NE 255 - Homework 6

University of California, Berkeley
Department of Nuclear Engineering

Daniel Hellfeld
dhellfeld@berkeley.edu

Problem 1
Using the direct inversion of CDF sampling method, derive sampling algorithms for

(a) The neutron direction in 3D if the neutron source is isotropic.

...

(b) The distance to the next collision in the direction of neutron motion if the neutron is in the center of the spherical volume that consists of three concentric layers with radii R_1, R_2, and R_3, each made of different materials with total cross sections Σ_{t1}, Σ_{t2}, and Σ_{t3}, respectively.

...

(c) The type of collision if it is assumed that the neutron can have both elastic and in-elastic scattering, and can be absorbed in fission or (n,γ) capture interactions. Assume monoenergetic neutron transport.

...

Problem 2

Use a rejection Monte Carlo method to evaluate $\pi = 3.14159$:

- From $\pi = 4\int_0^1 \sqrt{1-x^2}dx$
- From $\pi = 4\int_0^1 \frac{1}{1+x^2}dx$
- Assuming the $\pi = 3.14159$ is exact, calculate the relative error for 10, 100, 1,000, and 10,000 samples.
- What do you notice about the behavior of error as a function of the number of trials?

⇒ A code was written in Python to perform the rejection sampling and calculate π using the two functions above. The relative error is also calculated for the different number of samples for each function. The results were then also fit to a function of the form

