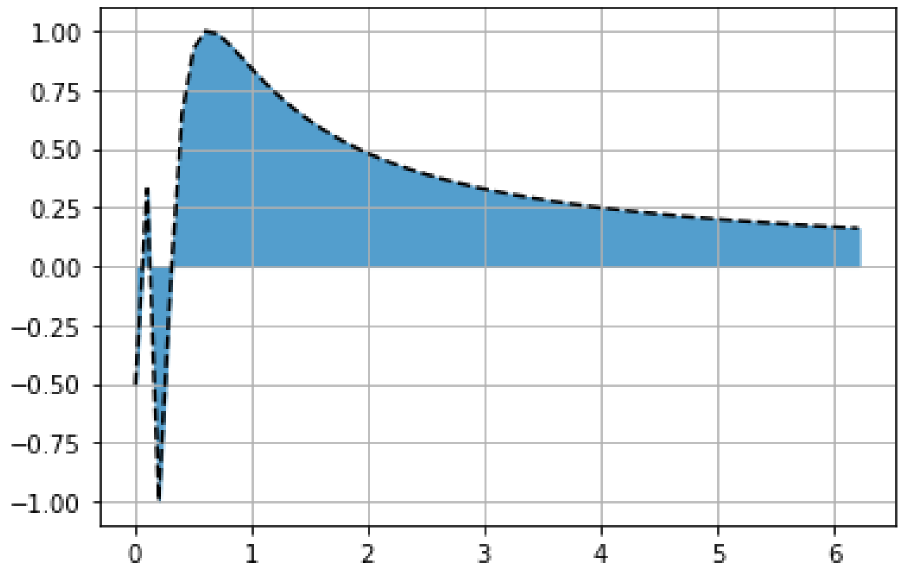
THis is tottaly wrong result , error is high

Second: We check with the boundary $[0, 2\pi]$

```
def f(x): return np.sin(1/x)


sum = at(f, 0.01, 2*(math.pi), 10)
print('The sum is: % d ' % sum)
x = np.arange(0.01, 2*(math.pi), 0.1)  # Our function is not defined at x = 0,
y = np.sin(1/x)  # I am skipping the initial value of 0 , starting from 0.01
# for accuracy measurement.
res, err = quad(f, 0.01, 2*(math.pi))
print("The actual numerical result is {:f} (+-{:g})"
    .format(res, err))
print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

```
The sum is:  0
The actual numerical result is 2.262686 (+-1.47477e-08)
Accuracy:  0.8079516470528277
```



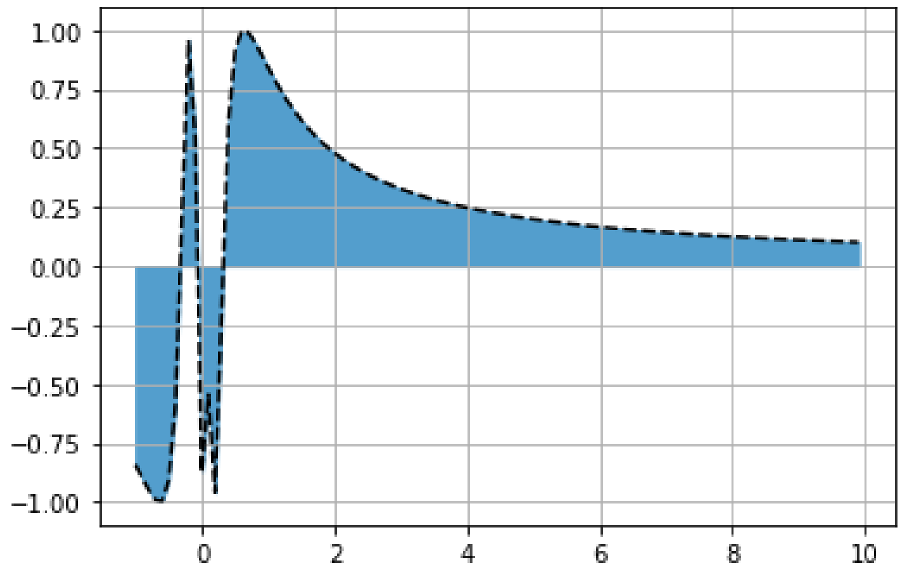
Here also error is high

Finaly we check with boudary [-1,1].

I was not able to skip the value for x=0 , which gives error like ' RuntimeWarning: divide by zero encountered in double_scalars'.I could not fix the error.

