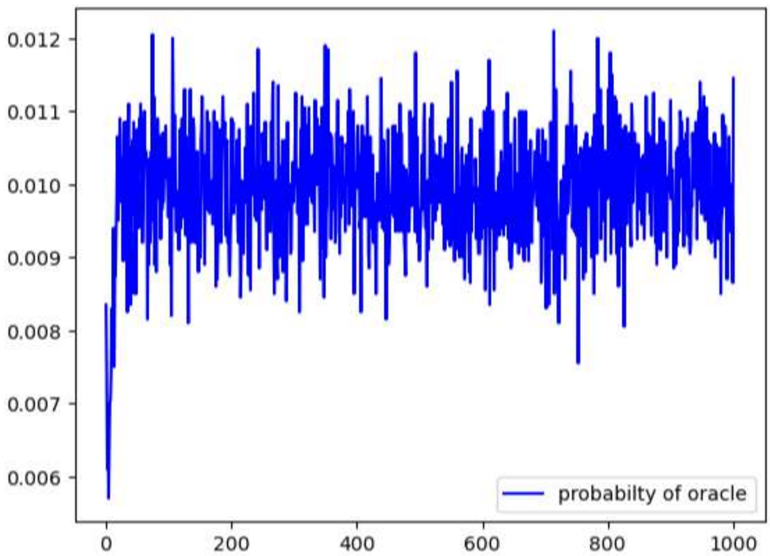
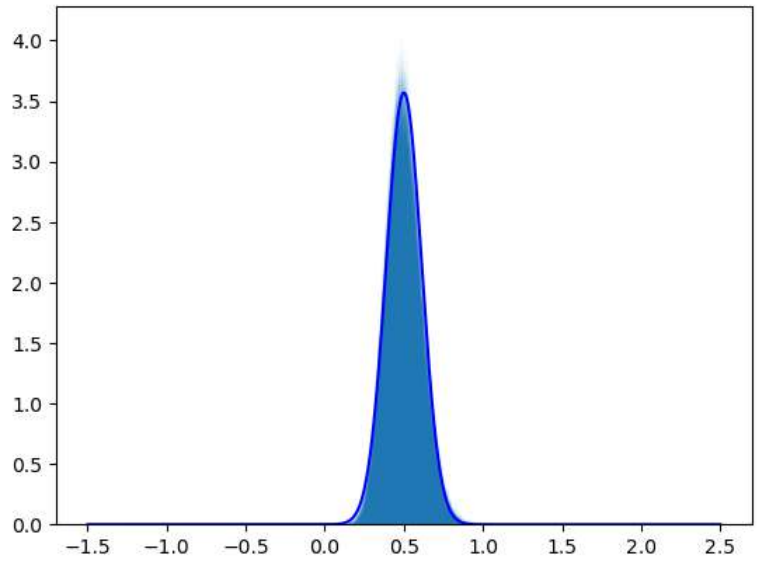
In [2]:

```
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
from scipy import integrate
from scipy.stats import norm
import numpy.ma as ma
N = 20000
n = 100
k = 100
ro = 0.5
p1 = 0.9
p2 = 0.5
p = 0.0001
fig, ax = plt.subplots(1, 1)
def f(k):
    x = np.random.binomial(1, p, N*n)
    y1 = np.random.binomial(k, p1, N*n)
    y2 = np.random.binomial(k, p2, N*n)
    res = ma.masked_array(y2, mask=x).filled(0)
    res +=  ma.masked_array(y1, mask=1 - x, fill_value = 0).filled(0)

    temp = np.repeat(np.maximum.reduceat(res,np.r_[:N*n:n]), n)
    temp = (res == temp)
    temp = (temp & x)
    temp = np.add.reduceat(temp,np.r_[:N*n:n])
    return np.mean(temp > 0)
f = np.vectorize(f)
min_k = 1
max_k = 1000
x = np.linspace(min_k, max_k, max_k - min_k + 1)
ax.plot(x, f(x), 'b-',label='probabilty of oracle')
ax.legend()
plt.show()
```



In [97]:

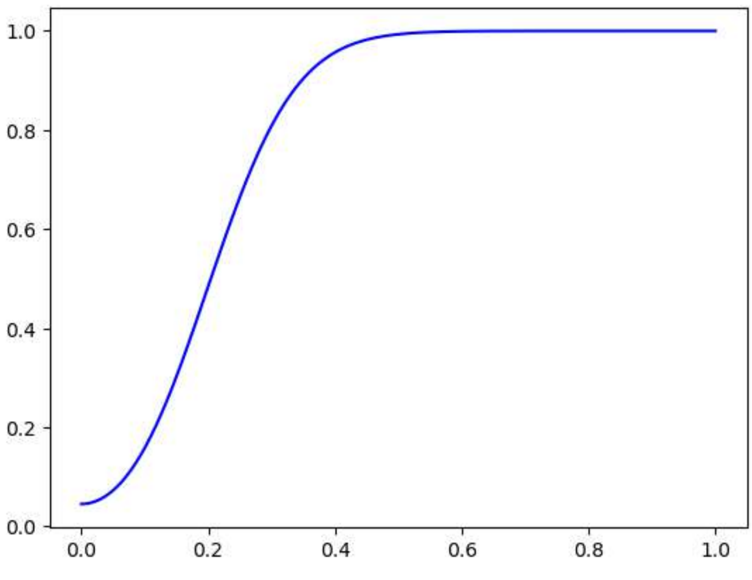
```
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
from scipy import integrate
from scipy.stats import norm

N = 1000000
n = 100
ro = 0.5
fig, ax = plt.subplots(1, 1)
mu1 = 0
sigma1 = np.sqrt((1 + ro)/2)
mu2 = 0
sigma2 = np.sqrt((1 - ro)/2)
x = np.random.normal(mu1, sigma1, N*n)
y = np.random.normal(mu2, sigma2, N*n)
z = (x**2 - y**2)
z = np.add.reduceat(z, range(0, N*n, n))
z = z / n
z = np.sort(z)
x = np.linspace(-2 + ro, 2 +ro, 10000)
ax.plot(x, norm.pdf(x, loc = ro, scale = np.sqrt((1 + ro**2)/(n))), 'b-',label='theoretical distribution')
ax.hist(z, histtype='stepfilled', density = True, bins = 10000, label = 'empirical distribution')
ax.legend()
plt.show()
```



