2.计算$\Phi(t)=\int_{-\infty}^{t}\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}dx$

从[t-[t]-2,t]取[t]+10个满足均匀分布的随机数，$t\leq-2$返回0

In [ ]:

```python
from scipy.stats import norm
from scipy.stats import uniform
from scipy.integrate import quad
import math

def phi(x):
    return math.e**(-x**2/2)/math.sqrt(2*math.pi)

# 准确值计算
def int_phi(t):
    if t < -2:
        return 0
    result,error  = quad(phi,t-math.floor(t)-2,t)
    return result

# 样本平均法计算
def SAA_phi(t):
    if t < -2:
        return 0

    n = 1000
    a = t - math.floor(t)-2
    b = t
    xn = uniform.rvs(loc = a, scale = b-a, size = n)
    result = sum((b-a)*phi(xn)/n)
    return result

# 重要性抽样
def IS_phi(t):
    # g(x)=e^{-x^2} if x < 0 else 2-e^{-x^2}
    def g(x):
        if x<0:
            return -1/(x-1)
        else:
            return 1/(x+1)
    def InverG(x):
        if 0<x<=1:
            return 1-1/x
        elif 1<x<=2:
            return 1-1/(2-x)
        else:
            exit(-1)
    X = uniform.rvs(loc = 0, scale = 1, size = 1000)
    X = [InverG(x) for x in X]
    result = sum([phi(x)/g(x) for x in X])/len(X)
    return result

# 误差估计
def error(realInt, simInt,x):
    result = [abs(realInt(x) - simInt(x))/realInt(x) for _ in range(1000)
    return sum(result)/len(result)
# 相对误差
```

```
print([error(int_phi,IS_phi,i) for i in range(5)])
print([error(int_phi,SAA_phi,i) for i in range(5)])
```

```
[0.35536744623032795, 0.6245078542974281, 0.6778727049984801, 0.6852830194
380101, 0.6857045475254053]
[0.012541708658464785, 0.010009002031641488, 0.012431501816253873, 0.01688
4170480061546, 0.020853438548700 36]
```

5.

$P(X > 20) = \int_{20}^{\infty} \phi(x)dx = \int_{-\infty}^{-20} \phi(x)dx, \phi(x) = \frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}$

In [ ]:
```
def SAA_phi_tail():
    rvs = uniform.rvs(loc = 20, scale = 1/20, size = 10000)
    result = sum([phi(rv) for rv in rvs])/len(rvs)
    return result
```

In [ ]:
```
int_phi_tail_value = quad(phi,20,20+1/20)[0]
SAA_phi_tail_value = SAA_phi_tail()
print("准确值：{}\nSAA估计值:{}\n绝对误差为:{}".format(int_phi_tail_value,SAA
```

```
准确值：1.7443984965185905e-89
SAA估计值:3.48146261279122e-88
绝对误差为:3.3070227631393614e-88
```

6.

(1).

$$E_f(h(x)) = \int_{-\infty}^{\infty} h(x)f(x)dx$$

其意义为，函数值h(x)的满足自变量概率密度函数为f(x)的数学期望

(2):用python代替

In [ ]:
```
def h(x):
    result = math.e**(-(x-3)**2/2)+math.e**(-(x-6)**2/2)
    return result

def g(x):
    return h(x)*phi(x)

def cal_h_f():
    rvs = norm.rvs(loc = 0, scale = 1, size = 1000)
    Accurate_value = quad(g,-100,100)[0]
    SAA_value = sum([h(rv) for rv in rvs])/len(rvs)
    error = abs(Accurate_value-SAA_value)
    return Accurate_value,SAA_value,error

cal_lst = [cal_h_f() for _ in range(1000)]
result = [[item[i] for item in cal_lst] for i in range(3)]
result = [sum(lst)/len(lst) for lst in result]
print("准确值:{}\n模拟值:{}\n相对误差:{}".format(result[0],result[1],result[
```

准确值:0.07461577032883317
模拟值:0.07461956898345597
相对误差:0.05055552259515488

(3):