```
def f(x): return np.sin(1/x)
sum = at(f, 0.01, 1, 1e-5)
print('The sum is: % d ' % sum)
x = np.arange(-1, 10, 0.1)  # As our function is not defined at x = 0,
y = np.sin(1/x)  # I am skipping the initial value of 0 , starting from 0.01
# for accuracy measurement.
res, err = quad(f, 0.01, 1)
print("The actual numerical result is {:f} (+-{:g})"
    .format(res, err))
print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

```
The sum is:  0
The actual numerical result is 0.503982 (+-8.82858e-09)
Accuracy:  5.236436629932256e-06
```



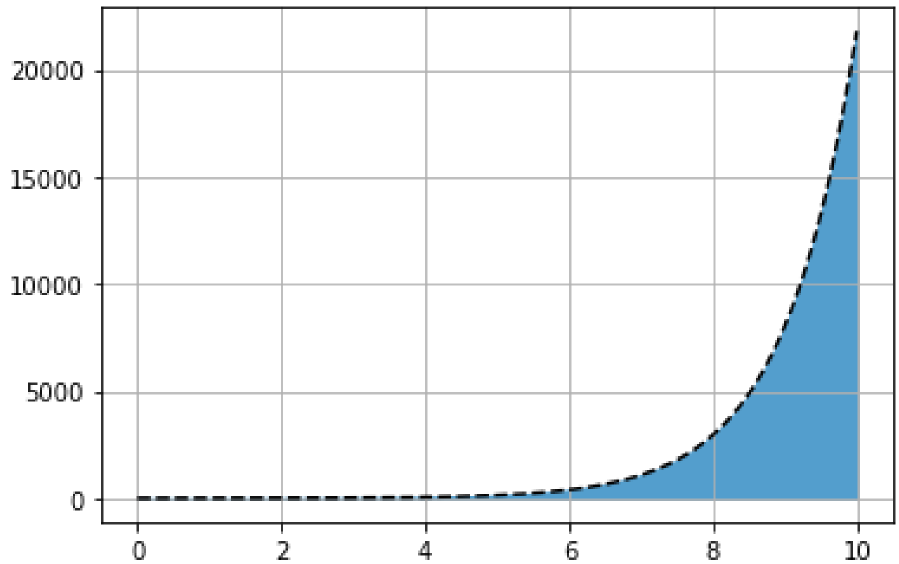
Method3 : Trapezoid

For function : e^x **with boundary [0,10] with subintervals 10.**

```
from integrals import trapezoid as tp
```

```
def f(x): return np.exp(x)
sum = tp(f, 0, 10, 10)
print('The sum is: % d ' % sum)
x = np.arange(0, 10, .01)
y = np.exp(x)
# for accuracy measurement.
res, err = quad(f, 0, 10)
print("The actual numerical result is {:f} (+-{:g})"
    .format(res, err))
print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

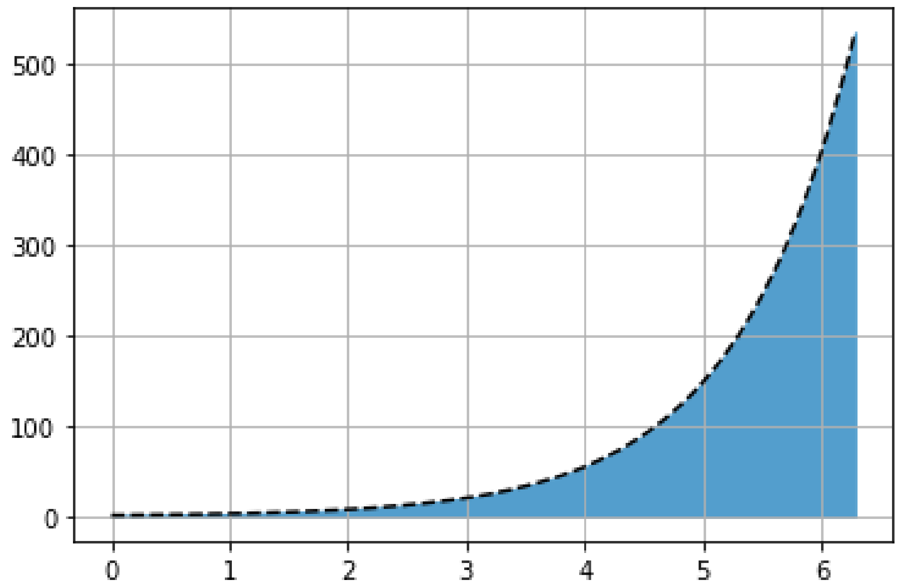
```
The sum is:  23831
The actual numerical result is 22025.465795 (+-6.23939e-10)
Accuracy:  0.08197670686932582
```



Second: We check with the boundary $[0, 2\pi]$

