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
```