Monte Carlo Lab 2 Submission

Chaitanya Chhabra, 220123012

August 7, 2024

[13]:
```python
import numpy as np
from matplotlib import pyplot as plt
from bisect import bisect
from scipy.integrate import quad as integrate
```

[14]:
```python
# Backbone: General Linear Congruence Generator
def GLCG(x0):
    a,b,m = 625,6571,31104
    while True:
        x0 = (a*x0+b)%m
        yield x0/m

glcg = GLCG(69)

def U(a,b):
    return a+(b-a)*next(glcg)
```

1 Task 1

1.0.1 Let's define the probability density f(x) and cumulative distribution F(x) functions as given

1.0.2 Clearly, inverse of $F(x) = 1 - (1 - x)^3$ is

$$F^{-1}(x) = 1 - (1 - x)^{1/3}$$

[15]:
```python
f = lambda x: 3*(1-x)**2
F = lambda x: 1-(1-x)**3
F_INV = lambda x: 1-(1-x)**(1/3)
```

1.0.3 True mean is EX, given by

$$\mu = \int_{-\infty}^{\infty} xf(x)dx$$

1.0.4 True variance is $EX^2 - (EX)^2$, given by

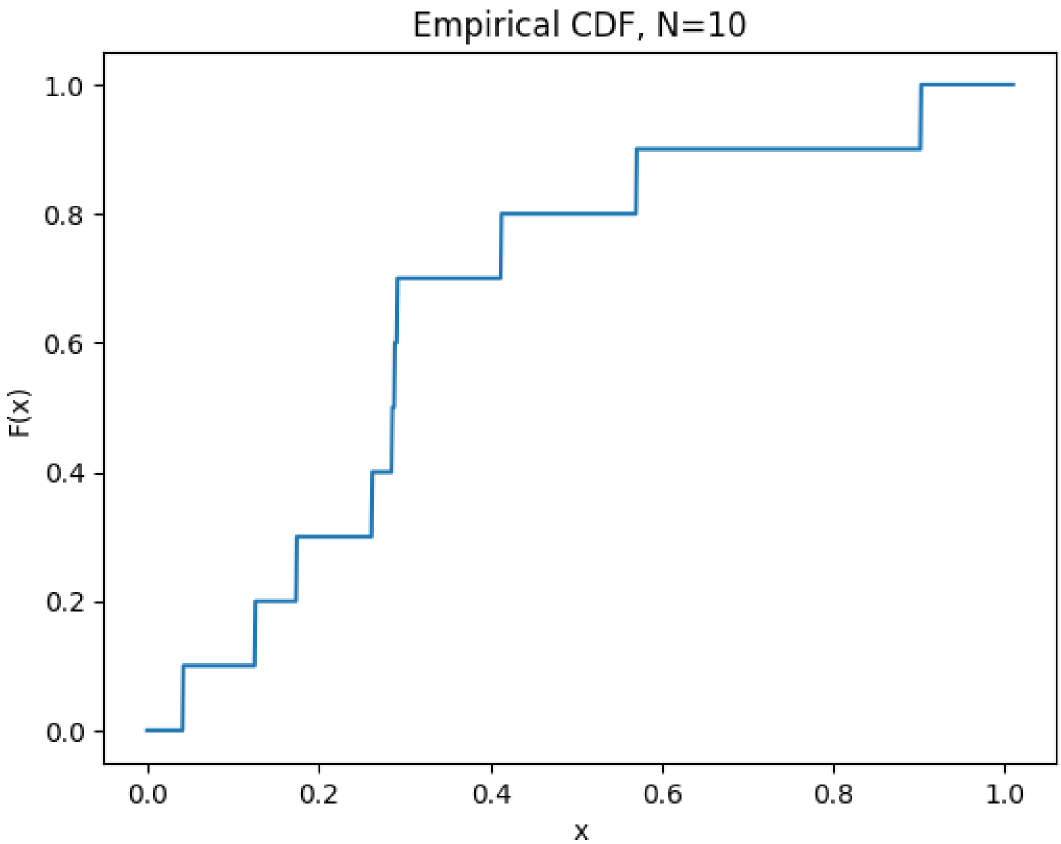
$$VX = \int_{-\infty}^{\infty} x^2 f(x)dx - \mu^2$$

[16]:
```
true_mean = integrate(lambda x: x*f(x), 0,1)[0]
true_var  = integrate(lambda x: x*x*f(x), 0,1)[0] - true_mean**2
true_var = round(true_var,3)

print(f"True Mean {true_mean}")
print(f"True Variance {true_var}")
```

