MC Estimations 1: Area calculation

1-

$$F(x) = 3x^3 - 5x^2 + 1$$

$$xmin = -1$$
$$xmax = 2$$

Double-click (or enter) to edit

1000 Samples

```
import numpy as np
from scipy.integrate import quad
from pylab import *


l=[]

def func1(x):

    f=1-(5*x**2)+(3*x**2)
    v=0
    if f>=0:
      l.append(x)
      return f
    else:
```

```
    else:
      return v

def function(x):
    f=1-(5*x**2)+(3*x**2)
    v=0
    if f>=0:

       return f
    else:
       return (v*x)

def mc_integrate(func, a, b, n = 1000):
    # Monte Carlo integration between x1 and x2 of given function from a to b

    vals = np.random.uniform(a, b, n)
    y = [abs(func(val)) for val in vals]

    y_mean = np.sum(y)/n
    integ = (b-a) * y_mean
    return integ

print(f"Monte Carlo solution: {mc_integrate(func1, -1, 2, 1000): .4f}")
res,_=quad(function, -1, 2)
print(f"Correct solution: {res: .4f}")
```

```
Monte Carlo solution:  0.9945
Correct solution:  0.9428
```

```
mx=math.floor(max(l))
mn=math.floor(min(l))
```

```
print(f'Calculation error: {np.sqrt((mx-mn)/12)/np.sqrt(1000)}')
```

```
Calculation error: 0.009128709291752768
```

10000 Samples

```
import numpy as np
from scipy.integrate import quad
from pylab import *


l=[]

def func1(x):

    f=1-(5*x**2)+(3*x**2)
    v=0
    if f>=0:
      l.append(x)
      return f
    else:
      return v

def function(x):
    f=1-(5*x**2)+(3*x**2)
    v=0
    if f>=0:

       return f
    else:
```

```
        return (v*x)

def mc_integrate(func, a, b, n = 10000):
    # Monte Carlo integration between x1 and x2 of given function from a to b

    vals = np.random.uniform(a, b, n)
    y = [abs(func(val)) for val in vals]

    y_mean = np.sum(y)/n
    integ = (b-a) * y_mean
    return integ

print(f"Monte Carlo solution: {mc_integrate(func1, -1, 2, 10000): .4f}")
res,_=quad(function, -1, 2)
print(f"Correct solution: {res: .4f}")
```

```
Monte Carlo solution:  0.9492
Correct solution:  0.9428
```

```
mx=math.floor(max(l))
mn=math.floor(min(l))
```

```
print(f'Calculation error: {np.sqrt((mx-mn)/12)/np.sqrt(10000)}')
```

```
Calculation error: 0.0028867513459481286
```

2-

$$F(x) = e^{-x}$$

$$xmin = -0.5$$
$$xmax = 1$$

Double-click (or enter) to edit

1000 Samples

```
import numpy as np
from scipy.integrate import quad
from pylab import *


l=[]

def func1(x):

    f= e**(-x)
    v=0
    if f>=0:
      l.append(x)
      return f
    else:
      return v

def function(x):
    f= e**(-x)
    v=0
    if f>=0:

       return f
    else:
       return (v*x)
```

```

def mc_integrate(func, a, b, n = 1000):
    # Monte Carlo integration between x1 and x2 of given function from a to b

    vals = np.random.uniform(a, b, n)
    y = [abs(func(val)) for val in vals]

    y_mean = np.sum(y)/n
    integ = (b-a) * y_mean
    return integ

print(f"Monte Carlo solution: {mc_integrate(func1, -0.5, 1, 1000): .4f}")
res,_=quad(function, -0.5,1 )
print(f"Correct solution: {res: .4f}")
```

```
Monte Carlo solution:  1.3110
Correct solution:  1.2808
```

```
mx=(max(l))
mn=(min(l))
```

```
print(f'Calculation error: {np.sqrt((mx-mn)/12)/np.sqrt(1000)}')
```

```
Calculation error: 0.011174739552668402
```

```

```

10000 Samples

```
import numpy as np
```

```
from scipy.integrate import quad
from pylab import *


l=[]

def func1(x):

    f= e**(-x)
    v=0
    if f>=0:
      l.append(x)
      return f
    else:
      return v

def function(x):
    f= e**(-x)
    v=0
    if f>=0:

       return f
    else:
       return (v*x)

def mc_integrate(func, a, b, n = 10000):
    # Monte Carlo integration between x1 and x2 of given function from a to b

    vals = np.random.uniform(a, b, n)
    y = [abs(func(val)) for val in vals]

