PS7

November 20, 2021

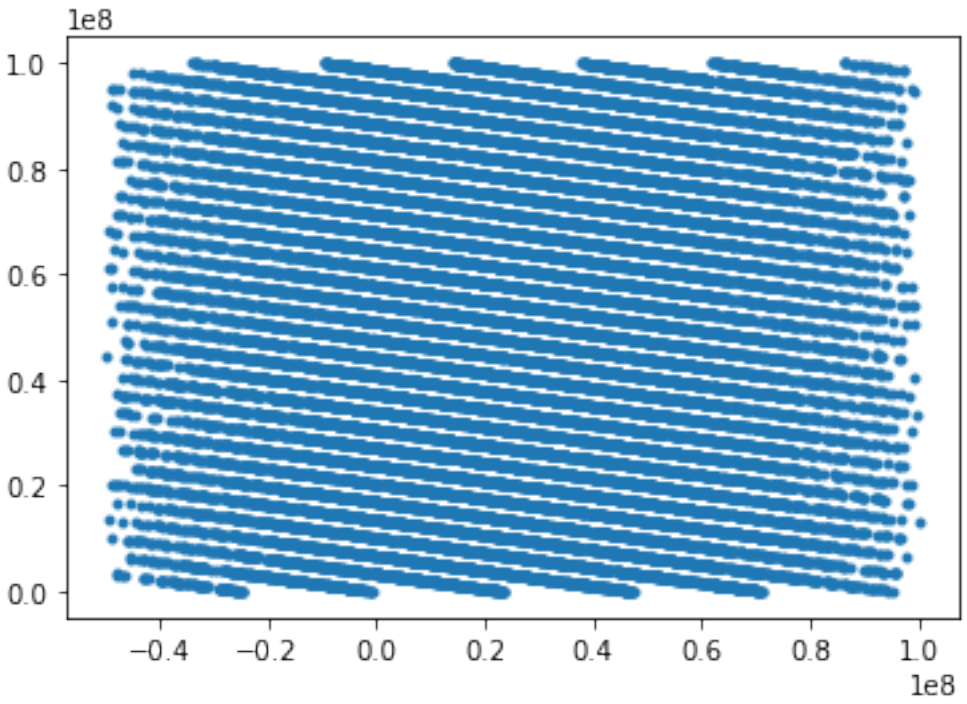
[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import ctypes
import numba as nb
```

0.1 Question 1

[2]:
```
data = np.loadtxt('rand_points.txt').T
xdata, ydata, zdata = data
```

[3]:
```
plt.plot(xdata-0.5*ydata, zdata, '.')
```

[3]:
```
[<matplotlib.lines.Line2D at 0x12a16c828>]
```

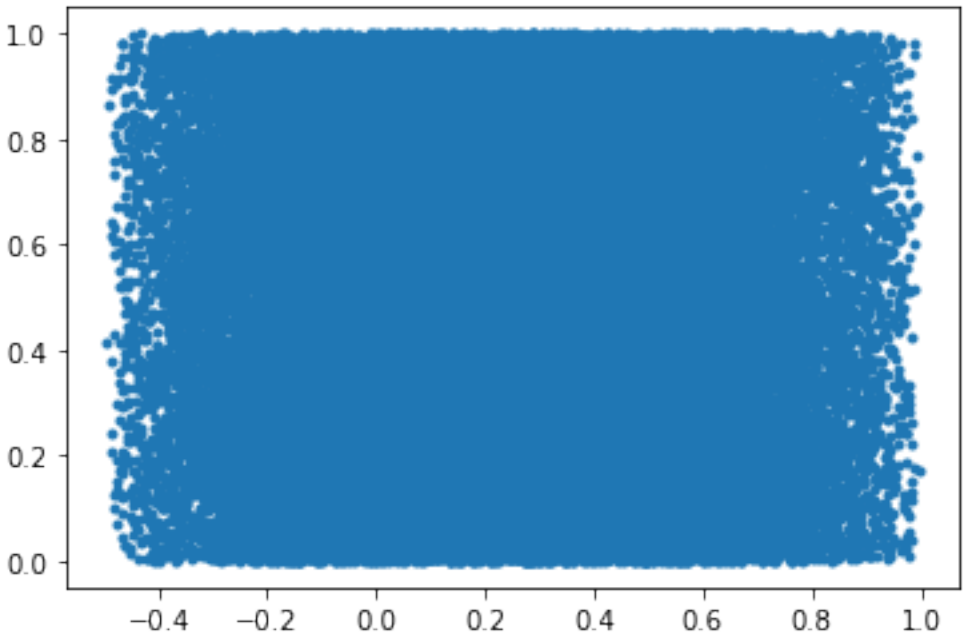


Choosing $a = 1$, $b = -0.5$ we can clearly see the individual planes in the plot.

