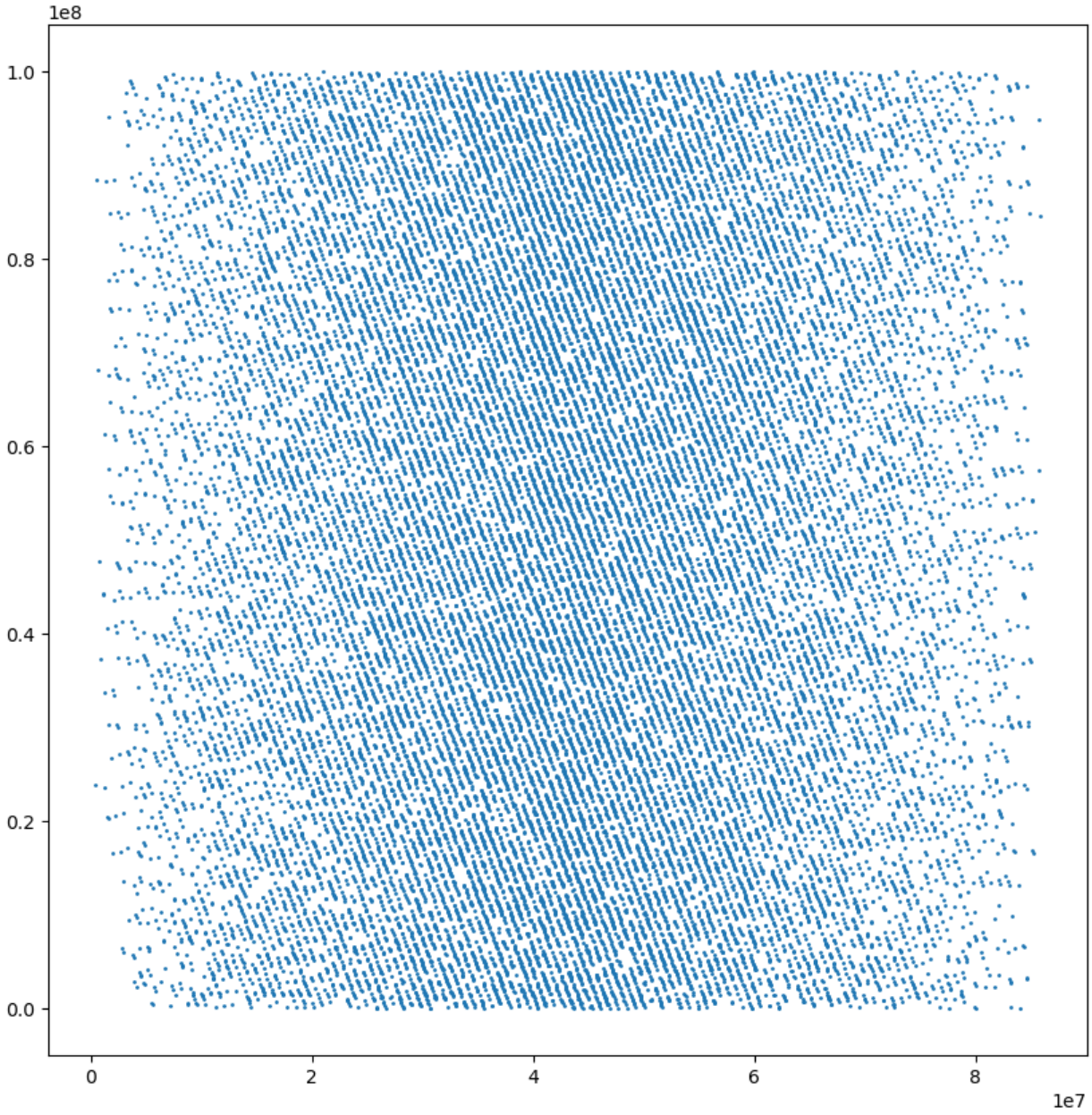
1)

In [120]:
```
# Given code to generate the random triples

import numpy as np
import ctypes
import numba as nb
import time
from matplotlib import pyplot as plt

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

n=300000000
vec=get_rands(n*3)
#vv=vec&(2**16-1)

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

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

f=open('rand_points.txt','w')
for i in range(vv2.shape[0]):
    myline=repr(vv2[i,0])+' '+repr(vv2[i,1])+' '+ repr(vv2[i,2])+'\n'
    f.write(myline)
f.close()
```

In [147]:
```
from scipy.optimize import curve_fit
data=np.loadtxt('rand_points.txt')
print(len(data))

def plane(data,a,b):
    x=data[:,0]
    y=data[:,1]
    return(a*x+b*y)

popt,pcov=curve_fit(plane,data,data[:,2])
print(popt)

a=popt[0]
b=popt[1]

plt.figure(figsize=(10,10))
plt.scatter(plane(data,a,b),data[:,2],s=1)
```

```
30238
[0.42798974 0.43431301]
```

Out[147]: <matplotlib.collections.PathCollection at 0x2c7cf1a30>



Clearly the points lie along some rather small number of planes.