In [ ]:
```
def IS_h():
    rvs = uniform.rvs(loc = -8, scale = 7, size = 1000)
    IS_value = 7*sum([g(rv) for rv in rvs])/len(rvs)
    Accurate_value = quad(g,-8,-1)[0]
    return Accurate_value,IS_value,abs(Accurate_value-IS_value)

cal_lst = [IS_h() for _ in range(1000)]
result = [[item[i] for item in cal_lst] for i in range(3)]
result = [sum(lst)/len(lst) for lst in result]

print("准确值:{}\n模拟值:{}\n相对误差:{}".format(result[0],result[1],abs(res
```

准确值:1.5164763694972008e-05
模拟值:1.5192010637912698e-05
相对误差:0.0017967271688899038

相对误差小于（2）的方法，说明模拟更加精确

随机投点法（Random shot point method）& 样本平均值法（Sample averaging approximately method）算法实现

In [ ]:
```
# random shot point method , function f, integral floor a, ceiling b, sup
def RSP(f, a, b):
    from numpy import zeros, mean
    from scipy.optimize import minimize
    nTrails = 100000 # 试验次数
    ceiling_value = -int(minimize(lambda x: -f(x), (a+b)/2, bounds=[(a,b)
    # 产生二维随机数
    random_array = zeros((2,nTrails))
    random_array[0,:] = uniform.rvs(loc = a, scale = b - a, size = nTrail
    random_array[1,:] = uniform.rvs(loc = 0, scale = ceiling_value , size
    under_f = f(random_array[0, :]) > random_array[1, :]
    p = mean(under_f) # 概率计算
    S = p * (b - a) * ceiling_value  # 积分值估计
    return S

# sample averaging approximately method
def  SAA(f,a,b):
    n = 100000 # 取点个数
    rvs = uniform.rvs(loc = a, scale = b-a, size = n)
    S = 1/n*(b-a)*sum([f(rv) for rv in rvs])
    return S

# 测试函数
def f(x):
    return 4*x**3

RSP_value, SAA_value, value = RSP(f,0,5), SAA(f,0,5), quad(f,0,5)[0]
print("RSP模拟值:{:.3f}\nSAA模拟值:{:.3f}\n准确值:{:.3f}".format(RSP_value,S
```

RSP模拟值:623.250
SAA模拟值:626.743
准确值:625.000

收敛性证明和误差估计

1.RSP方法，落在区域内的点的个数$n \sim B(N,p)$，从而有
$E(n)=Np,\sigma^2(n)=p(1-p)$

于是:$E(I')=E(M|D|p')=M|D|E(p')=M|D|\frac{E(n)}{N}=\frac{M|D|}{N}N_p=I$收敛

$\sigma^2(I')=\frac{(M|D|)^2}{N^2}\sigma^2(n)=\frac{(M|D|)^2}{N^2}Np(1-p)=\frac{1}{N}I(M|D|-I)\propto\frac{1}{N}$

2.SAA方法，
$E(I')=E(\frac{1}{N}(b-a)\sum f(X_i))=\frac{b-a}{N}\sum E(f(X_i))=(b-a)E(f(X_i))=I$收敛

这个不会，看不懂

采用课本中的重要性抽样即可，通过选取合适的$g_X(x)$,使得被积函数$\frac{f}{g_X}$的比值尽可能接近一，这样方差尽可能的小，从而减小误差

In [ ]:
```
# 以f(x)=4x^3在0到5的积分为例，选取g(x)=3x^2/125
def f(x):
    result = 4*x**3
    return result

def g(x):
    return 3*x**2/125

def h(x):
    return f(x)/g(x)

def inverse_G(x):
    result = 5*x**(1/3)
    return result

def IS(f,g,inverse_G, a,b):
    rvs = uniform.rvs(0,1,size = 100000)
    rvs = [inverse_G(rv) for rv in rvs]
    result = sum(h(rv) for rv in rvs)/len(rvs)
    return result

IS(f,g,inverse_G,0,1)
```

Out[ ]: 624.9313888562581

与准确值相差无几