[4]:
```
n = 50000
xdata = np.random.rand(n)
ydata = np.random.rand(n)
zdata = np.random.rand(n)
```

[5]:
```
plt.plot(xdata-0.5*ydata, zdata, '.')
```

[5]:
```
[<matplotlib.lines.Line2D at 0x129beda20>]
```



Playing around with a,b, we see no obvious patterns so it is a good pseudo-random number generator.

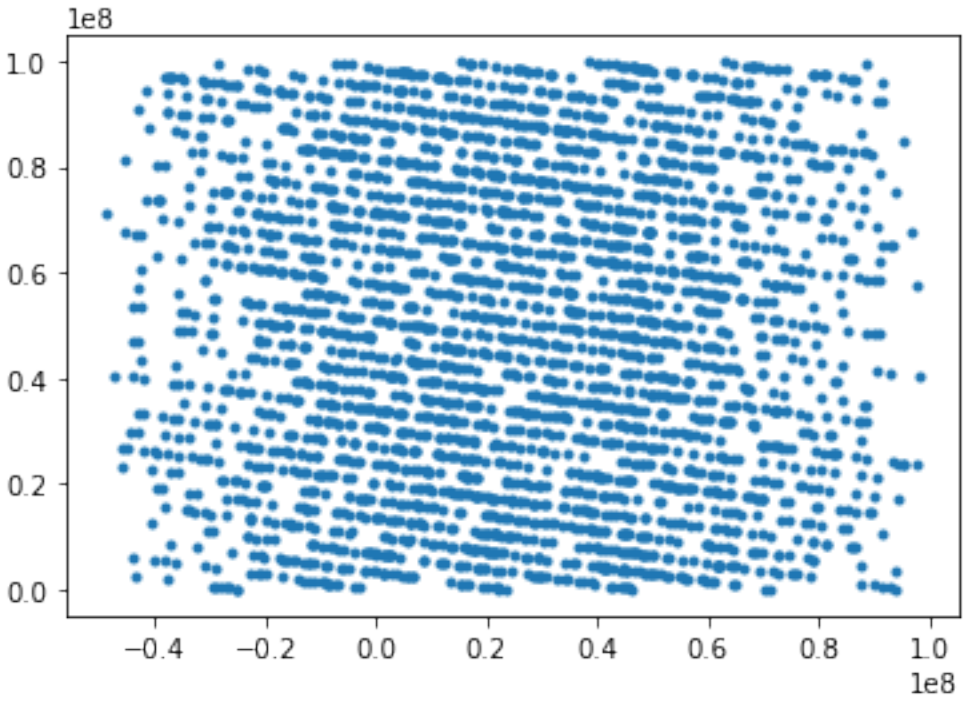
[6]:
```
mylib=ctypes.cdll.LoadLibrary("libc.dylib")
rand=mylib.rand
rand.argtypes=[]
rand.restype=ctypes.c_int