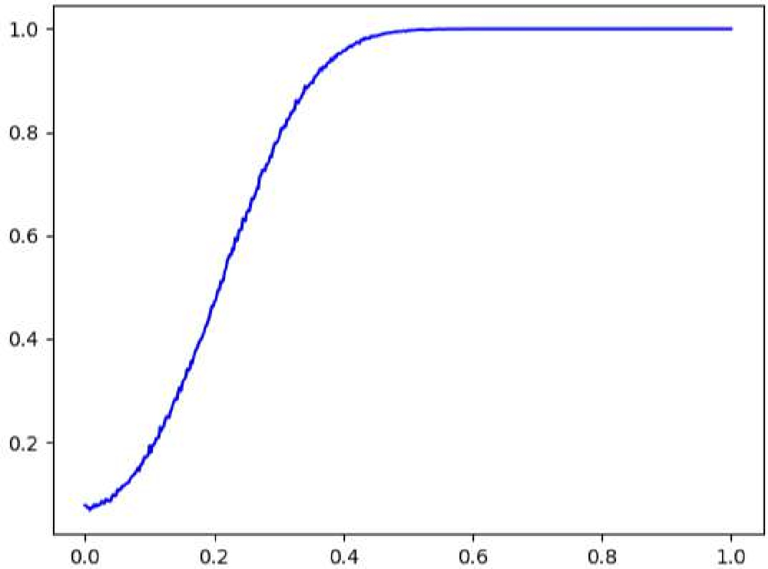
In [33]:
```
import torch
print(torch.version.cuda)
```

```
12.1
```

In [122]:
```
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
from scipy import special
from scipy import integrate
from scipy.stats import norm
ro = 0
n = 100
x = np.linspace(0, 1, 10000)
fig, ax = plt.subplots(1, 1)
def f(x):
    return 0.5* (special.erf(((ro - x) + 2*np.sqrt((1 + x**2)/n))/np.sqrt(2*(1 + x**2)/n)) - special.erf(((ro - x) - 2*np.sqrt((1
ax.plot(x, 1 - f(x), 'b-',label='theoreical power of criteria')
ax.legend()
plt.show()
```



In [123]:
```
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
from scipy import integrate
from scipy.stats import norm

N = 10000
n = 100
ro = 0.5
fig, ax = plt.subplots(1, 1)
def f(u):
    mu1 = 0
    sigma1 = np.sqrt((1 + ro)/2)
    mu2 = 0
    sigma2 = np.sqrt((1 - ro)/2)
    x = np.random.normal(mu1, sigma1, N*n)
    y = np.random.normal(mu2, sigma2, N*n)
    z = (x**2 - y**2)
    z = np.add.reduceat(z, range(0, N*n, n))
    z = z / n
    ret = (((ro - u) - 2*np.sqrt((1 + u**2)/n) < z) & (z < (ro - u) + 2*np.sqrt((1 + u**2)/n))).sum()
    ret /= len(z)
    return ret
f = np.vectorize(f)
x = np.linspace(0, 1, 500)
ax.plot(x, 1 - f(x), 'b-',label='empirical power of criteria')
ax.legend()
plt.show()
```



In [180]:
```
import matplotlib.pyplot as plt
import cupy as np
import scipy as sp
from scipy import integrate
from scipy.stats import norm

N = 500
k = 500
n = 12
ro = 0.9
fig, ax = plt.subplots(1, 1)
def f(u):
    mu1 = 0
    sigma1 = 1
    mu2 = 0
    sigma2 = 1
    x = np.random.normal(mu1, sigma1, n * N)

    x_mean = np.add.reduceat(x, range(0, n * N, n))/n
    x_mean = np.repeat(x_mean, n)
    x -= x_mean
    ans = []
    for i in range(0, N):
        xn =  x[i * n : (i + 1) * n]
        k = 0
        roi = -1
        while (roi < u and k < 1000):
            y = np.random.normal(mu2, sigma2, n)
            y_mean = np.array([np.mean(y)])

            y_mean = np.repeat(y_mean, n)

            y -= y_mean
            roi = np.sum(y * xn)/n
            k+=1
        ans+=[k]
    return np.mean(ans)
f = np.vectorize(f)
x = np.linspace(0, 0.95, 50)
ax.plot(x, f(x), 'b-',label='empirical expectaion of tries')
plt.show()
```