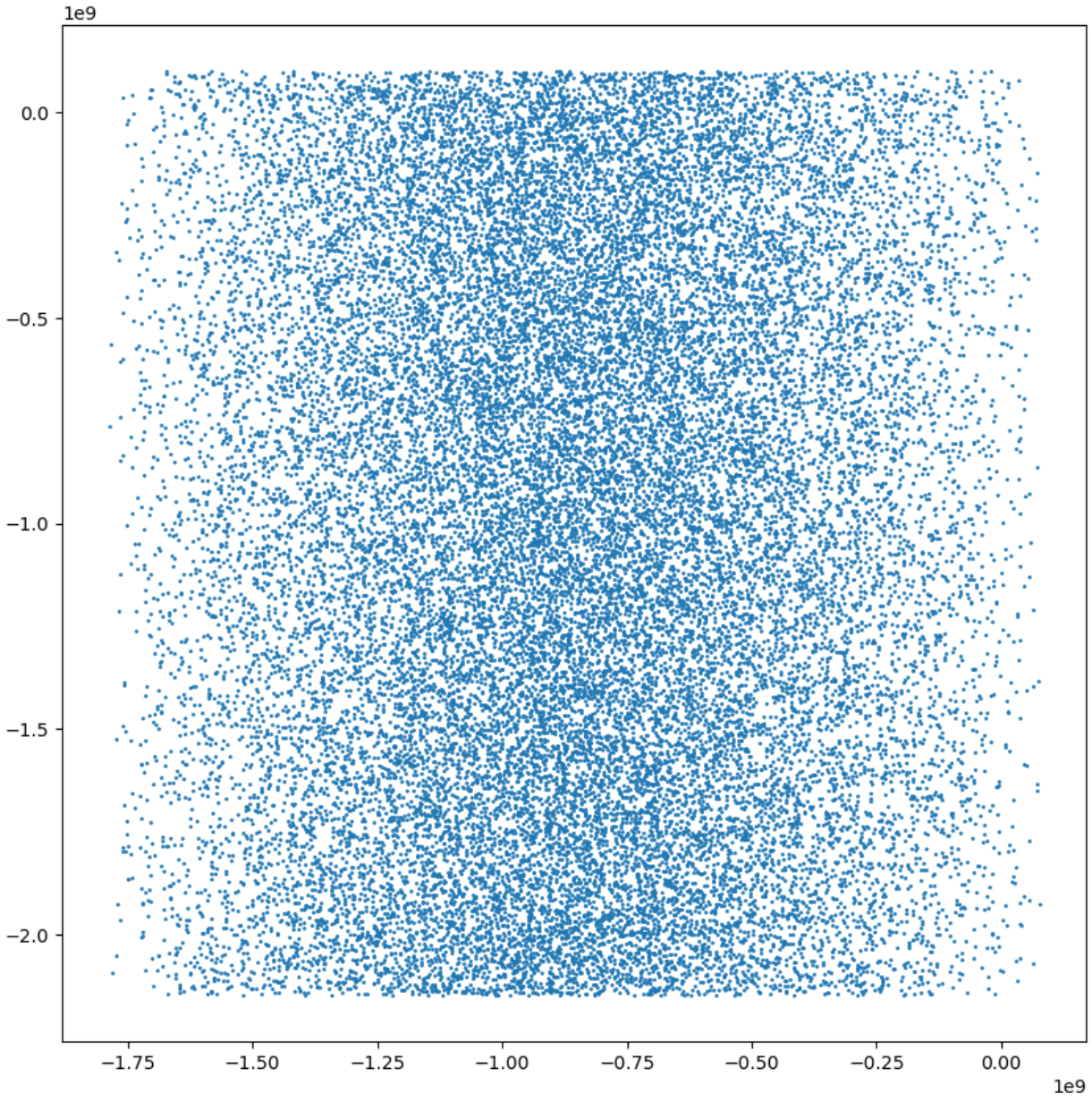
In [156]:
```
# Generate random triples with python now

import random

n=10000000
vec=np.empty(n*3,dtype='int32')
for i in range(n*3):
    vec[i]=random.randint(0,sys.maxsize)
    if(i%10000==0):
        print('Progress: {}%'.format(np.round((i/(n*3))*100,1)),end="\r",fl

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

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

```
Progress: 100.0%
```

In [157]:
```
data=vv2[:30000]

def plane(data,a,b):
    x=data[:,0]
    y=data[:,1]
    return(a*x+b*y)

popt,pcov=curve_fit(plane,data,data[:,2])
print(popt)

a=popt[0]
b=popt[1]