```

```
    y_mean = np.sum(y)/n
    integ = (b-a) * y_mean
    return integ

print(f"Monte Carlo solution: {mc_integrate(func1, -0.5, 1, 10000): .4f}")
res,_=quad(function, -0.5,1 )
print(f"Correct solution: {res: .4f}")
```

```
Monte Carlo solution:  1.2807
Correct solution:  1.2808
```

```
mx=(max(l))
mn=(min(l))
```

```
print(f'Calculation error: {np.sqrt((mx-mn)/12)/np.sqrt(10000)}')
```

```
Calculation error: 0.003535084187420184
```

3-

$$F(x) = sin(3x)$$

$$xmin = -\pi$$
$$xmax = 0$$

Double-click (or enter) to edit

1000 Samples

```
import numpy as np
from scipy.integrate import quad
```

```
from pylab import *


l=[]

def func1(x):

    f=math.sin(3*x)
    v=0
    if f>=0:
      l.append(x)
      return f
    else:
      return v

def function(x):
    f=math.sin(3*x)
    v=0
    if f>=0:

       return f
    else:
       return (v*x)

def mc_integrate(func, a, b, n = 1000):
    # Monte Carlo integration between x1 and x2 of given function from a to b

    vals = np.random.uniform(a, b, n)
    y = [abs(func(val)) for val in vals]

    y_mean = np.sum(y)/n
```

```
    integ = (b-a) * y_mean
    return integ

print(f"Monte Carlo solution: {mc_integrate(func1, -math.pi, 0, 1000): .4f}")
res,_=quad(function, -math.pi, 0)
print(f"Correct solution: {res: .4f}")
```

```
Monte Carlo solution:  0.6808
Correct solution:  0.6667
```

```
mx=(max(l))
mn=(min(l))
```

```
print(f'Calculation error: {np.sqrt((mx-mn)/12)/np.sqrt(1000)}')
```

```
Calculation error: 0.009325006456078368
```

Double-click (or enter) to edit

10000 Samples

```
import numpy as np
from scipy.integrate import quad
from pylab import *


l=[]

def func1(x):

```

```
    f=math.sin(3*x)
    v=0
    if f>=0:
      l.append(x)
      return f
    else:
      return v

def function(x):
    f=math.sin(3*x)
    v=0
    if f>=0:

       return f
    else:
       return (v*x)

def mc_integrate(func, a, b, n = 10000):
    # Monte Carlo integration between x1 and x2 of given function from a to b

    vals = np.random.uniform(a, b, n)
    y = [abs(func(val)) for val in vals]

    y_mean = np.sum(y)/n
    integ = (b-a) * y_mean
    return integ

print(f"Monte Carlo solution: {mc_integrate(func1, -math.pi, 0, 10000): .4f}")
res,_=quad(function, -math.pi, 0)
print(f"Correct solution: {res: .4f}")
```

```
Monte Carlo solution:  0.6735
```

```
Correct solution:  0.6667
```

```
mx=(max(l))
mn=(min(l))
```

```
print(f'Calculation error: {np.sqrt((mx-mn)/12)/np.sqrt(10000)}')
```

```
Calculation error: 0.002951649105715342
```

4-

$$F(x) = cos(4x)$$

$$xmin = -\pi$$
$$xmax = 0$$

Double-click (or enter) to edit

1000 Samples

```
import numpy as np
from scipy.integrate import quad
from pylab import *


l=[]

def func1(x):

    f=math.cos(4*x)
```

```
    v=0
    if f>=0:
      l.append(x)
      return f
    else:
      return v

def function(x):
    f=math.cos(4*x)
    v=0
    if f>=0:

       return f
    else:
       return (v*x)

def mc_integrate(func, a, b, n = 1000):
    # Monte Carlo integration between x1 and x2 of given function from a to b

    vals = np.random.uniform(a, b, n)
    y = [abs(func(val)) for val in vals]

    y_mean = np.sum(y)/n
    integ = (b-a) * y_mean
    return integ

print(f"Monte Carlo solution: {mc_integrate(func1, -math.pi, 0, 1000): .4f}")
res,_=quad(function, -math.pi, 0)
print(f"Correct solution: {res: .4f}")
```

```
Monte Carlo solution:  1.0190
Correct solution:  1.0000
```

```
mx=(max(l))
mn=(min(l))
```

```
print(f'Calculation error: {np.sqrt((mx-mn)/12)/np.sqrt(1000)}')
```

```
Calculation error: 0.016173080102728506
```

```
1
```