@nb.njit
def get_rands_nb(vals):
    n=len(vals)
    for i in range(n):
        vals[i]=rand()
    return vals

def get_rands(n):
    vec=np.empty(n, dtype='int32')
```

```
    get_rands_nb(vec)
    return vec
```

[7]:
```
n = 30000000
vec=get_rands(n*3)

vv=np.reshape(vec,[n,3])
vmax=np.max(vv,axis=1)

maxval=1e8
vv2=vv[vmax<maxval,:]

xdata = vv2[:,0]
ydata = vv2[:,1]
zdata = vv2[:,2]
```

[8]:
```
plt.plot(xdata-0.5*ydata, zdata, '.')
```

[8]:
```
[<matplotlib.lines.Line2D at 0x12a18a748>]
```



Plotting the randomly generated coordinates from system's PRNG, we observe the same flawed patterns as in the C PRNG.

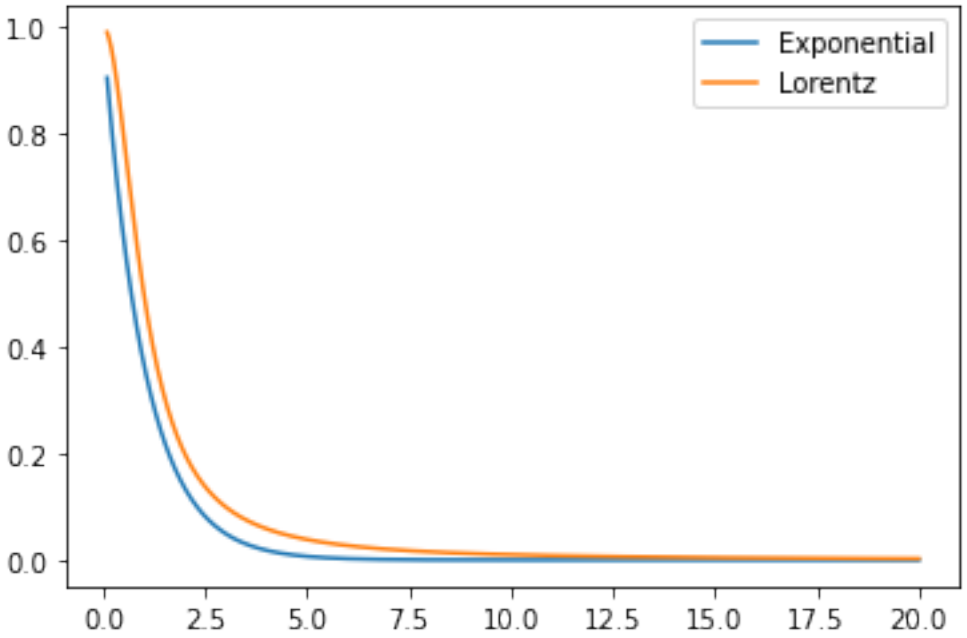
0.2 Question 2

0.2.1 Lorentzian

We know that exponential will always decay faster than lorentzian, so if our lorentzian at 0 is equal to or greater than our exponential at 0 then our lorentzian is always greater than or equal to our exponential. We can choose π for our lorentzian coefficient to satisfy this.

[84]:
```
x = np.linspace(0.1,20,1000)
exp = np.exp(-x)
M = 1
lorentz = M/(1+x**2)