plt.figure(figsize=(10,10))
plt.scatter(plane(data,a,b),data[:,2],s=1)
```

```
[0.4129878 0.4208622]
```

Out[157]:
```
<matplotlib.collections.PathCollection at 0x16a1eda00>
```



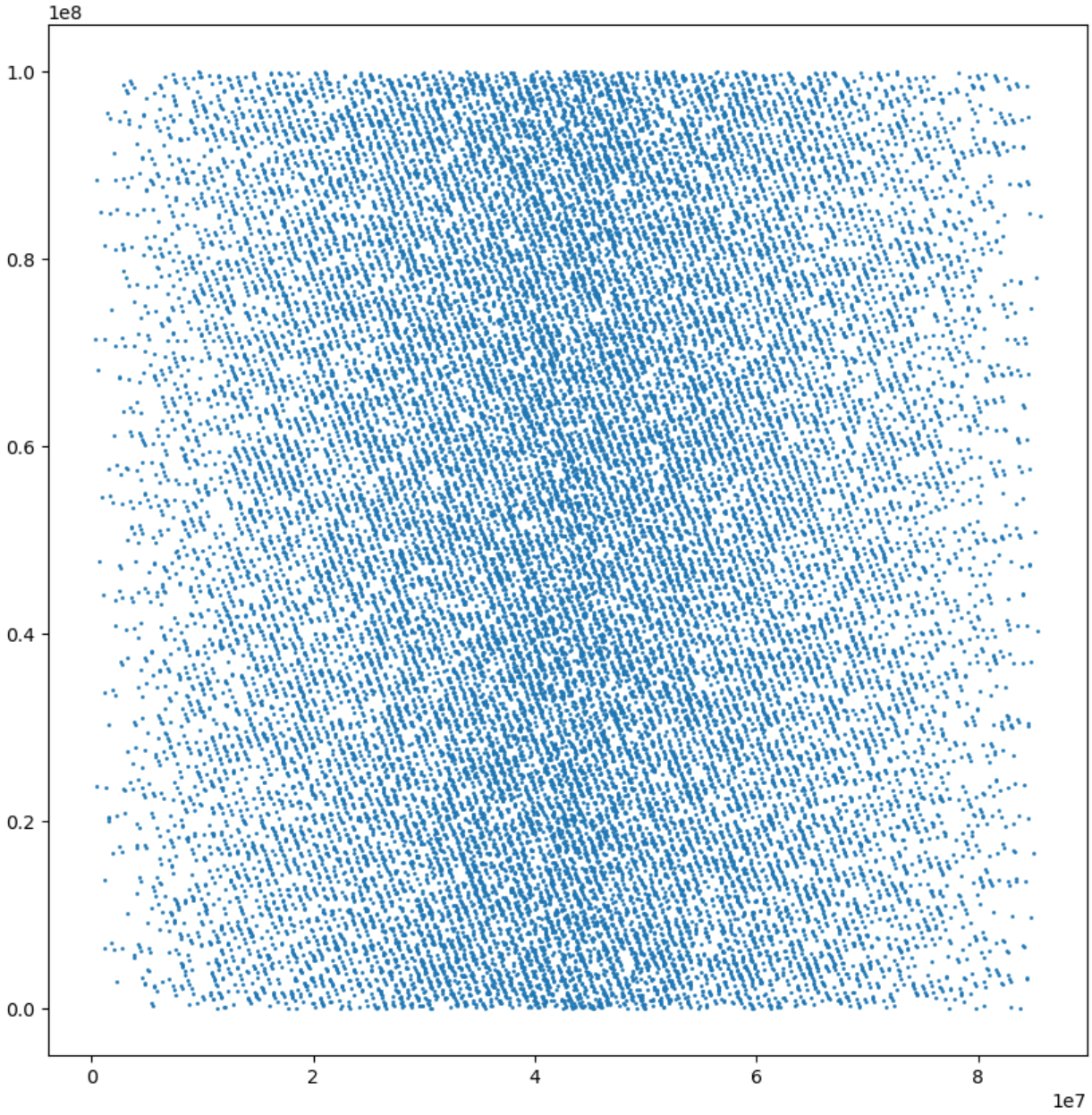
Using python's random int generator, there is no readily apparent correlation between the numbers unlike C's rand().

In [164]:
```
mylib=ctypes.cdll.LoadLibrary("libSystem.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

n=300000000
vec=get_rands(n*3)
#vv=vec&(2**16-1)

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

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

In [165]:
```
data=vv2

def plane(data,a,b):
    x=data[:,0]
    y=data[:,1]
    return(a*x+b*y)

popt,pcov=curve_fit(plane,data,data[:,2])
print(popt)

a=popt[0]
b=popt[1]

plt.figure(figsize=(10,10))
plt.scatter(plane(data,a,b),data[:,2],s=1)
```

```
[0.42794588 0.43164077]
```