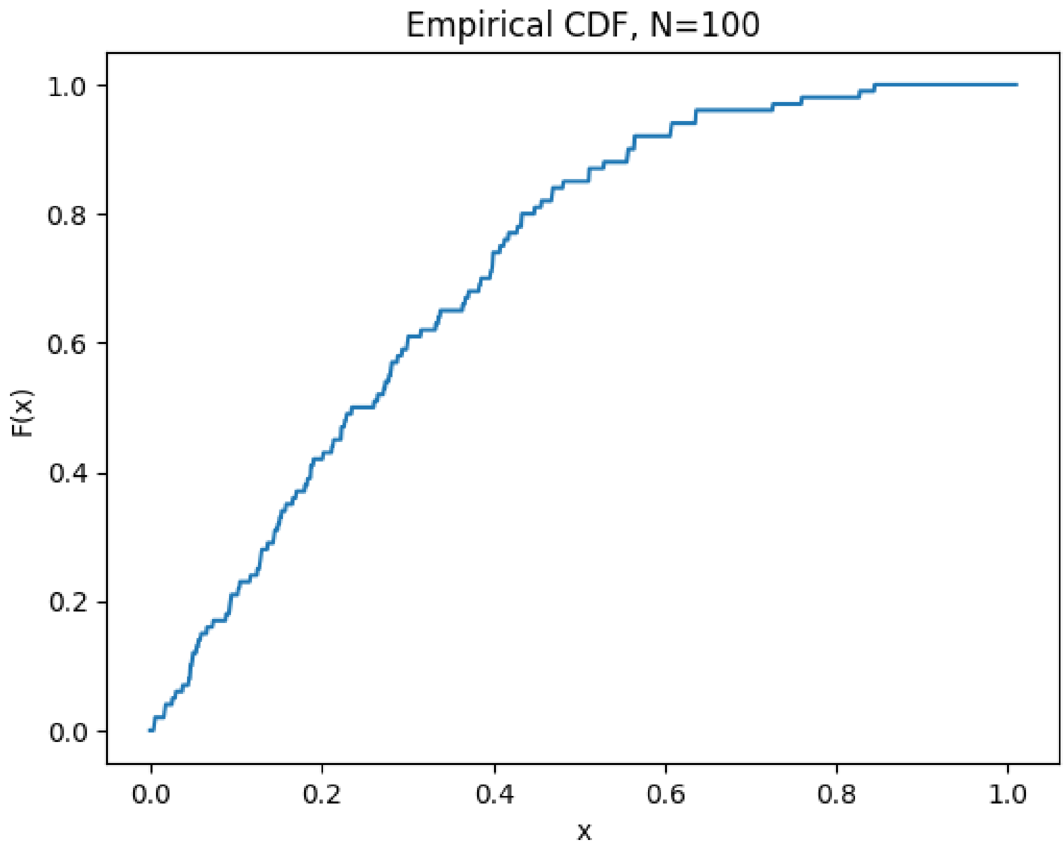
```
True Mean 0.25
True Variance 0.037
```

1.0.5 In inverse transform sampling, we will generate from $U(0,1)$, and apply F^{-1} before storing

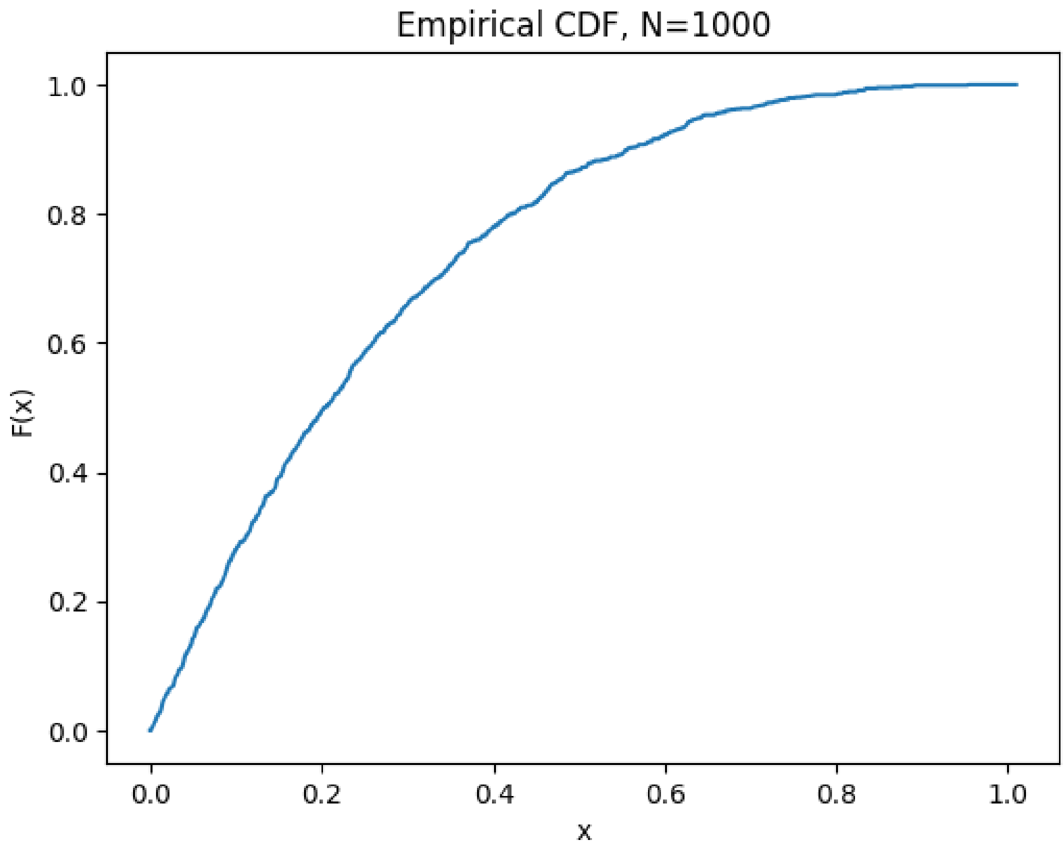
[17]:
```
for N in (10,100,1000,10_000,100_000):
    plt.title(f"Empirical CDF, N={N}")
    plt.xlabel("x")
    plt.ylabel("F(x)")
    Xs = []
    for _ in range(N):
        X = F_INV( U(0,1) )
        Xs.append(X)
    Xs.sort()

    # Bisect module does binary search to find the range where generated number␣
 ↪falls
    x = np.linspace(-0.001,1.01,1000)
    y = np.array([*map(lambda i: bisect(Xs,i), x)])/N
    plt.plot(x,y)
    plt.show()

    Xs = np.array(Xs)
    print(f"Mean is {round(Xs.mean(),3)}, should be {true_mean}")
    print(f"Error = {round(100*abs(Xs.mean()-true_mean),2)}%")
    print(f"Variance is {round(Xs.var(),2)}, should be {true_var}")
    print(f"Error = {round(100*abs(Xs.var()-true_var),2)}%")
```