```
def f(x): return np.exp(x)
sum = tp(f, 0, 2*(math.pi), 10)
print('The sum is: % d ' % sum)
x = np.arange(0, 2*(math.pi), .01)
y = np.exp(x)
# for accuracy measurement.
res, err = quad(f, 0, 2*(math.pi))
print("The actual numerical result is {:f} (+-{:g})"
    .format(res, err))
print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

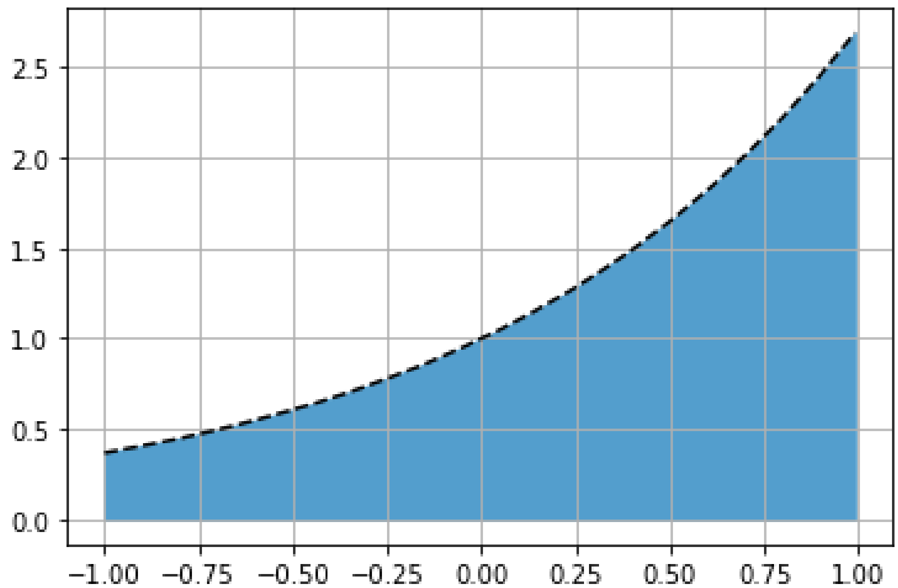
```
The sum is:  551
The actual numerical result is 534.491656 (+-5.93405e-12)
Accuracy:  0.03268423149301747
```



Finaly we check with boudary [-1,1]

```
def f(x): return np.exp(x)
sum = tp(f, -1, 1, 10)
print('The sum is: % d ' % sum)
x = np.arange(-1, 1, .01)
y = np.exp(x)
# for accuracy measurement.
res, err = quad(f, -1,1)
print("The actual numerical result is {:f} (+-{:g})"
    .format(res, err))
print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

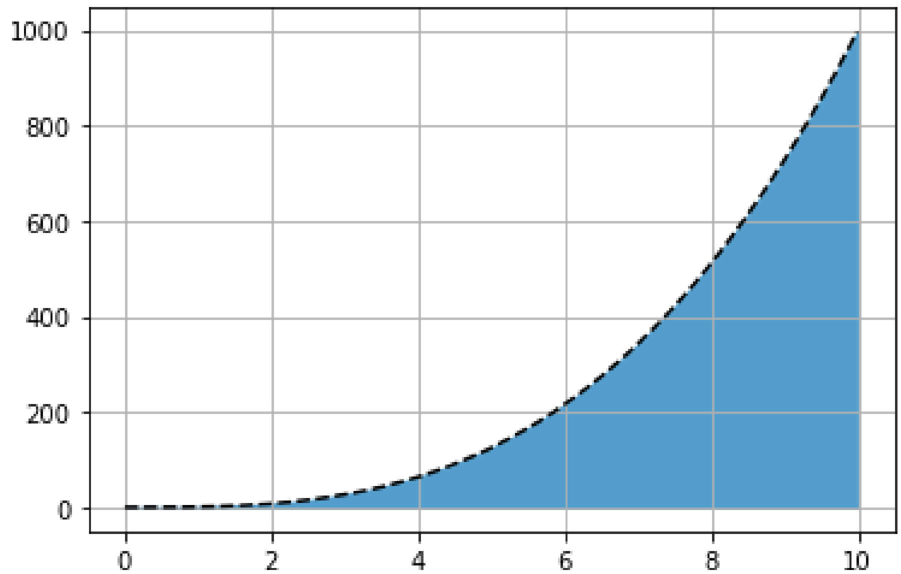
```
The sum is:  2
The actual numerical result is 2.350402 (+-2.60947e-14)
Accuracy:  0.0033311132253990156
```



For the 2nd function x^3, we check with interval [0,10]:

```
def f(x): return x**3
sum = tp(f, 0, 10, 10)
print('The sum is: % d ' % sum)
x = np.arange(0, 10, .01)
y = x**3
# for accuracy measurement.
res, err = quad(f, 0, 10)
print("The actual numerical result is {:f} (+-{:g})"
    .format(res, err))
print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

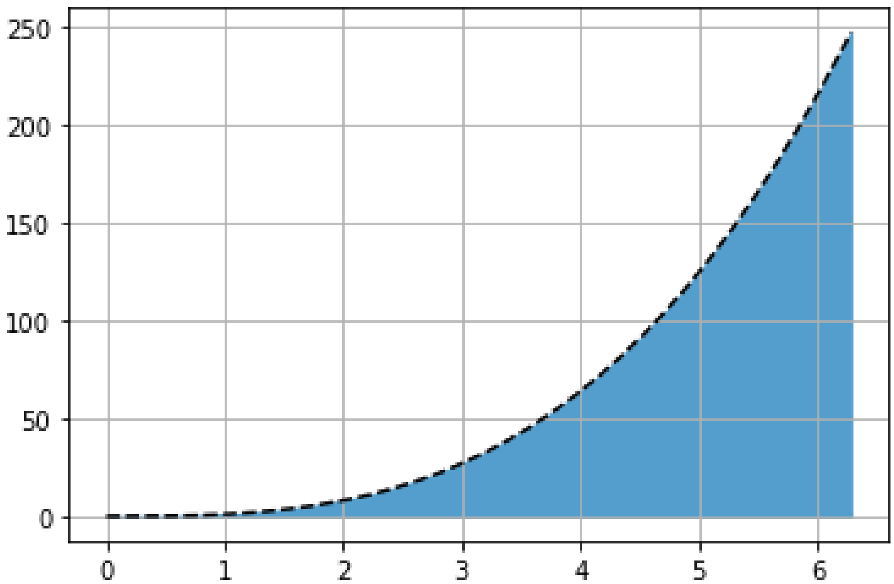
```
The sum is:  2525
The actual numerical result is 2500.000000 (+-2.77556e-11)
Accuracy:  0.009999999999999816
```



Second: We check with the boundary $[0, 2\pi]$

```
def f(x): return x**3
sum = tp(f, 0, 2*(math.pi), 10)
print('The sum is: % d ' % sum)
x = np.arange(0, 2*(math.pi), .01)
y = x**3
# for accuracy measurement.
res, err = quad(f, 0, 2*(math.pi))
print("The actual numerical result is {:f} (+-{:g})"
    .format(res, err))
print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

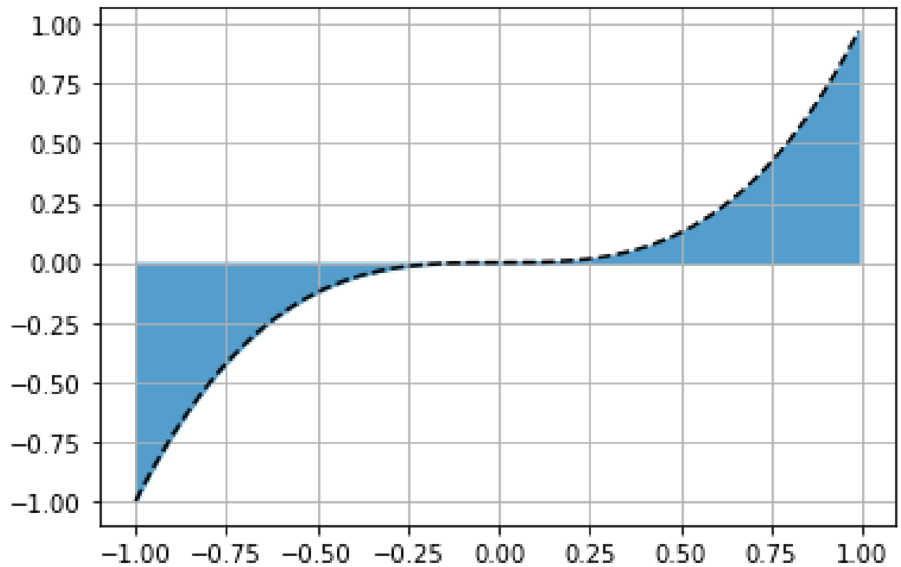
```
The sum is:  393
The actual numerical result is 389.636364 (+-4.32583e-12)
Accuracy:  0.009999999999999853
```



Finaly we check with boudary [-1,1] the accuracy method did not work for this