Out[165]:
```
<matplotlib.collections.PathCollection at 0x2b37473a0>
```

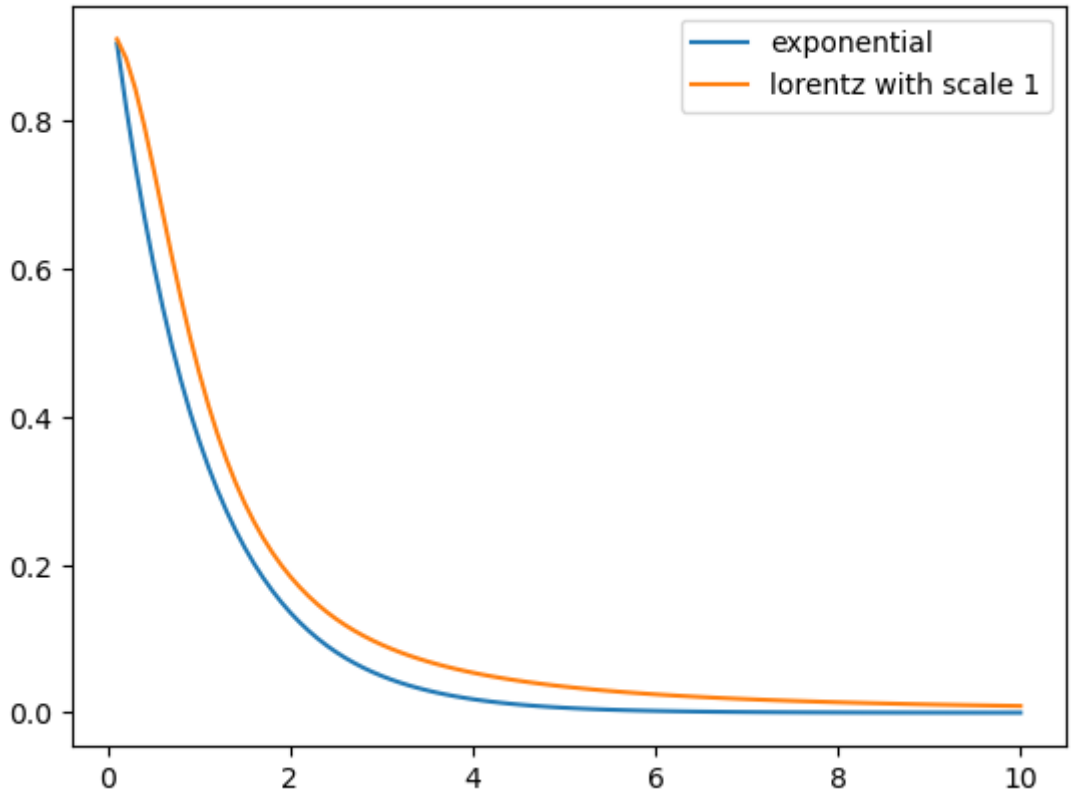


Using my local random int generator, seems to have the same problem as C but maybe to a lesser extent.

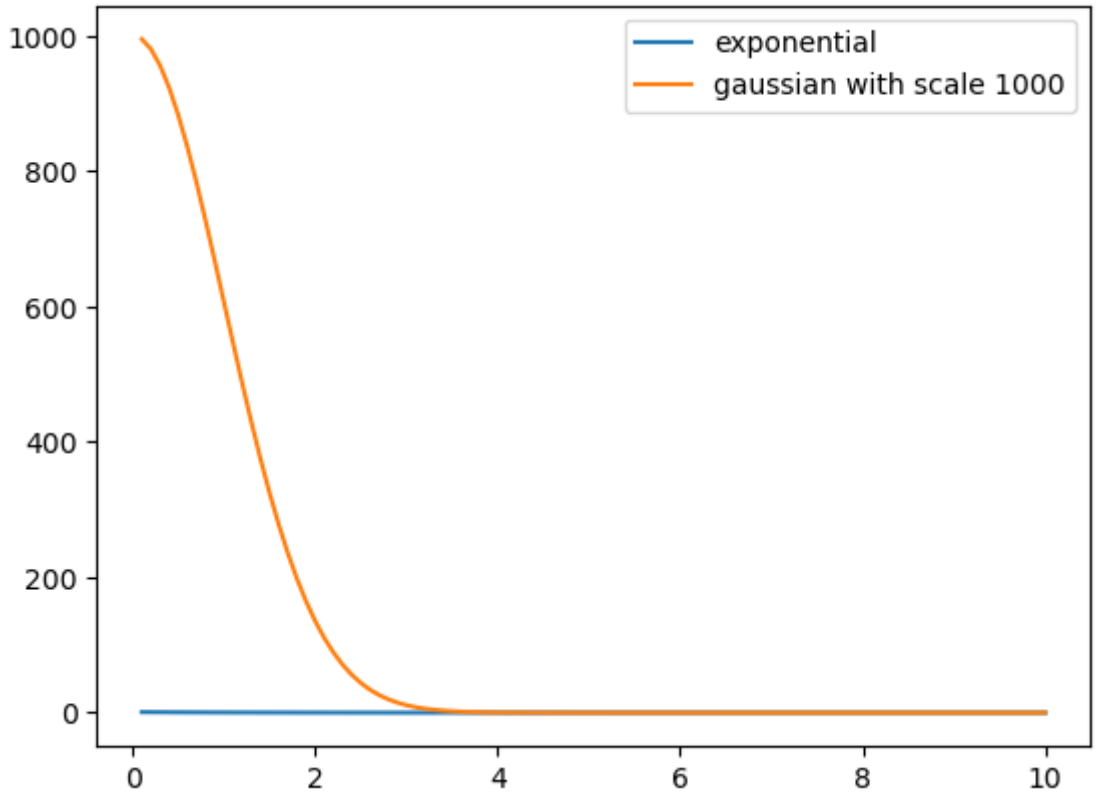
2)

First let us check that our bounding distributions are greater than the exponential in our chosen domain. Also scale each distribution to get as close to exponential as possible while still being larger on the domain.