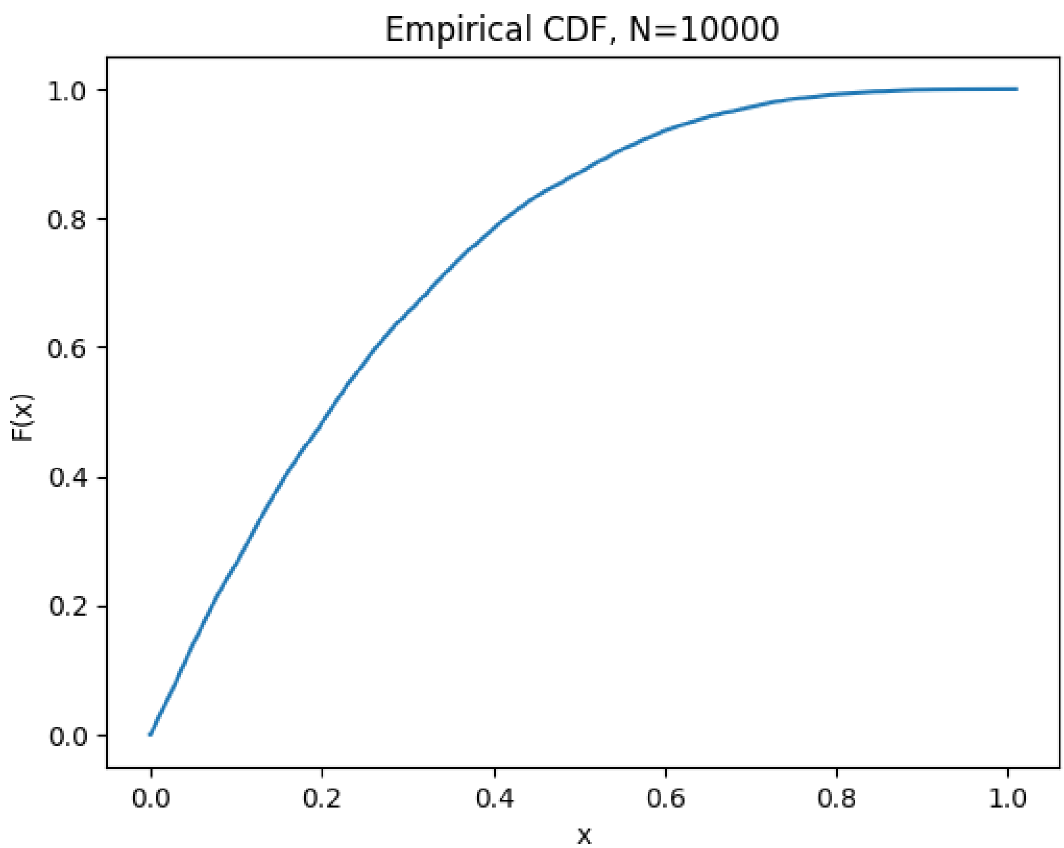

```
Mean is 0.335, should be 0.25
Error = 8.49%
Variance is 0.06, should be 0.037
Error = 1.82%
```



