In [1]:
```
import numpy as np
from scipy import integrate
import matplotlib.pyplot as plt
```

In [4]:
```
def f(x):
    return np.exp(x)
x=np.arange(0,1,0.1)
y=f(x)
trap_res=integrate.trapezoid(y,dx=0.1)
simp_res=integrate.simpson(y,dx=0.1)
romb_res=integrate.romberg(f,0,1,show=True)
print("Using Trapezoidal Rule=",trap_res)
print("Using Simpson's Rule=",simp_res)
print("Using Romberg Integration=",romb_res)
```

```
Romberg integration of <function vectorize1.<locals>.vfunc at 0x7f7
34431c400> from [0, 1]

 Steps  StepSize   Results
     1  1.000000  1.859141
     2  0.500000  1.753931  1.718861
     4  0.250000  1.727222  1.718319  1.718283
     8  0.125000  1.720519  1.718284  1.718282  1.718282
    16  0.062500  1.718841  1.718282  1.718282  1.718282  1.718282

The final result is 1.7182818284590782 after 17 function evaluation
s.
Using Trapezoidal Rule= 1.4608192444085148
Using Simpson's Rule= 1.4597451505369412
Using Romberg Integration= 1.7182818284590782
```

In [54]:
```
def f(x):
    return np.log(x)*x
n=4
xi=1
xf=2
dx=(xf-xi)/n
x=np.arange(xi,xf+dx,dx)
print(x)
y=f(x)
trap_res=integrate.trapezoid(y,dx=dx)
simp_res=integrate.simpson(y,dx=dx)
print("f(x)=x ln(x)")
print("Using Trapezoidal Rule=",trap_res)
print("Using Simpson's Rule=",simp_res)
```

```
[1.   1.25 1.5  1.75 2.  ]
f(x)=x ln(x)
Using Trapezoidal Rule= 0.6399004776879859
Using Simpson's Rule= 0.6363098297969493
```

In [70]:
```
def f(x):
    return 2/(x+4)
n=6
xi=0
xf=2
dx=(xf-xi)/n
x=np.arange(xi,xf+dx,dx)[:-1] #The code was giving an error due to ma
print(x)
y=f(x)
trap_res=integrate.trapezoid(y,dx=dx)
simp_res=integrate.simpson(y,dx=dx)
print("f(x)=x ln(x)")
print("Using Trapezoidal Rule=",trap_res)
print("Using Simpson's Rule=",simp_res)
```

```
[0.         0.33333333 0.66666667 1.         1.33333333 1.66666667
 2.        ]
f(x)=x ln(x)
Using Trapezoidal Rule= 0.8115725777490483
Using Simpson's Rule= 0.8109327491680433
```

In [71]:
```
def f(x):
    return np.tan(x)
n=8
xi=0
xf=np.pi*3/8
dx=(xf-xi)/n
x=np.arange(xi,xf+dx,dx)
print(x)
y=f(x)
trap_res=integrate.trapezoid(y,dx=dx)
simp_res=integrate.simpson(y,dx=dx)
print("f(x)=x ln(x)")
print("Using Trapezoidal Rule=",trap_res)
print("Using Simpson's Rule=",simp_res)
```

```
[0.         0.14726216 0.29452431 0.44178647 0.58904862 0.73631078
 0.88357293 1.03083509 1.17809725]
f(x)=x ln(x)
Using Trapezoidal Rule= 0.9709263066791303
Using Simpson's Rule= 0.9610553984955355
```

In [17]:
```
#Romberg Integration
def r33(f,a,b):
    P=b-a #Period of Integration
    r11=(f(a)+f(b))*P/2
    r21=(f(a)+f(b)+2*f((a+b)/2))*P/4
    r31=(f(a)+f(b)+(f(a+(b-a)/4)+f(a+2*(b-a)/4)+f(a+3*(b-a)/4))*2)*P/
    r32=r31+(r31-r21)/3
    r22=r21+(r21-r11)/3
    r33=r32+(r32-r22)/15
    return r33
```

In [27]:
```
def f(x):
    return x**2 * np.log(x)
print("Using My code=",r33(f,1,1.5))
print("Using scipy.integrate:")
integrate.romberg(f,1,1.5,show=True)
```

```
Using My code= 0.19225933731444386
Using scipy.integrate:
Romberg integration of <function vectorize1.<locals>.vfunc at 0x7f7
343776ca0> from [1, 1.5]

 Steps  StepSize   Results
     1  0.500000  0.228074
     2  0.250000  0.201203  0.192245
     4  0.125000  0.194494  0.192258  0.192259
     8  0.062500  0.192818  0.192259  0.192259  0.192259
    16  0.031250  0.192399  0.192259  0.192259  0.192259  0.192259

The final result is 0.19225935773277802 after 17 function evaluatio
ns.
```

Out[27]:
```
0.19225935773277802
```

In [29]:
```
def f(x):
    return x**2 * np.exp(-x)
print("Using My code=",r33(f,0,1))
print("Using scipy.integrate:")
integrate.romberg(f,0,1,show=True)
```

```
Using My code= 0.1606105286979897
Using scipy.integrate:
Romberg integration of <function vectorize1.<locals>.vfunc at 0x7f7
3437758a0> from [0, 1]

 Steps  StepSize   Results
     1  1.000000  0.183940
     2  0.500000  0.167786  0.162402
     4  0.250000  0.162488  0.160722  0.160611
     8  0.125000  0.161080  0.160610  0.160603  0.160603
    16  0.062500  0.160722  0.160603  0.160603  0.160603  0.160603

The final result is 0.16060279414376905 after 17 function evaluatio
ns.
```

Out[29]:
```
0.16060279414376905
```

In [30]:
```
def f(x):
    return np.cos(x)**2
print("Using My code=",r33(f,0,np.pi/4))
print("Using scipy.integrate:")
integrate.romberg(f,0,np.pi/4,show=True)
```

```
Using My code= 0.6426969730669724
Using scipy.integrate:
Romberg integration of <function vectorize1.<locals>.vfunc at 0x7f7
343777ce0> from [0, 0.7853981633974483]

 Steps  StepSize   Results
     1  0.785398  0.589049
     2  0.392699  0.629714  0.643269
     4  0.196350  0.639478  0.642733  0.642697
     8  0.098175  0.641895  0.642701  0.642699  0.642699
    16  0.049087  0.642498  0.642699  0.642699  0.642699  0.642699

The final result is 0.6426990816982282 after 17 function evaluation
s.
```

Out[30]:
```
0.6426990816982282
```

In [ ]:
```
```