plt.plot(x, exp, label = 'Exponential')
plt.plot(x, lorentz, label = 'Lorentz')
plt.legend()
```

[84]:
```
<matplotlib.legend.Legend at 0x14a4b8208>
```



0.2.2 Gaussian

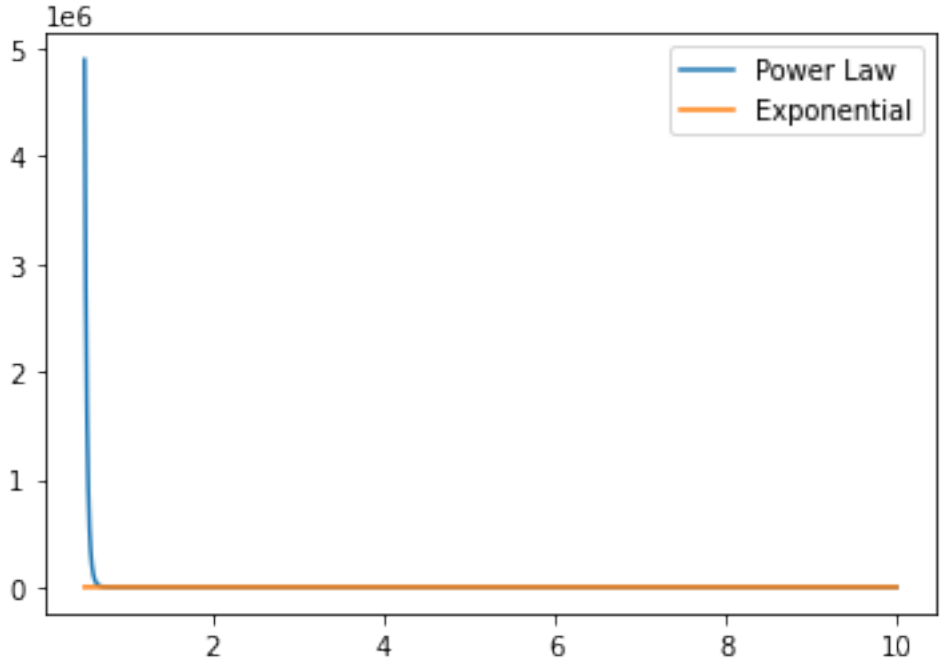
Since Gaussians scale as e^{-x^2}, this will always decay to zero faster than e^{-x} so we cannot use rejection method.

0.2.3 Power Law

[396]:
```
a = 1
x = np.linspace(0.5, 10, 100000)
xmin = np.min(x)
exp = np.exp(-x)
M = 5

alist = np.arange(0.1, 20, 0.1)
for i in range(len(alist)):
    a = alist[i]
    powlaw = M  * (x)**(-a)
    if np.all(powlaw >= exp):
        print(a)
plotpow = M*x**(-a)
plt.plot(x, plotpow, label = 'Power Law')
plt.plot(x, exp, label = 'Exponential')
plt.legend()
plt.show()
```

```
0.1
0.2
0.30000000000000004
0.4
0.5
0.6
0.7000000000000001
0.8
0.9
1.0
1.1
1.2000000000000002
1.3000000000000003
1.4000000000000001
1.5000000000000002
1.6
1.7000000000000002
1.8000000000000003
1.9000000000000001
2.0
2.1
2.2
2.3000000000000003
2.4000000000000004
2.5000000000000004
2.6
2.7
2.8000000000000003
```

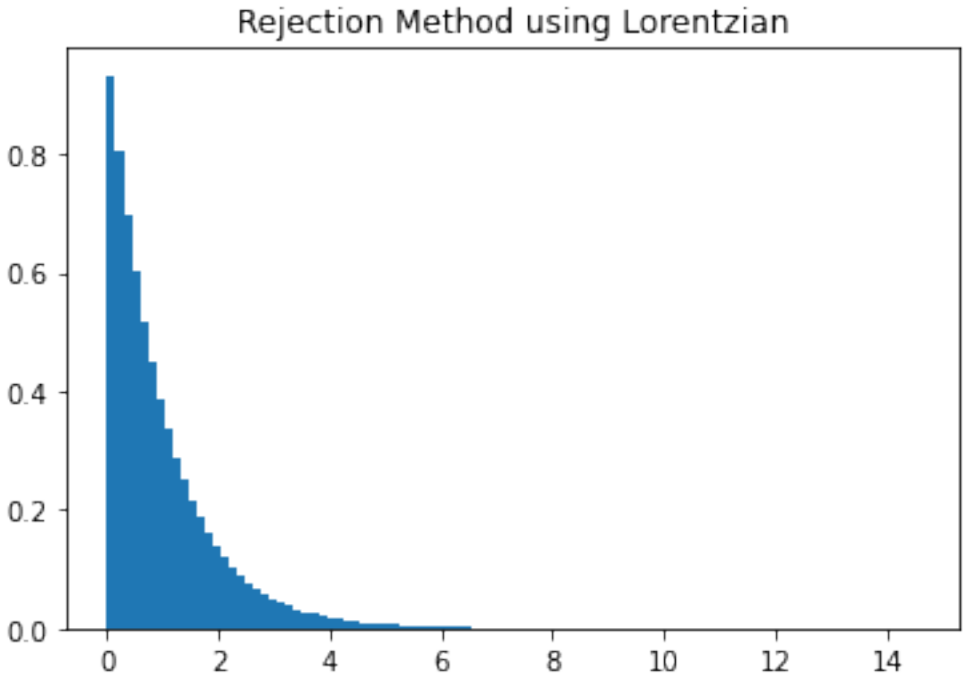
```
2.9000000000000004
3.0000000000000004
3.1
3.2
3.3000000000000003
3.4000000000000004
3.5000000000000004
3.6
3.7
3.8000000000000003
3.9000000000000004
4.0
```



Here we can see that there is some M coefficient large enough such that the powerlaw distribution is always larger than exponential. So if we choose good M and α terms, we can utilise rejection method.

[394]:
```
n = 10000000
M = 1
def exp(x):
    return np.exp(-x)