10000 Samples

```
import numpy as np
from scipy.integrate import quad
from pylab import *


l=[]

def func1(x):

    f=math.cos(4*x)
    v=0
    if f>=0:
      l.append(x)
      return f
    else:
      return v

def function(x):
```

```
    f=math.cos(4*x)
    v=0
    if f>=0:

       return f
    else:
       return (v*x)

def mc_integrate(func, a, b, n = 10000):
    # Monte Carlo integration between x1 and x2 of given function from a to b

    vals = np.random.uniform(a, b, n)
    y = [abs(func(val)) for val in vals]

    y_mean = np.sum(y)/n
    integ = (b-a) * y_mean
    return integ

print(f"Monte Carlo solution: {mc_integrate(func1, -math.pi, 0, 10000): .4f}")
res,_=quad(function, -math.pi, 0)
print(f"Correct solution: {res: .4f}")
```

```
Monte Carlo solution:  0.9959
Correct solution:  1.0000
```

```
mx=(max(l))
mn=(min(l))
```

```
print(f'Calculation error: {np.sqrt((mx-mn)/12)/np.sqrt(10000)}')
```

```
Calculation error: 0.005116468044522063
```

5-

$$F(x) = sin(e^{x/2})$$

$$x = 0$$
$$x = \pi$$

1000 Samples

```
1
```

```
import numpy as np
from scipy.integrate import quad
from pylab import *


l=[]

def func1(x):

    f=math.sin(e**(x/2))
    v=0
    if f>=0:
      l.append(x)
      return f
    else:
      return v

def function(x):
```

```
    f=math.sin(e**(x/2))
    v=0
    if f>=0:

       return f
    else:
       return (v*x)

def mc_integrate(func, a, b, n = 1000):
    # Monte Carlo integration between x1 and x2 of given function from a to b

    vals = np.random.uniform(a, b, n)
    y = [abs(func(val)) for val in vals]

    y_mean = np.sum(y)/n
    integ = (b-a) * y_mean
    return integ

print(f"Monte Carlo solution: {mc_integrate(func1, 0,math.pi, 1000): .4f}")
res,_=quad(function, 0,math.pi)
print(f"Correct solution: {res: .4f}")
```

```
Monte Carlo solution:  1.8693
Correct solution:  1.8117
```

```
mx=(max(l))
mn=(min(l))
```

```
print(f'Calculation error: {np.sqrt((mx-mn)/12)/np.sqrt(1000)}')
```

```
Calculation error: 0.013797480057624496
```

```
1
```

Double-click (or enter) to edit

```
1
```

10000 Samples

```
import numpy as np
from scipy.integrate import quad
from pylab import *


l=[]

def func1(x):

    f=math.sin(e**(x/2))
    v=0
    if f>=0:
      l.append(x)
      return f
    else:
      return v

def function(x):
    f=math.sin(e**(x/2))
    v=0
    if f>=0:
```

```
22
23        return f
24     else:
25        return (v*x)
26
27 def mc_integrate(func, a, b, n = 10000):
28     # Monte Carlo integration between x1 and x2 of given function from a to b
29
30     vals = np.random.uniform(a, b, n)
31     y = [abs(func(val)) for val in vals]
32
33     y_mean = np.sum(y)/n
34     integ = (b-a) * y_mean
35     return integ
36
37 print(f"Monte Carlo solution: {mc_integrate(func1, 0,math.pi, 10000): .4f}")
38 res,_=quad(function, 0,math.pi)
39 print(f"Correct solution: {res: .4f}")
```

```
Monte Carlo solution:  1.8117
Correct solution:  1.8117
```

```
1 mx=(max(l))
2 mn=(min(l))
```

```
1 print(f'Calculation error: {np.sqrt((mx-mn)/12)/100}')
```

```
Calculation error: 0.004366816008795486
```

```
1
```

Double-click (or enter) to edit