```
def f(x): return x**3
sum = tp(f, -1, 1, 10)
print('The sum is: % d ' % sum)
x = np.arange(-1, 1, .01)
y = x**3
# for accuracy measurement.
#res, err = quad(f, -1, 1)
#print("The actual numerical result is {:f} (+-{:g})"
#    .format(res, err))
#print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

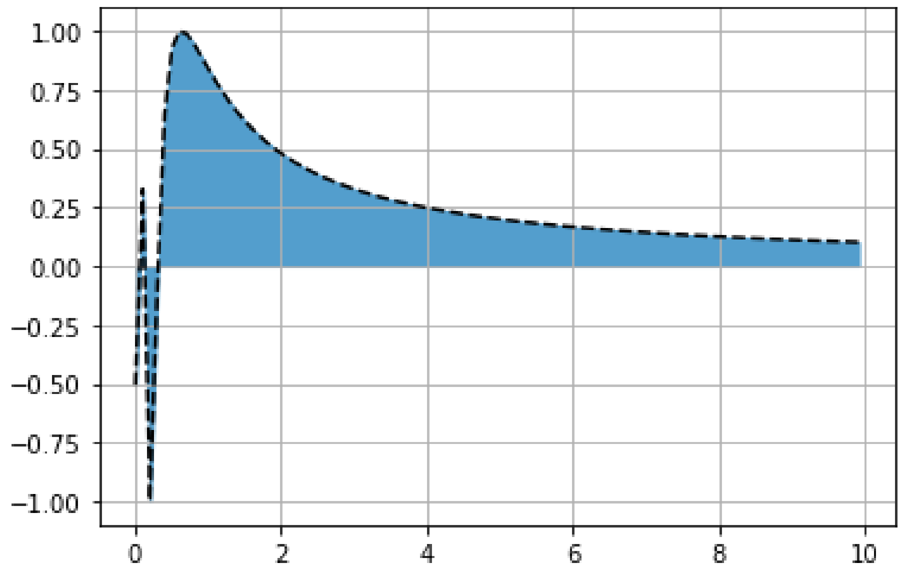
```
The sum is:  0
```



For the 3rd function $sin(1/x)$, first , we check with interval [0,10]:

```
def f(x): return np.sin(1/x)
sum = tp(f, 0.01, 10, 10)
print('The sum is: % d ' % sum)
x = np.arange(0.01, 10, 0.1)  # As our function is not defined at x = 0,
y = np.sin(1/x)  # I am skipping the initial value of 0 , starting from 0.01
# for accuracy measurement.
res, err = quad(f, 0.01,10)
print("The actual numerical result is {:f} (+-{:g})"
    .format(res, err))
print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

