Syed Ali

COSC 750

Assignment due 2/16/2020

① ⓑ (i) $I_1 = \int_0^1 e^{-x}\,dx$

$$\frac{e^{-x}}{-1}\Big|_0^1 = -e^{-x}\Big|_0^1 = -[e^{-1}-1] = 1 - \frac{1}{e} \approx 0.6321$$

(ii) $I_2 = \int_0^5 (1-x^2)^2\,dx$

$$\Rightarrow \int_0^5 (1+x^4-2x^2)\,dx$$

$$\Rightarrow x + \frac{x^5}{5} - \frac{2x^3}{3}\Big|_0^5$$

$$\Rightarrow 5 + \frac{5^5}{5} - \frac{2(5^3)}{3}$$

$$\Rightarrow 546.67$$

ⓐ Monte-Carlo integration is given by:-

(i) $I_1 = \int_0^1 e^{-x}\,dx$

$\mu = \int g(x)\,dx$ Where $g(x) = e^{-x}$

$f(x) = 1$ i.e. $x \sim unif(0,1)$

We can calculate the above integral using monte-carlo integration as follows

$I_2 = C\,E_{p(x)}\,h(x)$

$$\approx \frac{b-a}{N}\sum_{i=1}^{N}\left(\exp(-x_i)\right) \Rightarrow \boxed{\frac{1}{N}\sum_{i=1}^{N}\exp(-x_i)}$$

See Python Code attached

(ii) Similarly for $I_2 = \int_0^5 (1-x^2)^2 dx$

again $h(x) = (1-x^2)^2$

we identify $g(x) = 1$

from this we can determine pdf $P(x) \in (a,b) = (0,3)$

$$P(x) = \frac{g(x)}{\int_0^5 g(x) dx} = \frac{1}{5} = \frac{1}{5}$$

We can calculate the Monte-Carlo approximation is as follows:-

$I_2 = C \, E_{p(x)} h(x)$

$$\boxed{I_2 = \frac{1}{N} \sum_{i=1}^{N} (1-x_i^2)^2 \text{ Where, } x_i \text{ is sampled from unif}(0,5)}$$

See Python Code for Calculation

ⓒ Area obtained using Monte-Carlo integration => 0.638 for exp(-x)

Area obtained using regular integration => 0.632 for exp(-x)

Area obtained using // // => 556.38 for $(1-x^2)^2$

Area obtained using // regular // => 546.67 for $(1-x^2)^2$

//

There is discrepancy in the results since Monte-Carlo utilizes integration using random numbers, while regular integration performs analysis at a regular grid.

Generally speaking, monte-carlo technique is very useful evaluating higher dimensional integrals.

Scanned with CamScanner

② ⓐ $\int_{-\infty}^{+\infty} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right) dx$

$\int_{-\infty}^{+\infty} \exp\left(-\frac{1}{2}(x)\right)^2 dx$ When $\mu=0$ and $\sigma=1$

$\mu = \int g(x)\,dx$ Where $g(x) = \exp\left(-\frac{1}{2}x^2\right)$

$f(x) = 1$ ie $x \sim$ uniform $(-\infty, \infty)$

We can calculate the above integral using Monte-Carlo integration as follows

$$I_2 = C E_{p(x)} h(x)$$

$$\approx \boxed{\frac{b-a}{N} \sum_{i=1}^{N} \left(\exp\left(\frac{-x_i^2}{2}\right)\right) \Longrightarrow \frac{1}{N} \sum_{i=1}^{N} \exp\left(\frac{-x_i^2}{2}\right)}$$

Similarly when $\mu=10$, and $\sigma=4$

$$\int_{-\infty}^{\infty} \exp\left(-\frac{1}{2}\left(\frac{x-10}{4}\right)^2\right) dx$$

$$I_2 = C E_{p(x)} h(x).$$

$$\approx \boxed{\frac{b-a}{N} \sum_{i=1}^{N} \left(\exp\left(-\frac{1}{2}\left(\frac{x_i-10}{4}\right)^2\right)\right) \Rightarrow \frac{1}{N} \sum_{i=1}^{N} \left(\exp\left(-\frac{1}{2}\left(\frac{x_i-10}{4}\right)^2\right)\right)}$$

See python code attached

Scanned with CamScanner

Monte-Carlo Estimation for Problem 1a Assignment 2

In [10]:
```
from scipy import random
import numpy as np
```

(i) exp(-x)

In [11]:
```
a = 0
b = 1
N = 1000
xrand = np.zeros(N)

for i in range(len(xrand)):
    xrand[i] = random.uniform(a,b)

def func(x):
    return np.exp(-x)

integral = 0.0

for i in range(N):
    integral += func(xrand[i])

answer = (b-a)/float(N)*integral
print("the integral from 0 to 1 of exp(-x) is", answer)
```

```
the integral from 0 to 1 of exp(-x) is 0.6376685156211883
```

(ii) (1-x^2)^2

In [7]:
```
a = 0
b = 5
N = 1000
xrand = np.zeros(N)

for i in range(len(xrand)):
    xrand[i] = random.uniform(a,b)

def func(x):
    return (1-x**2)**2

integral = 0.0

for i in range(N):
    integral += func(xrand[i])

answer = (b-a)/float(N)*integral
print("the integral from 0 to 5 of (1-x^2)^2 is", answer)
```

```
the integral from 0 to 5 of (1-x^2)^2 is 556.3822219373099
```

Monte-Carlo Estimation for Problem 2 a

```
In [37]: import numpy as np
```

test with a given number of points and throw randomly in minimal bounding rectangle

```
In [38]: r = np.random.rand()
         #y = np.exp(-0.5*((x-mu)/sigma)^2)
         n1 = 0
         n = 100000
         sig = 1
         xmax = 4 * sig
         xmin = 0
         ymin = 0
         ymax = 1.0
         a = xmax * ymax
```

fill the points in the rectangle and count those folling inside the curve and compute the ratio of the points inside to total

```
In [39]: for i in range(n):
             x = np.random.rand()*(xmax - xmin)
             y = np.exp(-0.5*x*x/sig/sig)
             y1 = np.random.rand()*(ymax - ymin)
             if y1 < y:
                 n1 = n1 + 1
```

```
In [40]: a1 = n1*a/n
         ar = np.sqrt(2*np.pi)*sig
         print("monte carlo = ", 2*a1,", theoretical area = ", ar )
```

```
monte carlo =  2.5232 , theoretical area =  2.5066282746310002
```

Monte-Carlo Estimation for Problem 2 b

```
In [41]: import numpy as np
```

test with a given number of points and throw randomly in minimal bounding rectangle

```
In [42]: r = np.random.rand()
         #y = np.exp(-0.5*((x-mu)/sigma)^2)
         n1 = 0
         n = 100000
         mue = 10
         sig = 4
         xmax = 4 * sig
         xmin = 0
         ymin = 0
```

```
ymax = 1.0
a = xmax * ymax
```

fill the points in the rectangle and count those folling inside the curve and compute the ratio of the points inside to total

In [43]:
```
for i in range(n):
    x = np.random.rand()*(xmax - xmin)
    y = np.exp(-0.5*(x*x-mue*mue)/sig/sig)
    y1 = np.random.rand()*(ymax - ymin)
    if y1 < y:
        n1 = n1 + 1
```

In [44]:
```
a1 = n1*a/n
ar = np.exp(-0.5*(-mue*mue)/sig/sig)
print("monte carlo = ", 2*a1,", theoretical area = ", ar )
```

```
monte carlo =  22.80736 , theoretical area =  22.75989509352673
```