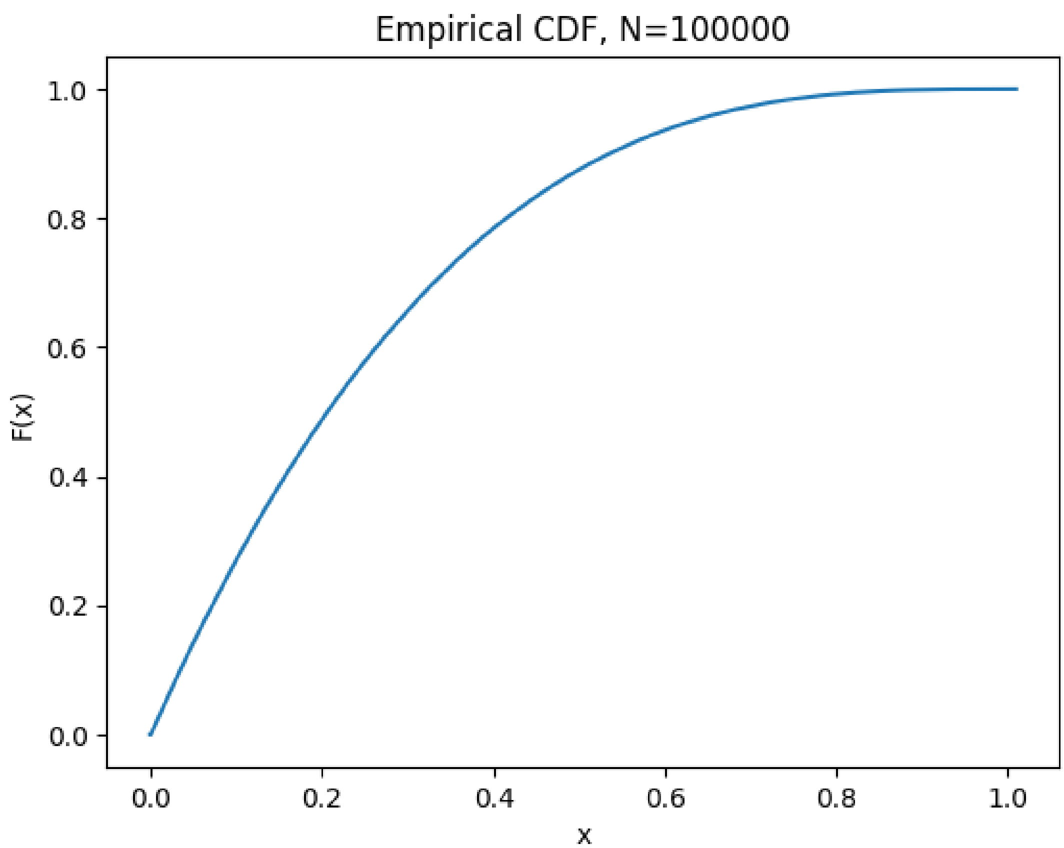

```
Mean is 0.281, should be 0.25
Error = 3.08%
Variance is 0.04, should be 0.037
Error = 0.24%
```




```
Mean is 0.253, should be 0.25
Error = 0.27%
Variance is 0.04, should be 0.037
Error = 0.34%
```




```
Mean is 0.251, should be 0.25
Error = 0.15%
Variance is 0.04, should be 0.037
Error = 0.08%
```




```
Mean is 0.25, should be 0.25
Error = 0.01%
Variance is 0.04, should be 0.037
Error = 0.04%
```

1.1 Task 2

1.1.1 Same as earlier, define f(x) and F(x).

1.1.2 As for F^{-1}, it is piecewise as well, and we see that the switching point is $F(1)$ i.e $1 - 1/e$

1.1.3 The inverse thus becomes

1.1.4 $F^{-1} = x \mapsto -\ln(1-x)$ if $x \leq F(1)$ else $\frac{1}{2}(1 - \ln(1-x))$

[18]:
```
switch = 1-np.exp(-1)
f = lambda x: (0<x<=1) * (np.exp(-x)) + (1<x) * (2*np.exp(1-2*x))
F = lambda x: (0<x<=1) * (1-np.exp(-x)) + (1<x) * (1-np.exp(1-2*x))
F_INV = lambda x: (0<x<=switch) * (-np.log(1-x)) + (switch<x) * (1-np.log(1-x))/
 ↪2
```

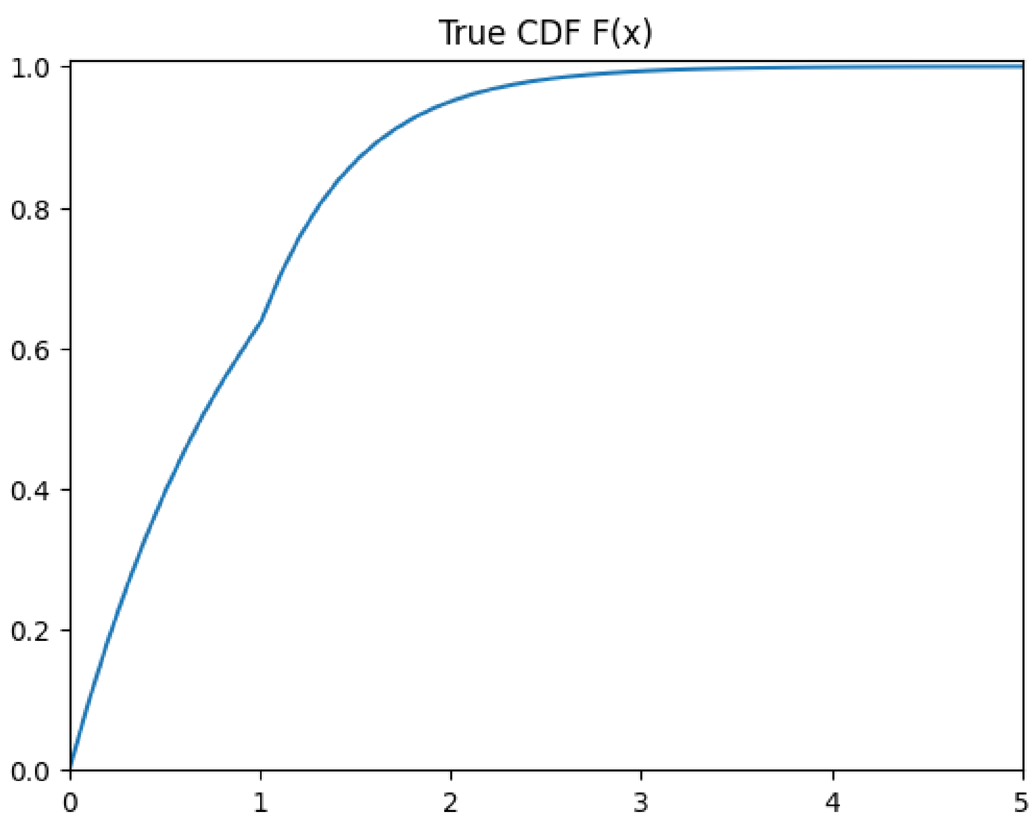
1.1.5 Earlier we took the expectation and variance integrals from 0 to 1, now we need to take them from 0 to infinity