$$f(x) = \frac{a}{\sqrt{N}}$$

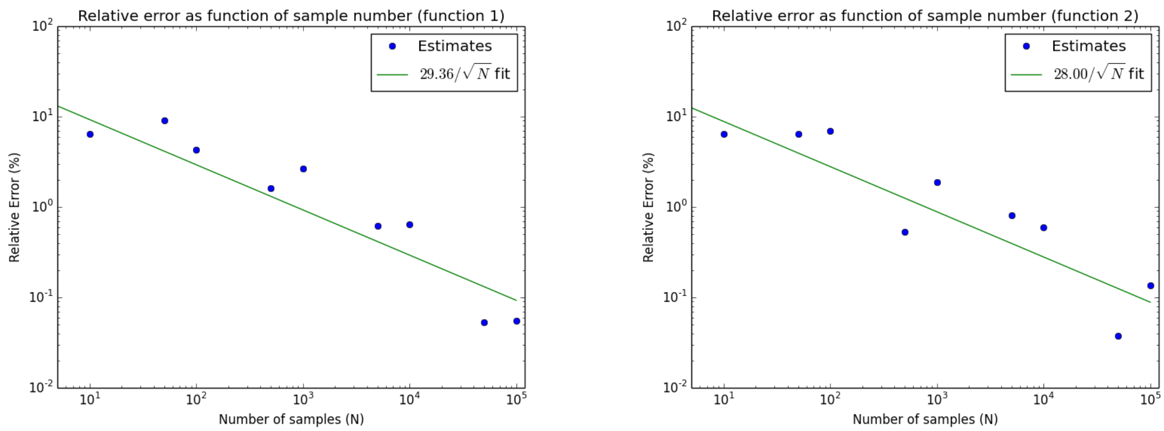
and plotted together. This is because we expect the relative error to decrease with the number of samples as $1/\sqrt{N}$. The code is shown below and the relative error results (for both functions) are tabulated in Table 1 and plotted in Fig. 1 with the fitted function.

```
# NE 255 - Homework 6 - Problem 2
# D. Hellfeld
# 12/1/16

# Imports
import numpy as np
import matplotlib.pyplot as plt
import random as random
from scipy.optimize import curve_fit

# Actual PDF(s)
def p1(x):
    return 4. * np.sqrt(1. - x**2)

def p2(x):
    return 4. * (1. / (1. + x**2))

# Simple enclosing PDF (uniform), defined on [a,b]
def g(x,a,b,const):
    if (x >= a and x <= b):
        return const
    else:
        return 0.

def G(xi,a,b,const):
    return (xi * (b-a) + a)

# Define variables
N           = np.asarray([10,50,100,500,1000,5000,10000,50000,100000])
a           = 0.
b           = 1.
const       = 4.2
pi_exact    = 3.14159
pi_estimate_1 = (np.zeros_like(N)).astype(float)
pi_estimate_2 = (np.zeros_like(N)).astype(float)
rel_error_1  = (np.zeros_like(N)).astype(float)
rel_error_2  = (np.zeros_like(N)).astype(float)

# Loop through different sample values
itr = 0
for n in N:

    # Sample
    accept_1 = 0.
    accept_2 = 0.
    for i in range(int(n)):

        xi  = random.random()
        eta = random.random()
        x   = G(xi, a, b, const)

        if (eta * g(x, a, b, const) < p1(x)):
```

```
            accept_1 += 1.
        if (eta * g(x, a, b, const) < p2(x)):
            accept_2 += 1.

    # Calculate results
    pi_estimate_1[itr] = (const*(b-a)) * (accept_1/n)
    rel_error_1[itr] = (np.fabs(pi_estimate_1[itr] - pi_exact)/pi_exact) * 100.
    pi_estimate_2[itr] = (const*(b-a)) * (accept_2/n)
    rel_error_2[itr] = (np.fabs(pi_estimate_2[itr] - pi_exact)/pi_exact) * 100.

    # Increment loopnum
    itr += 1

# Print results
print "Number of samples = ", N
print "Pi estimates (function 1) = ", pi_estimate_1
print "Relative errors (function 1) = ", rel_error_1, "%"
print "Pi estimates (function 2) = ", pi_estimate_2
print "Relative errors (function 2) = ", rel_error_2, "%"

# Define fit function
def fit(x,a):
    return a / np.sqrt(x)

# Fit to data
popt_1, pcov_1 = curve_fit(fit, N, rel_error_1)
popt_2, pcov_2 = curve_fit(fit, N, rel_error_2)

# Plot results with fit (function 1)
x = np.linspace(1,100000,100000)
plt.figure()
plt.plot(N, rel_error_1, linestyle='none', marker='o',label='Estimates')
plt.plot(x, fit(x,popt_1), label='$%.2f/\sqrt{N}$ fit' %popt_1)
plt.xscale('log'); plt.yscale('log')
plt.xlabel('Number of samples (N)'); plt.ylabel('Relative Error (%)')
plt.title('Relative error as function of sample number (function 1)')
plt.xlim(5,1.2e5); plt.ylim(1e-2,1e2)
plt.legend(numpoints=1)

# Plot results with fit (function 2)
plt.figure()
plt.plot(N, rel_error_2, linestyle='none', marker='o',label='Estimates')
plt.plot(x, fit(x,popt_2), label='$%.2f/\sqrt{N}$ fit' %popt_2)
plt.xscale('log'); plt.yscale('log')
plt.xlabel('Number of samples (N)'); plt.ylabel('Relative Error (%)')
plt.title('Relative error as function of sample number (function 2)')
plt.xlim(5,1.2e5); plt.ylim(1e-2,1e2)
plt.legend(numpoints=1)

# Render plots
plt.show()
```

Number of Samples (N)	Function 1: $4\sqrt{1-x^2}$		Function 2: $4/(1+x^2)$	
	Estimated value	Relative Error (%)	Estimated value	Relative Error (%)
10	2.9400	6.4168	2.9400	6.4168
50	2.8560	9.0906	2.9400	6.4168
100	3.2760	4.2784	3.3600	6.9522
500	3.0912	1.6039	3.1584	0.5350
1000	3.0576	2.6734	3.0828	1.8713
5000	3.1609	0.6152	3.1668	0.8024
10000	3.1214	0.6413	3.1227	0.6012
50000	3.1399	0.0531	3.1427	0.0377
100000	3.1433	0.0551	3.1458	0.1353

Table 1. Rejection sampling estimate and relative errors for $\pi = 3.14159$ using two different functions.



(a) (b)

Fig. 1. (a) Relative errors as function of sample number for function 1, plotted with fit line. (b) Relative errors as function of sample number for function 2, plotted with fit line.

We see that the relative errors generally decrease as the sample number is increased. Of course there is some variation because we are using random numbers. However, in both cases, we observe that the general trend of the relative errors seems to follow the expected $\propto N^{-1/2}$ behavior.