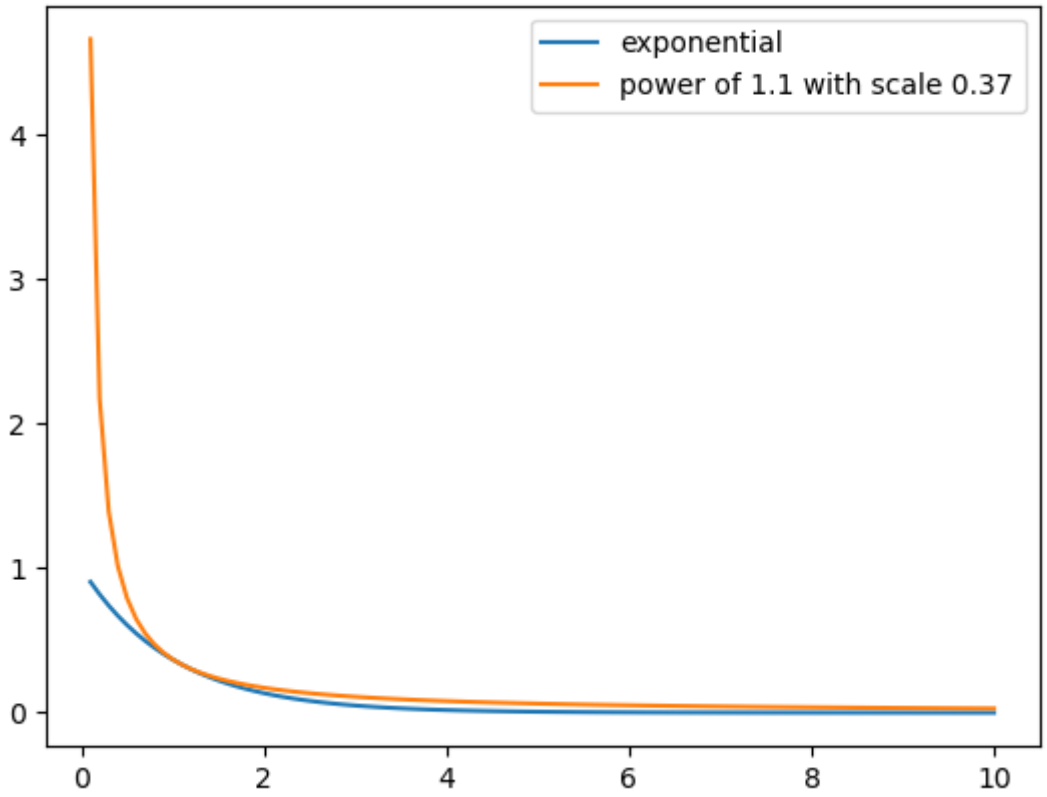
In [244]:
```
def lorentz(x):
    return (1/(1+x**2))

def gauss(x):
    return np.exp(-(x**2)/2)

def power(x,a):
    return (x**(-a))

# Avoiding x=0 to not divide by 0 in power law
x=np.linspace(0.1,10,100)

plt.plot(x,np.exp(-x),label='exponential')
plt.plot(x,0.92*lorentz(x),label='lorentz with scale 1')
plt.legend()
plt.show()
print('Always greater:',np.all(np.exp(-x)<=0.92*lorentz(x)))

plt.plot(x,np.exp(-x),label='exponential')
plt.plot(x,1000*gauss(x),label='gaussian with scale 1000')
plt.legend()
plt.show()
print('Always greater:',np.all(np.exp(-x)<=1000*gauss(x)))

plt.plot(x,np.exp(-x),label='exponential')
plt.plot(x,0.37*power(x,1.1),label='power of 1.1 with scale 0.37')
plt.legend()
plt.show()
print('Always greater:',np.all(np.exp(-x)<=0.37*power(x,1.1)))
```



```
Always greater: True
```




```
Always greater: False
```




```
Always greater: True
```

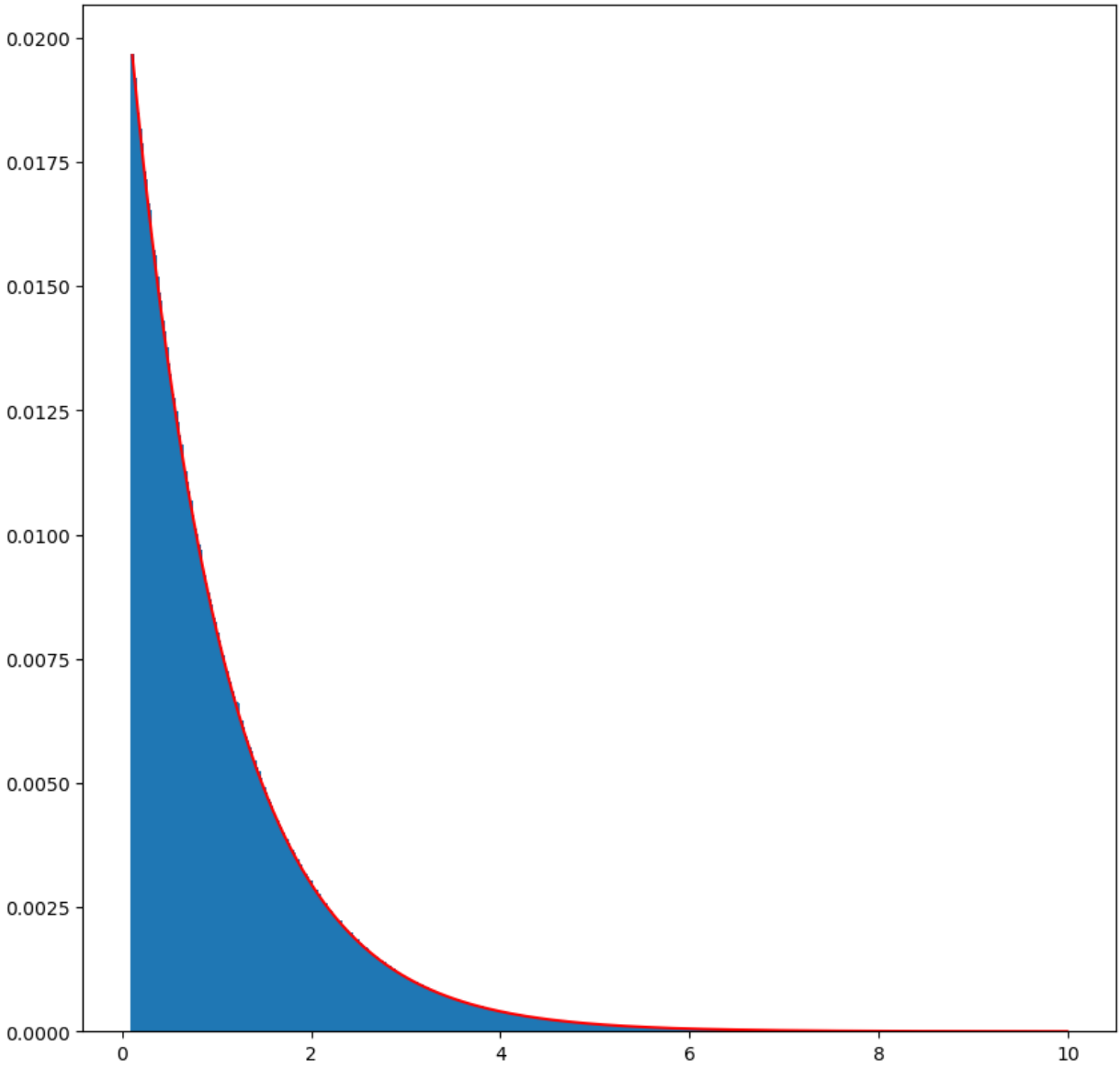
In our chosen range of 0.1 to 10, we can use lorentz and power bounding distributions with scaling factors 0.92 and 0.37, respectively. A gaussian does not seem to ever be always greater in this range regardless of the scaling.

From class we have that the CDF of lorentz is atan(x)/pi+1/2. So to get the x coordinate it is x=tan(pi(y-1/2)), where y is in $[0, 1)$ chosen from a uniform distribution. We then simply plug in these x coordinates sampled from a lorentz distribution, into our scaled lorentz function, scale each point again with a uniform distribution in $[0, 1)$ and check if it is less than our exponential. If so we accept it.

Similarly for the power law we get x=y^(1/(1-a)).

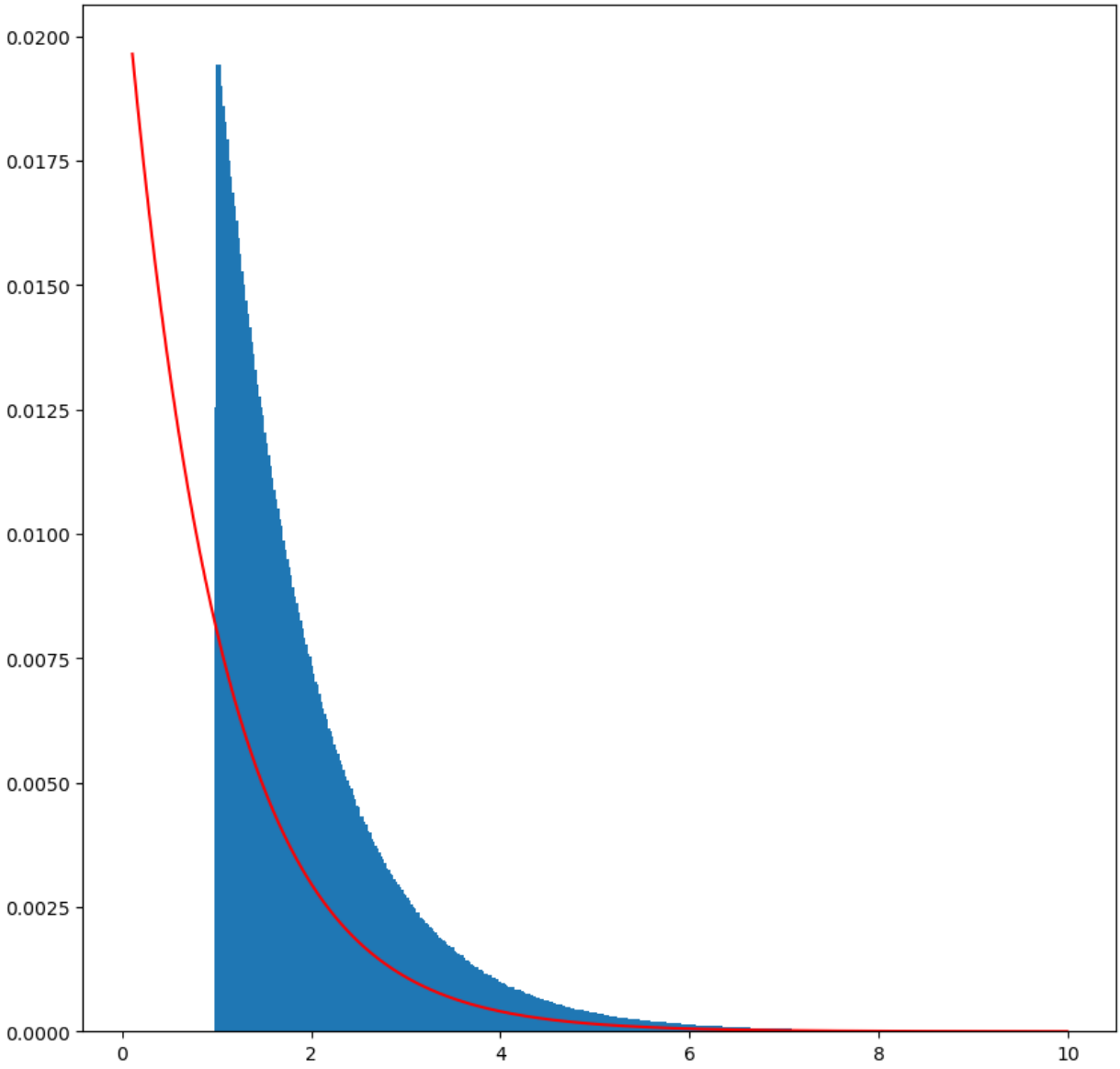
In [238]:
```
# Generate exponential deviate from lorentz bounding

n=10000000
scale=0.92
x=np.tan(np.pi*(np.random.rand(n)-0.5))
y=scale*lorentz(x)*np.random.rand(n)
accept=y<np.exp(-x)
print('accept fraction is',np.mean(accept))

x_use=x[accept]
aa,bb=np.histogram(x_use,np.linspace(0.1,10,500))
b_cent=0.5*(bb[1:]+bb[:-1])
pred=np.exp(-b_cent)
pred=pred/pred.sum()
aa=aa/aa.sum()
plt.figure(figsize=(10,10))
plt.plot(b_cent,pred,'r')
plt.bar(b_cent,aa,0.05)
plt.show()
```

```
/var/folders/0f/xf8275ns0xl9fdqqwqm4c7jr0000gn/T/ipykernel_2111/120805454
8.py:7: RuntimeWarning: overflow encountered in exp
  accept=y<np.exp(-x)

accept fraction is 0.8447894
```


0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
0
2
4
6
8
10

In [265]:
```
# Now do exponential deviate from power law bounding with a=1.1

n=100000000
scale=0.37
a=1.1
x=np.random.rand(n)**(1/(1-a))
y=scale*power(x,a)*np.random.rand(n)
accept=y<np.exp(-x)
print('accept fraction is',np.mean(accept))

x_use=x[accept]
aa,bb=np.histogram(x_use,np.linspace(0.1,10,500))
b_cent=0.5*(bb[1:]+bb[:-1])
pred=np.exp(-b_cent)
pred=pred/pred.sum()
aa=aa/aa.sum()
plt.figure(figsize=(10,10))
plt.plot(b_cent,pred,'r')
plt.bar(b_cent,aa,0.05)
plt.show()
```

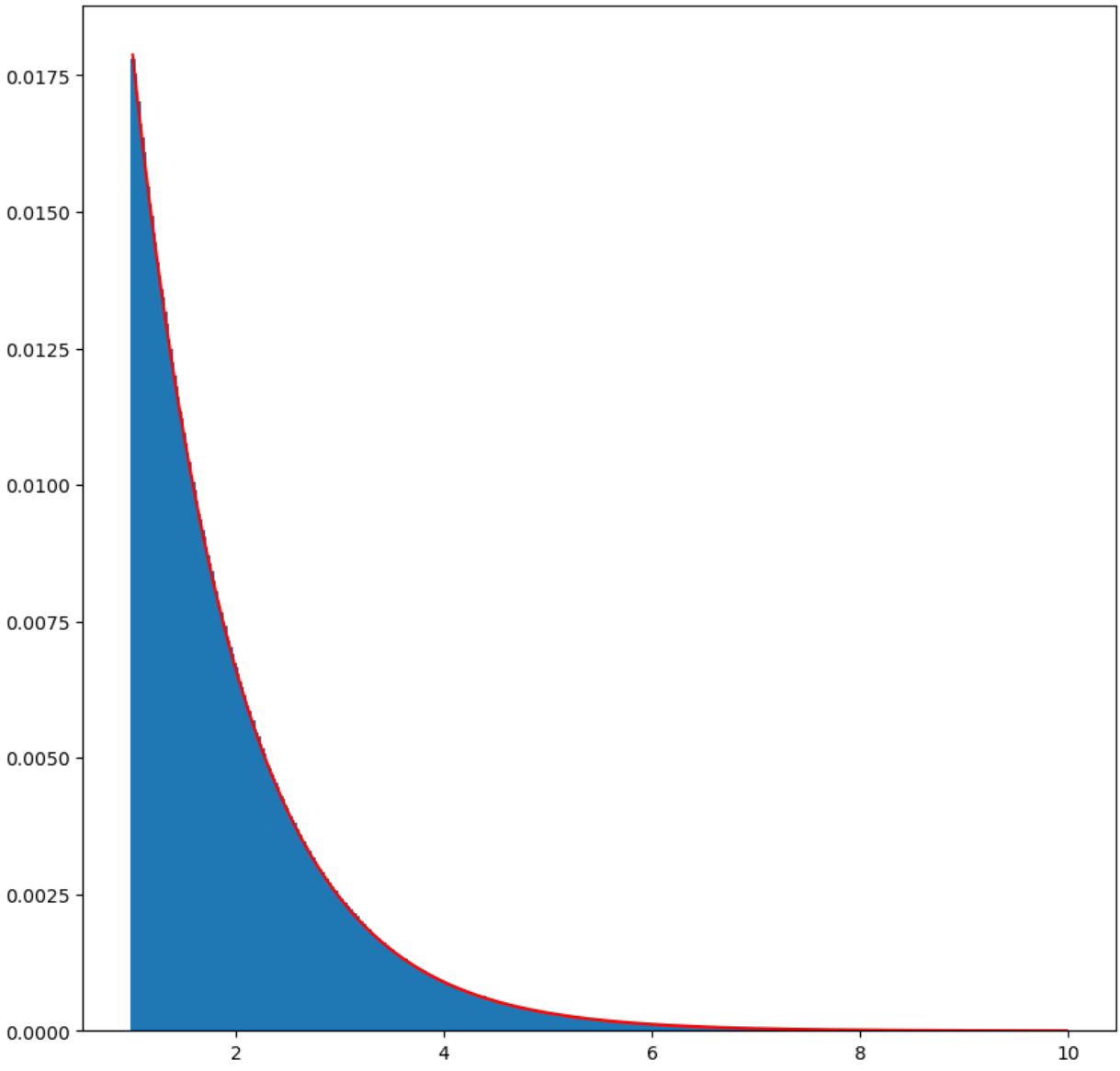
```
accept fraction is 0.09941797
```



Does not seem to work for values below x=1. Restrict to just $[1, 10]$. Happening because inverting the function to find the x values can only give a minimum of x=1.

In [249]:
```
n=100000000
scale=0.37
a=1.1
x=np.random.rand(n)**(1/(1-a))
y=scale*power(x,a)*np.random.rand(n)
accept=y<np.exp(-x)
print('accept fraction is',np.mean(accept))

x_use=x[accept]
aa,bb=np.histogram(x_use,np.linspace(1,10,500))
b_cent=0.5*(bb[1:]+bb[:-1])
pred=np.exp(-b_cent)
pred=pred/pred.sum()
aa=aa/aa.sum()
plt.figure(figsize=(10,10))
plt.plot(b_cent,pred,'r')
plt.bar(b_cent,aa,0.05)
plt.show()
```

```
accept fraction is 0.0994339
```



Now they match.

3)

We have that u is in $[0, 1]$, and we need $0 < u < \sqrt{p(v/u)}$. In our case, $0 < u < \sqrt{(e^{-v/u})}$. Thus let us find the range for v.

$u < e^{-v/(2u)}$

$\ln u = -v/(2u)$

$v = -2u * \ln u$

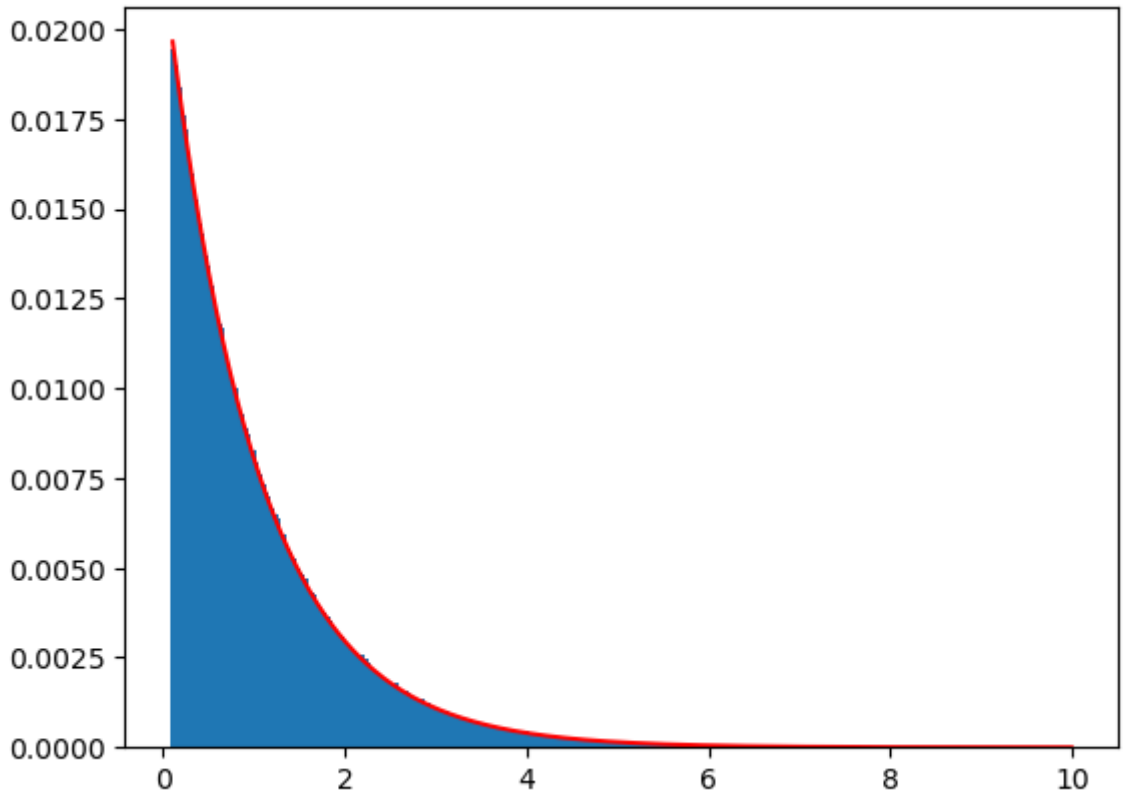
In [276]:
```
u=np.linspace(0,1,2001)
u=u[1:]

v=-2*u*np.log(u)
print('max v is',v.max())
```

```
max v is  0.7357588428385197
```

In [283]:
```
n=1000000
u=np.random.rand(n)
v=np.random.uniform(low=0,high=v.max(),size=n)
r=v/u
accept=u<np.exp(-0.5*r)
print('accept fraction is',np.mean(accept))
r_use=r[accept]

aa,bb=np.histogram(r_use,np.linspace(0.1,10,500))
aa=aa/aa.sum()
b_cent=0.5*(bb[1:]+bb[:-1])
pred=np.exp(-b_cent)
pred=pred/pred.sum()
plt.bar(b_cent,aa,0.05)
plt.plot(b_cent,pred,'r')
```

```
accept fraction is 0.679177
```

Out[283]:
```
[<matplotlib.lines.Line2D at 0x2822d17c0>]
```



Not as efficient as lorentz, but still works, and only needs uniform.