[19]:
```
inf = float("inf")
true_mean = integrate(lambda x: x*f(x),   0,inf)[0]
true_var  = integrate(lambda x: x*x*f(x), 0,inf)[0] - true_mean**2
true_var  = round(true_var,3)

print(f"True Mean {true_mean}")
print(f"True Variance {true_var}")
```

```
True Mean 0.816060279414278
True Variance 0.414
```

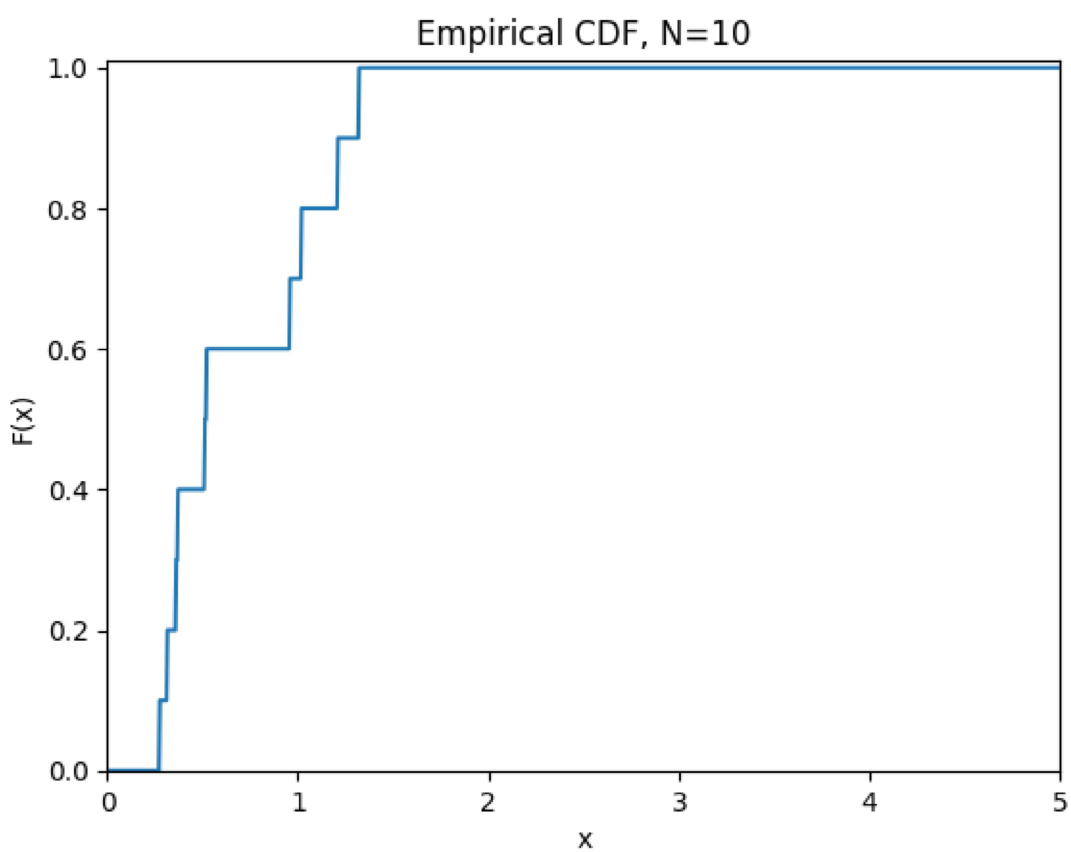
1.1.6 Plot of true CDF

[20]:
```
X = np.linspace(0,10,100)
Y = y = np.array([*map(lambda i: F(i), X)])
plt.title("True CDF F(x)")
plt.xlim(0,5); plt.ylim(0,1.01)
plt.plot(X, Y)
plt.show()
```