def lorentz(x):
    return 1/(1+x**2)
```

```
random = np.random.rand(n)
cdfinv = np.tan(np.pi*(random - 0.5))

newcdfinv = cdfinv[cdfinv > 0]
exp = exp(newcdfinv)
lorentz = lorentz(newcdfinv)

y = exp/lorentz
u = np.random.rand(len(y))
accept = u < y
ycut = newcdfinv[accept]

plt.hist(ycut, bins = 100, density = True)
plt.title('Rejection Method using Lorentzian')
plt.show()
print('Method Accepts', len(ycut)/len(y), 'of samples.')
```

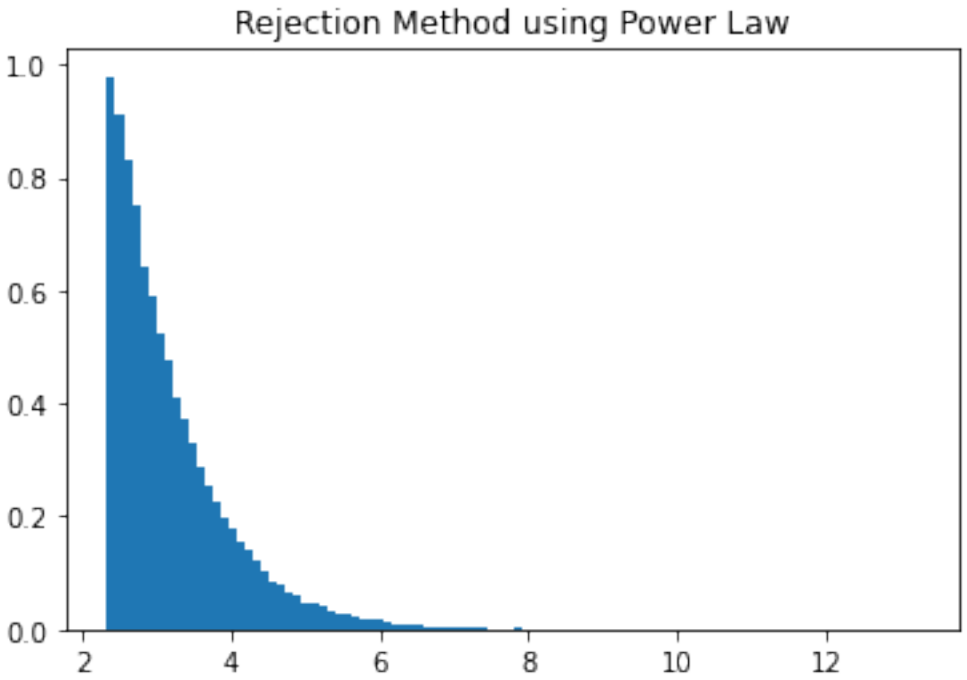



