RNG

September 14, 2022

[120]:
```
import numpy as np
import matplotlib.pyplot as plt
import sympy as sy
from IPython.display import Math, display

#np.random.seed(42)
```

1 Problem 1: LCG Random Number Generator:

[121]:
```
def LCG(N, m, a, n0 = 1):
    """
    n0 = initial seed
    N = N random numbers to generate (int)
    m = modulus
    a = multiplier
    """
    n = np.empty(N,)
    #assume n0 = 1 for all problems
    n[0] = n0
    for i in range(1, N):
        n[i] = (a * n[i - 1]) % m

    return n
```

1.0.1 Part I

[122]:
```
print("(a) ", LCG(10, 2**3, 2))
```

```
(a)  [1. 2. 4. 0. 0. 0. 0. 0. 0. 0.]
```

[123]:
```
print("(b) ", LCG(10, 2**3, 4))
```

```
(b)  [1. 4. 0. 0. 0. 0. 0. 0. 0. 0.]
```

(c) The sequence becomes 0 rapidly after few numbers generated

1.0.2 Part II

[124]:
```
print("(a) ", LCG(10, 2**3, 3))
```

```
(a)  [1. 3. 1. 3. 1. 3. 1. 3. 1. 3.]
```

[125]:
```
print("(b) ", LCG(10, 2**3, 5))
```

```
(b)  [1. 5. 1. 5. 1. 5. 1. 5. 1. 5.]
```

(c) The period (length of non-repeating sequence) of the RNG is 2

1.0.3 Part III

[126]:
```
print(LCG(10, 2**4, 3))
print("For a = 3, m = 2^4, period = 4")
print(LCG(15, 2**5, 3))
print("For a = 3, m = 2^5, period = 8")
```

```
[ 1.  3.  9. 11.  1.  3.  9. 11.  1.  3.]
For a = 3, m = 2^4, period = 4
[ 1.  3.  9. 27. 17. 19. 25. 11.  1.  3.  9. 27. 17. 19. 25.]
For a = 3, m = 2^5, period = 8
```

1.0.4 Part IV

My numerical experiments do support that claim that the period of an LCG RNG with $m = 2^k$ and odd a is 2^{k-2}.

1.0.5 Part V

RANDU RNG uses $a = 65539$ and $m = 2^{31}$, so period $= 2^{29} = 5.3687x10^8 < 1$ billion

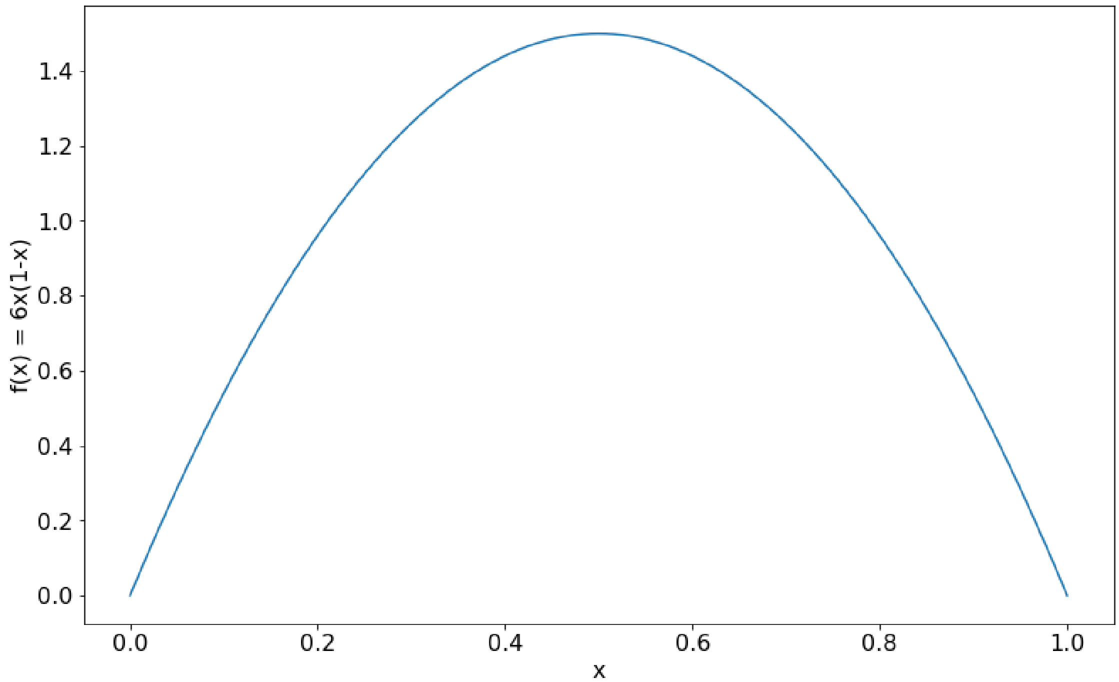
2 Problem 2: Sampling 1D Distribution:

###

$f(x) = 6x(1-x), 0 \leq x \leq 1$

2.1 (a) accept-reject method:

[127]:
```
xi = np.linspace(0, 1, 1000); fi = 6.* xi * (1. - xi)
plt.rcParams['figure.figsize'] = (13,8)
plt.rcParams['font.size'] = 16
plt.plot(xi, fi)
plt.xlabel("x")
plt.ylabel("f(x) = 6x(1-x)")
plt.show()
print('f_max = 1.5 at x = 0.5')
```



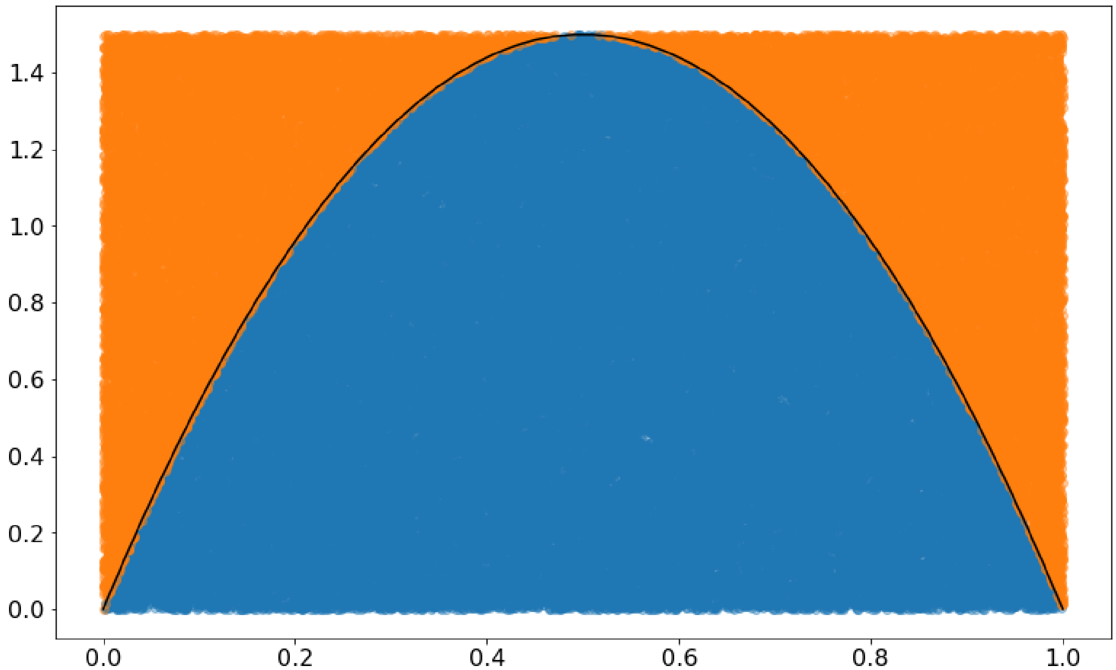

```
f_max = 1.5 at x = 0.5
```

[128]:

```
def acceptReject(samples, isPlot = True):
    xmin = 0.
    xmax = 1.
    fmax = 1.5

    x = np.random.uniform(xmin, xmax, samples)
    u = np.random.uniform(0., fmax, samples)
    fx = 6.* x * (1. - x)

    y = x[u <= fx]

    if isPlot:
        xi = np.linspace(0, 1); fi = 6.* xi * (1. - xi)
        plt.plot(y, u[u < fx],'o', alpha=0.4) #all points below f(x)
        plt.plot(x[u > fx], u[u > fx],'o', alpha=0.4) #all pts above f(x)
        plt.plot(xi, fi,'k-') #f(x)

    return y
```

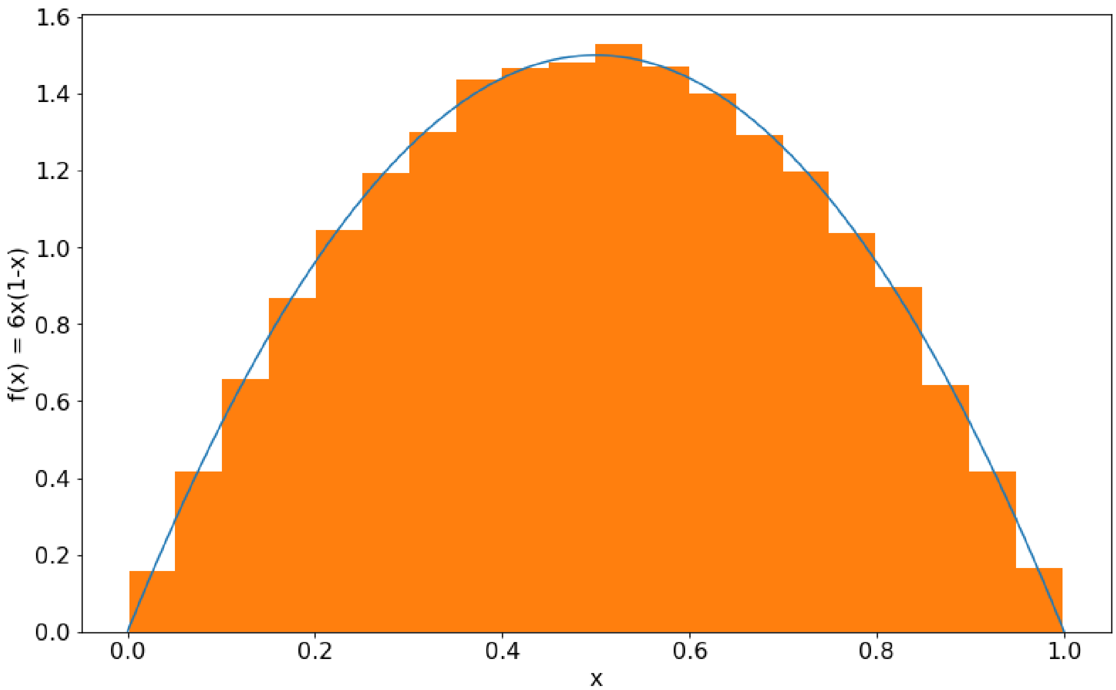
[129]:

```
#generate samples
x = acceptReject(10**5)
```



[130]:

```
plt.plot(xi, fi)
plt.hist(x, 20, density = True)
plt.xlabel("x")
plt.ylabel("f(x) = 6x(1-x)")
plt.show()
```



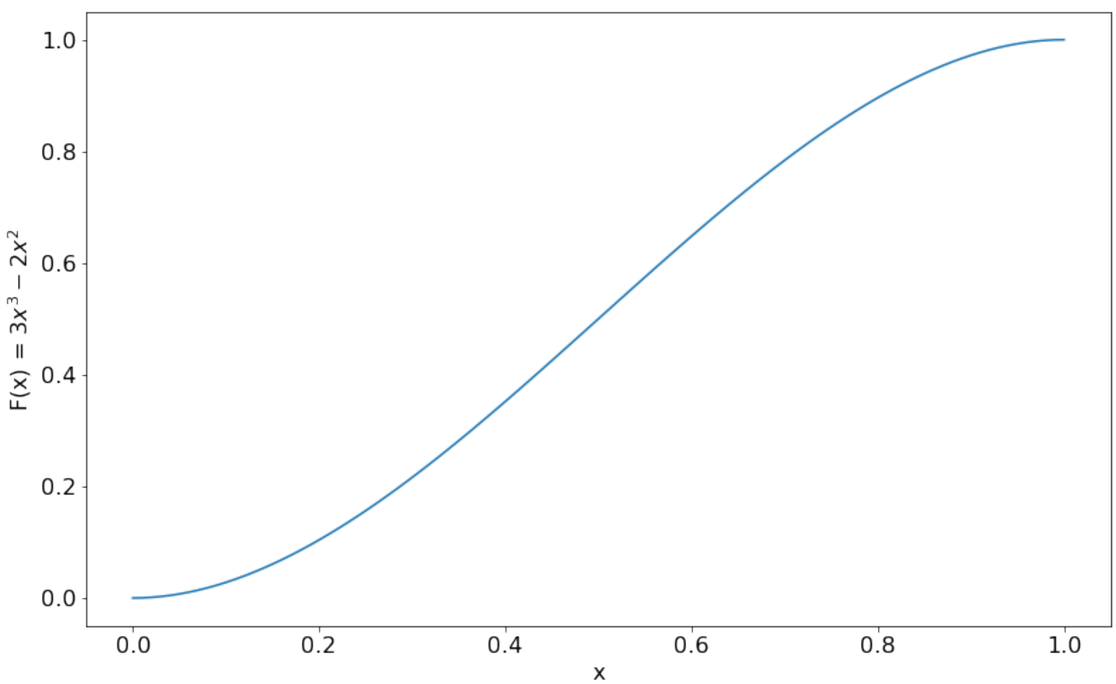
2.2 (b) transformation method:

[131]:
```
x = sy.symbols('x')
f = 6.* x * (1. - x)
F = f.integrate(x)
display(Math('F(x) = ' +  sy.latex(F)))
```

$F(x) = -2.0x^3 + 3.0x^2$

[137]:
```
#Generate table of xi and F(xi) values
xi2 = np.linspace(0, 1, 100); yi = 3.* xi2 ** 2 - 2. * xi2 ** 3
#values = np.array((xi, yi)).transpose()
u = np.random.uniform(0, 1, 10**5)
```

[138]:
```
plt.plot(xi2, yi)
plt.xlabel("x")
plt.ylabel(r"F(x) = $3x^3 - 2x^2$")
plt.show()
```



[142]:
```
#inverse interpolate F(x) using the table of xi's and yi's
from scipy import interpolate, optimize
```

```
invInterpF = interpolate.interp1d(yi, xi2) #(yi, xi) order since we want x =␣
    ↪F^-1(u)
x = invInterpF(u)
```

[141]:
```
plt.plot(xi, fi)
plt.hist(x, 20, density = True)
plt.xlabel("x")
plt.ylabel("f(x) = 6x(1-x)")
plt.show()
```