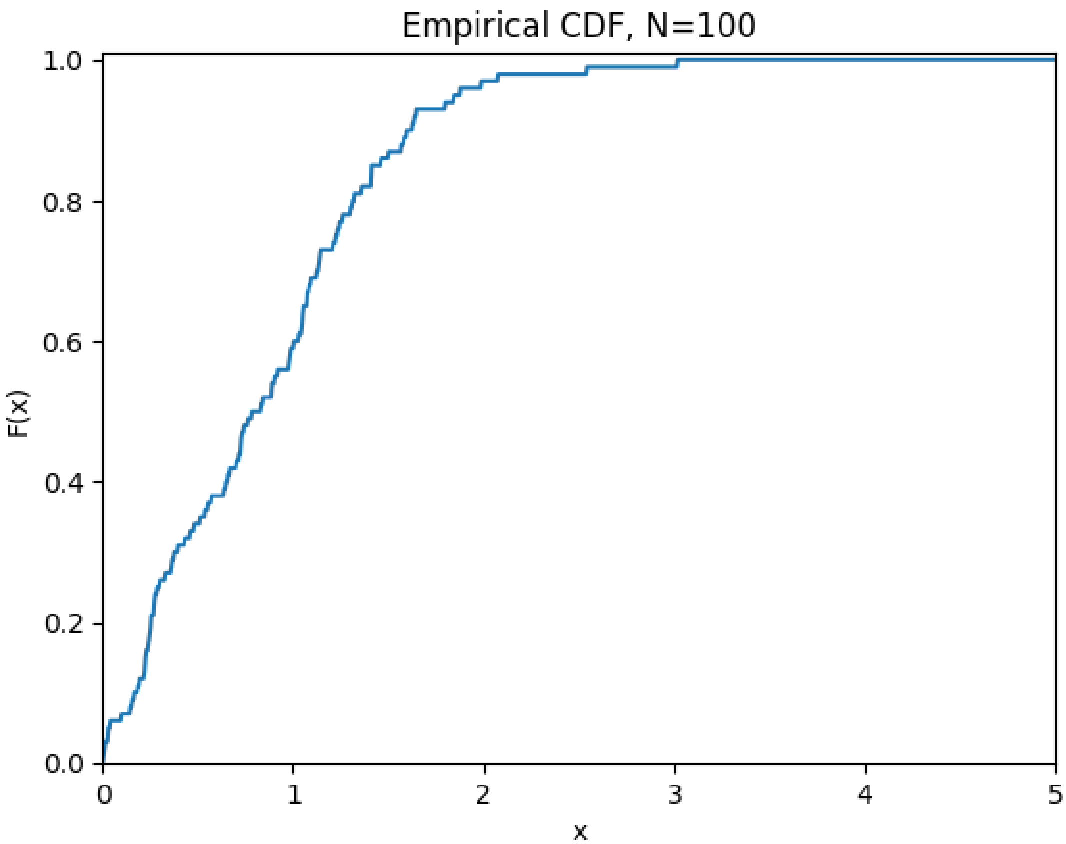


1.1.7 Empirical Plots

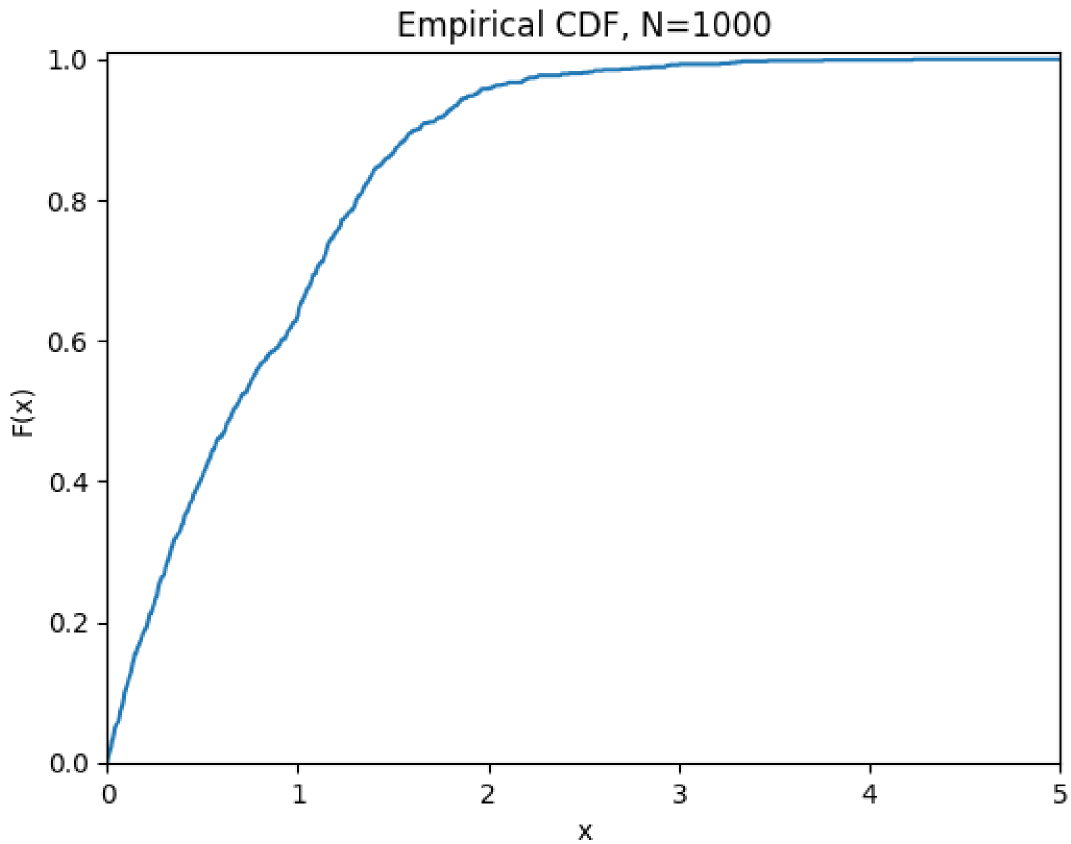
[21]:
```
for N in (10,100,1000,10_000,100_000):
    plt.title(f"Empirical CDF, N={N}")
    plt.xlabel("x")
    plt.ylabel("F(x)")
    Xs = []
    for _ in range(N):
        X = F_INV( U(0,1) )
        Xs.append(X)
    Xs.sort()
    x = np.linspace(-0.01,5.01,1000)
    y = np.array([*map(lambda i: bisect(Xs,i), x)])/N
    plt.xlim(0,5); plt.ylim(0,1.01)
    plt.plot(x,y)
    plt.show()

    Xs = np.array(Xs)
    print(f"Mean is {round(Xs.mean(),3)}, should be {true_mean}")
    print(f"Error = {round(100*abs(Xs.mean()-true_mean),2)}%")
    print(f"Variance is {round(Xs.var(),2)}, should be {true_var}")
    print(f"Error = {round(100*abs(Xs.var()-true_var),2)}%")
```



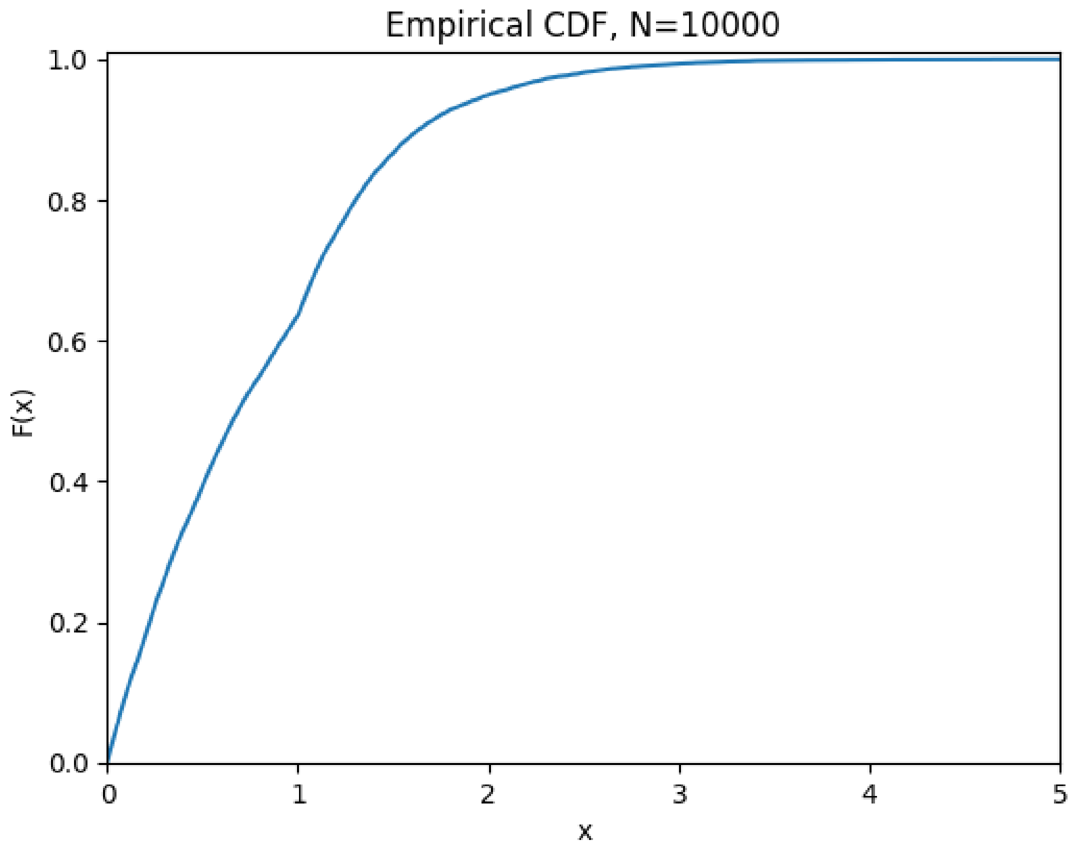
```
Mean is 0.685, should be 0.816060279414278
Error = 13.09%
Variance is 0.14, should be 0.414
Error = 27.09%
```



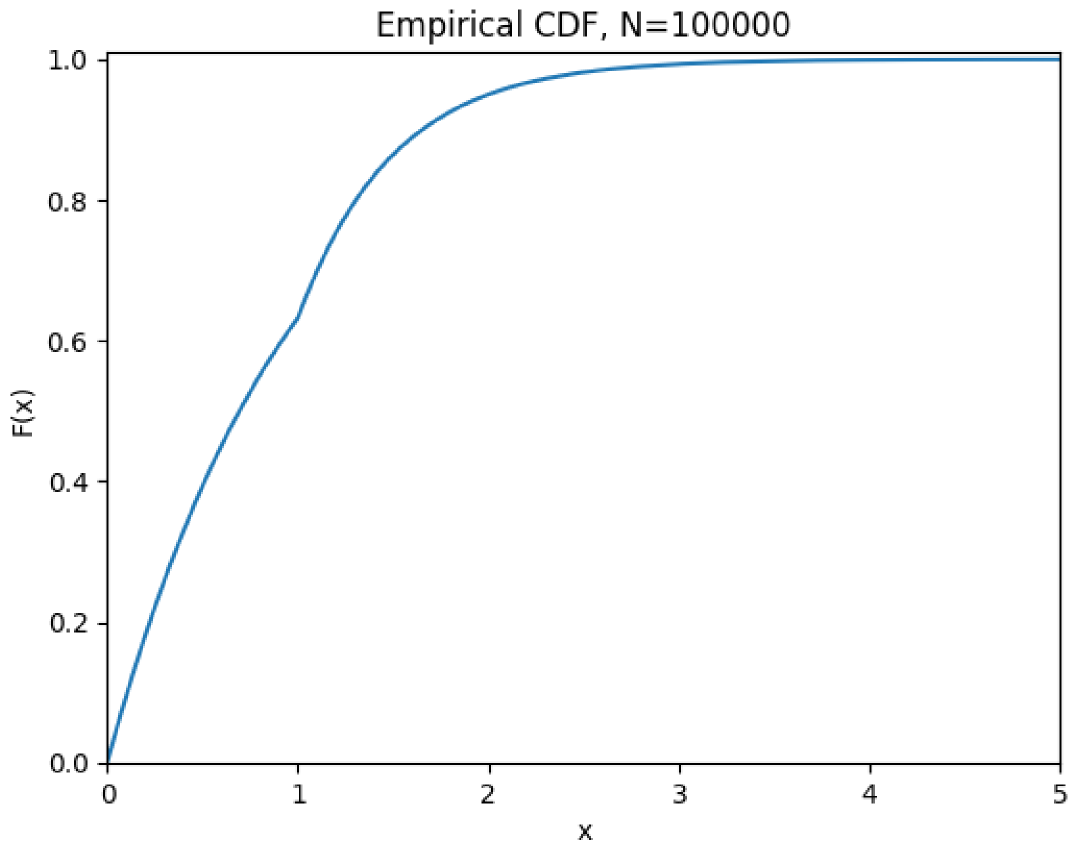

```
Mean is 0.851, should be 0.816060279414278
Error = 3.53%
Variance is 0.35, should be 0.414
Error = 6.16%
```




```
Mean is 0.801, should be 0.816060279414278
Error = 1.55%
Variance is 0.41, should be 0.414
Error = 0.02%
```




```
Mean is 0.813, should be 0.816060279414278
Error = 0.32%
Variance is 0.41, should be 0.414
Error = 0.35%
```




```
Mean is 0.817, should be 0.816060279414278
Error = 0.05%
Variance is 0.41, should be 0.414
Error = 0.1%
```

1.2 Task 3

Using the rules $U(ka, kb) = kU(a, b)$ and $U(x + a, x + b) = x + U(a, b)$, it is quite simple to see that

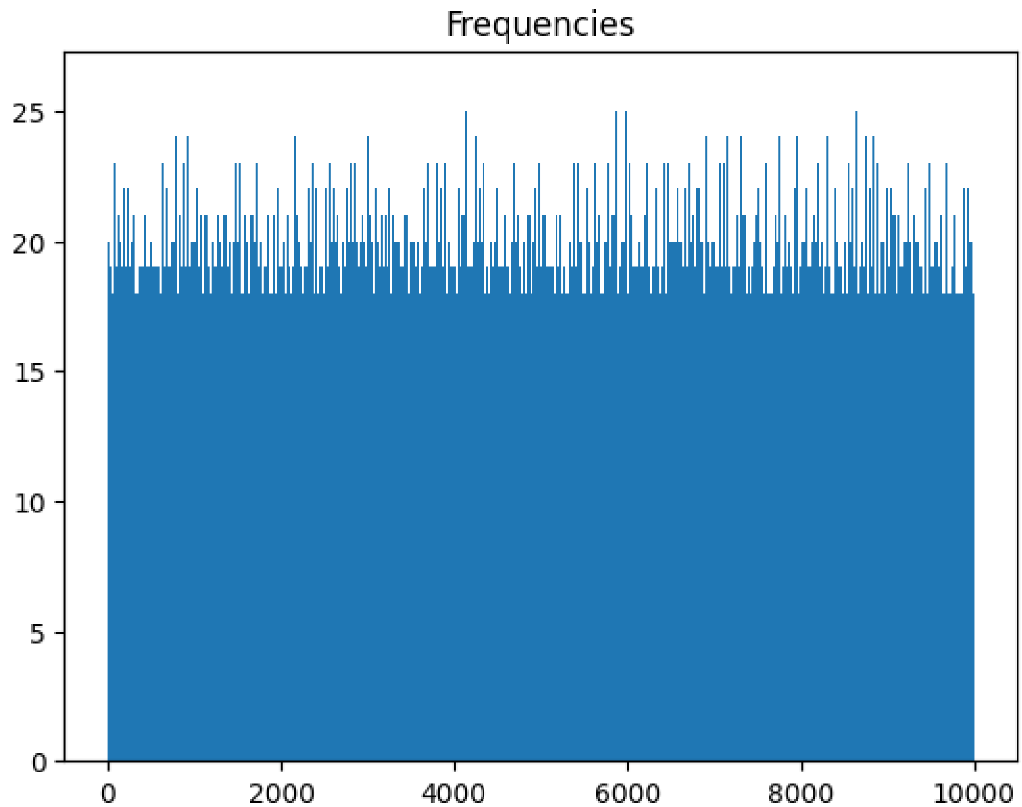
$$U\{1, 3...9999\} = 2 \cdot U\{0, 1...4999\} + 1$$

And $U\{0, 1...4999\}$ can be easily made from $\lfloor U(0, 5000)\rfloor$, which in turn equals $\lfloor 5000 \cdot U(0, 1)\rfloor$

1.2.1 Therefore, the required distribution is

$$2\lfloor 5000 \cdot U(0, 1)\rfloor + 1$$

[22]:
```
N = 10**5
X = []

for _ in range(N):
    x = 2*int(U(0,1)*5000)+1
    X.append(x)

plt.title("Frequencies")
plt.hist(X, bins=5000, align='mid')
plt.show()
```



[23]:
```
X = np.array(X)
print(f"Min is {X.min()}, Max is {X.max()}")
print(f"Mean is {round(X.mean(),3)}, should be {5000}")
```

```
Min is 1, Max is 9999
Mean is 5000.953, should be 5000
```