```
Method Accepts 0.6365483216794692 of samples.
```

[395]:
```
n = 1000000
random = np.random.rand(n)
M = 5
a = 2
xmin = np.min(random )
powlaw = M * (random)**(-a)
```

```
cdfinv = np.log(M/random)/np.log(a)

pospowlaw = cdfinv
exp = np.exp(-pospowlaw)
powlaw2 = M * (pospowlaw)**(-a)


y = exp/powlaw2
u = np.random.rand(len(y))
ycut = pospowlaw[u < y]
print(ycut.min())

plt.hist(ycut, bins = 100, density = True)
plt.title('Rejection Method using Power Law')
plt.show()
print('Method Accepts', len(ycut)/len(y), 'of samples.')
```

```
2.3220104552138996
```

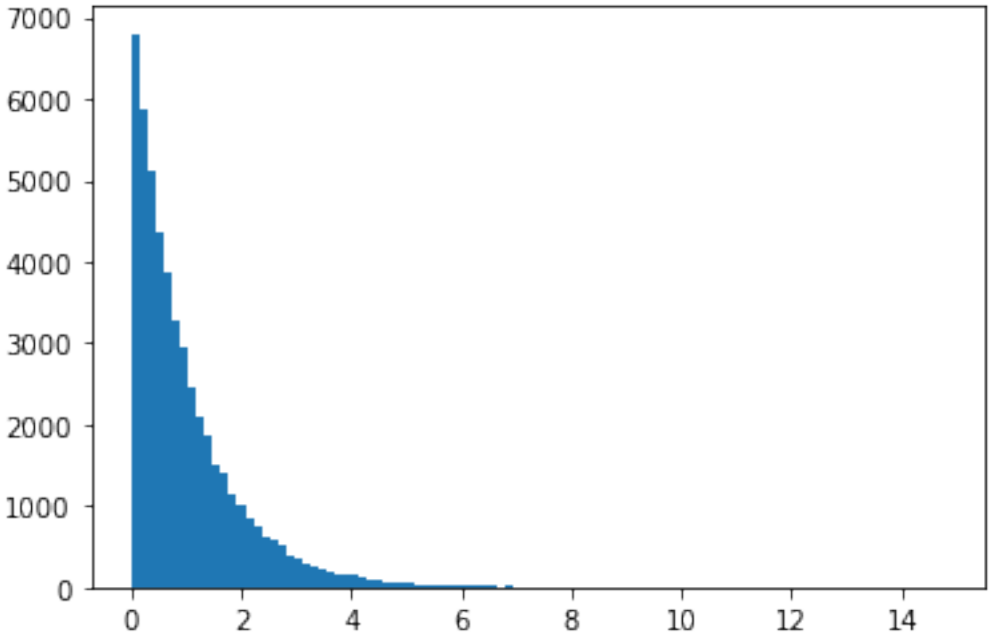



```
Method Accepts 0.070977 of samples.
```

We notice that the acceptance rate of samples of the rejection method using the power law distribution is extremely low, but that makes sense since our plot of the Power Law and Exponential distribution shows that in order for the Power Law distribution to be above the Exponential for all time, it needs to start very high above the exponential which leads to significant samples being rejection since theres such a large gap.

0.3 Problem 3

[398]:
```
n = 100000
u = np.random.rand(n)
v = np.random.rand(n)

ulim = np.sqrt(np.exp(-(v/u)))
ucut = u[u<ulim]
vcut = v[u<ulim]
y = vcut/ucut

plt.hist(y, bins = 100)
plt.show()
print('Method accepts',len(ucut)/len(u), 'of samples.')
```




```
Method accepts 0.49975 of samples.
```

[ ]:

[ ]: