﻿import pandas as pd

import scipy.stats as st

import random

import numpy as np

from scipy.stats import bernoulli

from scipy.integrate import quad

import matplotlib.pyplot as plt

import seaborn as sns

import math

**a**

def p\_x(x, sigma, theta):

p\_x = st.norm.pdf(x, theta, sigma)

return p\_x

def g\_x(x,theta):

g\_x = 0

bottom = np.sin(x-theta)\*\*2

top = np.sin(5 \* (x - theta))

if bottom is 0:

g\_x = 0

else:

g\_x = (top\*\*2) / (25\*bottom)

return g\_x

def g\_zero(x,theta):

g\_zero = 1 - g\_x(x,theta)

return g\_zero

def a1a(x):

left = g\_zero(x,0)

right = p\_x(x, 4, 0)

result = left \* right

return result

#def ber(x, theta):

#f\_x = 0

#f\_x = f\_x(x,theta)

#bernoulli = bernoulli(f\_x)

#return ber

def integrand(a,b,n):

temp, size\_spacing = np.linspace(start = a, stop = b, num = n, retstep = True)

result = 0

for i in temp:

result += a1a(i)\*size\_spacing

return result

﻿a1a\_answer = round(integrand(-20, 20,10000),2)

print("1a: The answer is %s"%a1a\_answer)

﻿1a: The answer is 0.8

**B**

﻿# 1 b

x\_seq = np.random.normal(0, 4, 1)

u = np.random.uniform(0,1)

def rejection\_sampling(x,iter=10000):

sample = []

x\_sample = []

accept = 0

reject = 0

while accept < iter:

x\_temp = np.random.normal(0, 4, 1)

u = np.random.uniform(0,1)

if u < g\_x(x\_temp,0):

accept +=1

sample.append(u)

x\_sample.append(x\_temp)

else:

reject +=1

return accept, reject, sample, x\_sample

accept, reject, sample, x\_sample = rejection\_sampling(x\_seq,iter=10000)

fraction = round(accept/(reject+accept),4)

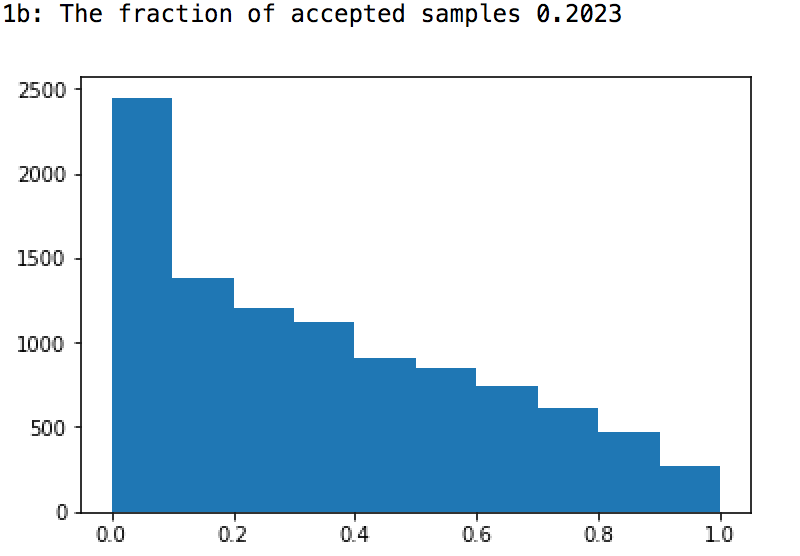
print("1b: The fraction of accepted samples %s"%fraction)

plt.hist(sample)

plt.show()

plt.savefig("b")

fraction is 0.2023



**C**

﻿# 1 c

# after cancallation the self-normalized importance sampler is 1/k \* sum(f(xi))

# f'(x) is 1 - g\_x()

def self\_normalized(x,iter=1000):

sample = []

x\_sample = []

accept = 0

reject = 0

while accept < iter:

x\_temp = np.random.normal(0, 4, 1)

u = np.random.uniform(0,1)

if u < (1-g\_x(x\_temp,0)):

accept +=1

sample.append(u)

x\_sample.append(x\_temp)

else:

reject +=1

return accept, reject, sample, x\_sample

accept, reject, sample, x\_sample = rejection\_sampling(x\_seq,iter=10000)

fraction = round(accept/(reject+accept),3)

print("1c: The fraction of photons absorbed is %s"%fraction)

﻿1c: The fraction of photons absorbed is 0.198

**D**

﻿#1 D

def theta\_function(theta):

bottom = 10\*math.pi\*(1+(theta/10)\*\*2)

t\_f = 1/bottom

return t\_f

def a1d\_func(theta):

current\_theta = theta\_function(theta)

left = p\_x(1.7, 4, theta) \* current\_theta

right = g\_x(1.7,theta)

result = left \* right

return result

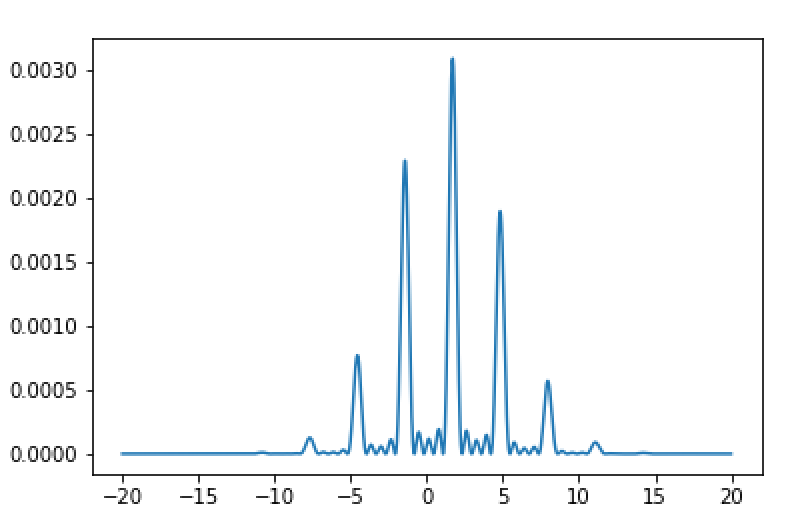
theta\_seq = np.arange(-20,20,0.004)

a1d\_y = a1d\_func(theta\_seq)

plt.plot(theta\_seq,a1d\_y)

plt.show()

plt.savefig("d")



E

﻿def sample\_func(theta):

return theta\_function(theta) \* g\_x(1.7,theta)

def metropolis(iters, sd, init):

samples = np.zeros(iters)

n\_accept = 0

theta = init

for i in range(iters):

prop = np.random.normal(theta, sd)

px = st.norm.pdf(theta, prop, sd) \* sample\_func(prop)

qx = st.norm.pdf(prop, theta, sd) \* sample\_func(theta)

# compute acceptance probability

p\_accept = np.minimum(1, px/qx)

if np.random.rand() < p\_accept:

# accept

theta = prop

n\_accept += 1

samples[i] = theta

return samples, 1. \* n\_accept / iters

sample\_e, fraction = metropolis(10000, 1,0)

plt.hist(sample\_e,bins=1000)

plt.title("initial theta = 0, sigma = 1")

plt.show()

plt.savefig("e1")

sample\_e, fraction = metropolis(10000,2,0)

plt.hist(sample\_e,bins=1000)

plt.title("initial theta = 0, sigma = 2")

plt.show()

plt.savefig("e2")

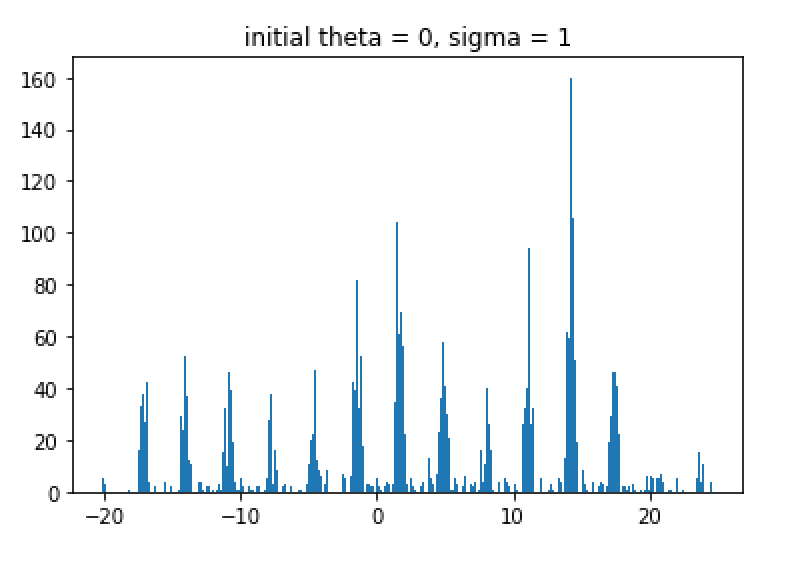
sample\_e, fraction = metropolis(10000,4,0)

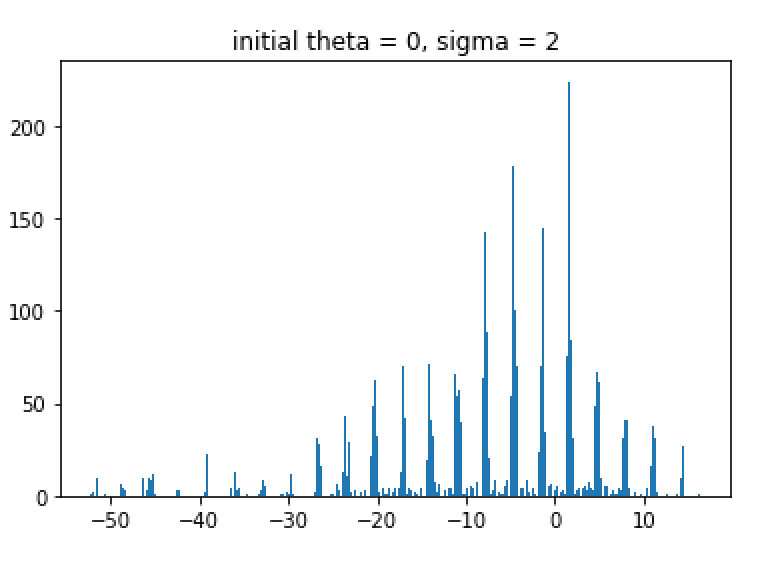
plt.hist(sample\_e,bins=1000)

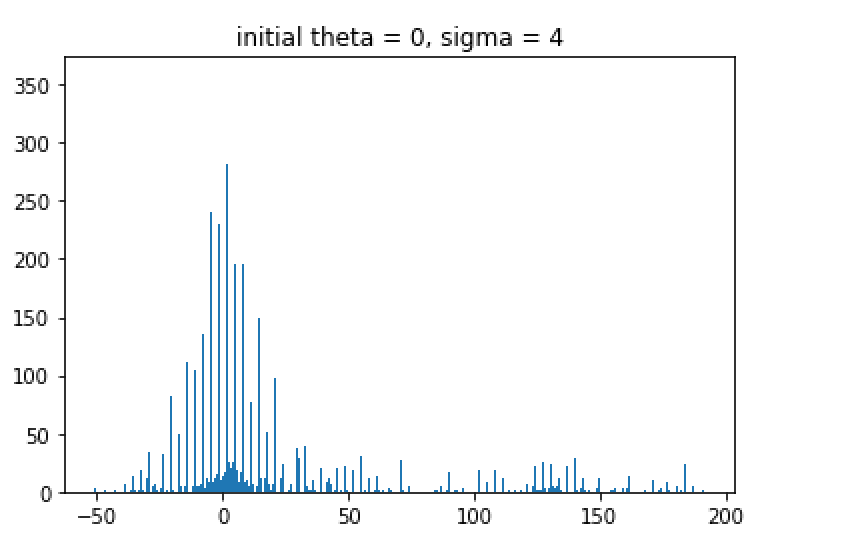
plt.title("initial theta = 0, sigma = 4")

plt.show()

plt.savefig("e3")







**F**

﻿#1 f

theta\_seq = np.arange(-3,3,0.001)

def posterior\_probability(init,iters):

samples = np.zeros(iters)

n\_accept = 0

reject = 0

for i in range(iters):

theta = init[i]

p = p\_x(1.7,theta, 4) \* theta\_function(theta) \* g\_x(1.7,theta)

# compute acceptance probability

if -3 < p < 3:

# accept

n\_accept += 1

else:

reject +=1

samples[i] = p

return samples, 1. \* n\_accept / iters

samples, fraction = posterior\_probability(theta\_seq,6000)

af = round(fraction,2)

print("1f: The fraction of calibrated to within 3 degrees around zero is %s"%af)

﻿1f: The fraction of calibrated to within 3 degrees around zero is 0.5