```
The sum is:  2
The actual numerical result is 2.726117 (+-3.25786e-08)
Accuracy:  0.1107537010441309
```

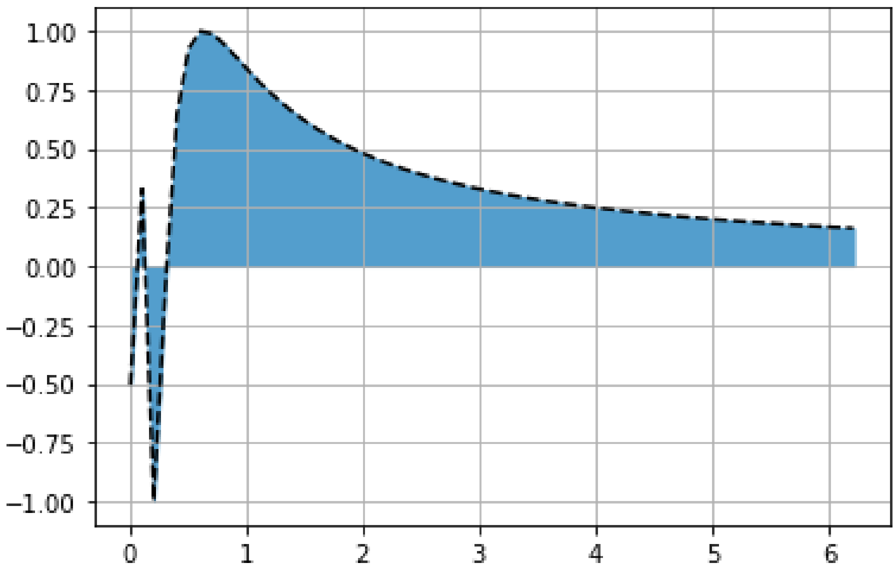


Error is high here.

Second: We check with the boundary $[0, 2\pi]$

```
def f(x): return np.sin(1/x)
sum = tp(f, 0.01, 2*(math.pi), 10)
print('The sum is: % d ' % sum)
x = np.arange(0.01, 2*(math.pi), 0.1)  # Our function is not defined at x = 0,
y = np.sin(1/x)  # I am skipping the initial value of 0 , starting from 0.01
# for accuracy measurement.
res, err = quad(f, 0.01, 2*(math.pi))
print("The actual numerical result is {:f} (+-{:g})"
    .format(res, err))
print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

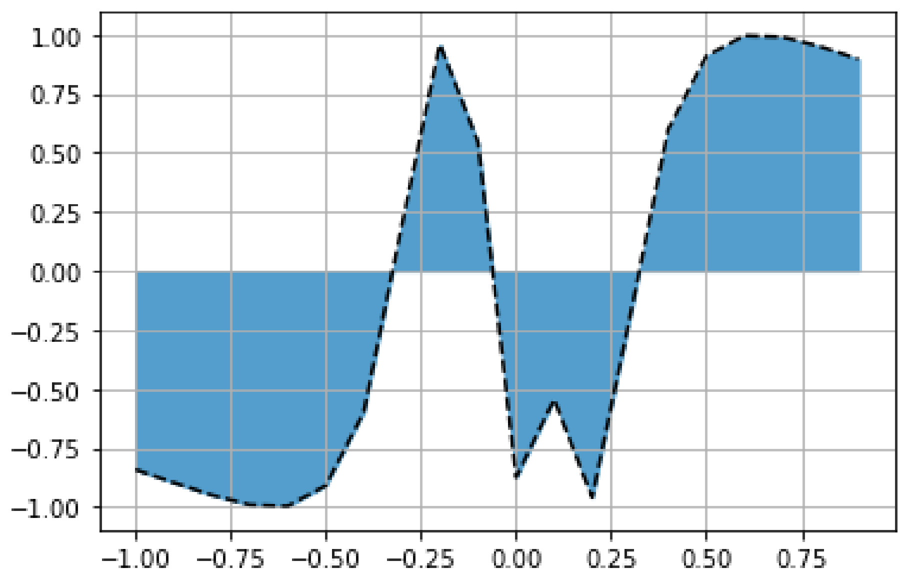
```
The sum is:  2
The actual numerical result is 2.262686 (+-1.47477e-08)
Accuracy:  0.001671780845322321
```



Finaly we check with boudary [-1,1]

```
def f(x): return np.sin(1/x)
sum = tp(f, -1, 1, 0.1)
print('The sum is: % d ' % sum)
x = np.arange(-1, 1, 0.1)  # As our function is not defined at x = 0,
y = np.sin(1/x)  # I am skipping the  value of x= 0 , by giving steps 0.01
# for accuracy measurement.
#res, err = quad(f, -1, 1)
#print("The actual numerical result is {:f} (+-{:g})"
#     .format(res, err))
#print("Accuracy: ",np.abs(sum-res)/res)
plt.plot(x, y, 'k--')
plt.fill_between(x, y, color='#539ecd')
plt.grid()
plt.show()
```

```
The sum is:  0
```



Looks like this method gives very good estimate for some function and very off for other functions.