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.

$\Rightarrow$ First, we note that the neutron direction is determined using the azimuthal ($\phi \in [0, 2\pi)$) and polar angles ($\theta \in [0, \pi]$). If the neutron source is isotropic, one might think to just simple randomly sample θ on the interval $[0, \pi]$ and ϕ on the interval $[0, 2\pi)$. However this will lead to clustering near the poles (small θ). This is because of the $\sin\theta$ term in the the differential solid angle element $d\Omega = \sin\theta d\theta d\phi$.

Isotropic means there is an equal probability to be emitted in any direction in 4π. The total probability should of course be 1 (the neutron is emitted in some direction) and we assume directions are continuous, therefore the probability density function (PDF) can be determined by

$$1 = \int_{4\pi} d\Omega C = C \int_0^{2\pi} \int_0^{\pi} \sin\theta d\theta d\phi = C(4\pi)\,,$$
$$\Rightarrow C = \frac{1}{4\pi} = \left(\frac{1}{2}\right)\left(\frac{1}{2\pi}\right)\,.$$

Now the direction inversion sampling method is to set a random number equal to the cumulative density function (CDF) and invert. We can do this for θ and ϕ separately, using random numbers $\xi \in (0, 1)$ and $\eta \in (0, 1)$

$$\begin{aligned}
\xi &= \int_0^{\theta} \frac{1}{2}\sin\theta' d\theta' \\
\xi &= -\frac{1}{2}\cos\theta'\Big|_0^{\theta} \\
\xi &= \frac{1}{2}(1 - \cos\theta) \\
\cos\theta &= -(2\xi - 1) \\
\theta &= \cos^{-1}(-(2\xi - 1)) \\
\Rightarrow \theta &= \cos^{-1}(2\xi - 1)
\end{aligned}$$

$$\begin{aligned}
\eta &= \int_0^{\phi} \frac{1}{2\pi} d\phi' \\
\eta &= \frac{1}{2\pi}\phi'\Big|_0^{\phi} \\
\eta &= \frac{1}{2\pi}\phi \\
\Rightarrow \phi &= 2\pi\eta
\end{aligned}$$

These two randomly sampled angles will produce a neutron direction that 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?

$\Rightarrow$ 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

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

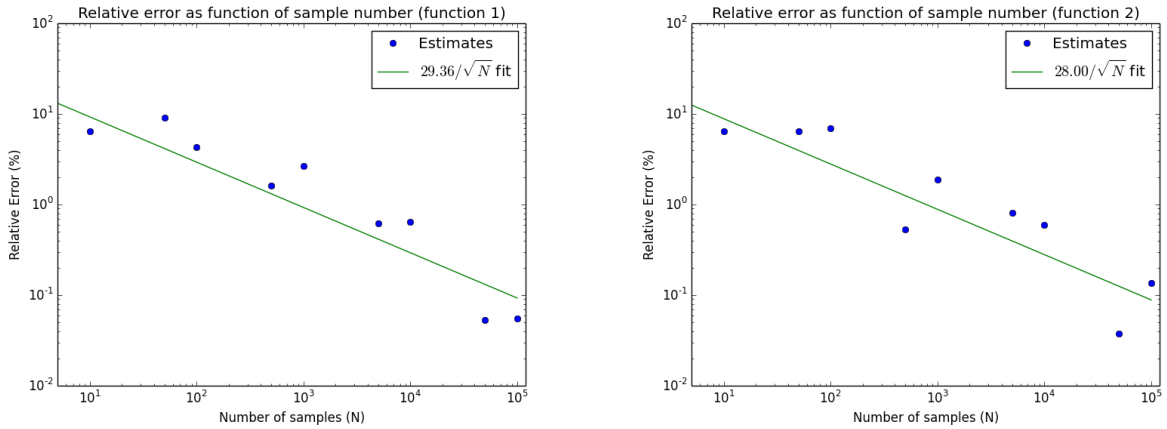
and plotted together. This is because we expect the relative error to decrease with the number of samples as $1/\sqrt{N}$. The code is shown below and the relative error results for a single execution (for both functions) are tabulated in Table 1 and plotted in Fig. 1 with the fitted function. The results will change with each execution, when using a different random seed each time.

```
1  # NE 255 - Homework 6 - Problem 2
2  # D. Hellfeld
3  # 12/1/16
4
5  # Imports
6  import numpy as np
7  import matplotlib.pyplot as plt
8  import random as random
9  from scipy.optimize import curve_fit
10
11 # Actual PDF(s)
12 def p1(x):
13     return 4. * np.sqrt(1. - x**2)
14
15 def p2(x):
16     return 4. * (1. / (1. + x**2))
17
18 # Simple enclosing PDF (uniform), defined on [a,b]
19 def g(x,a,b,const):
20     if (x >= a and x <= b):
21         return const
22     else:
23         return 0.
24
25 def G(xi,a,b,const):
26     return (xi * (b-a) + a)
27
28 # Define variables
29 N            = np.asarray([10,50,100,500,1000,5000,10000,50000,100000])
30 a            = 0.
31 b            = 1.
32 const        = 4.2
33 pi_exact     = 3.14159
34 pi_estimate_1 = (np.zeros_like(N)).astype(float)
35 pi_estimate_2 = (np.zeros_like(N)).astype(float)
36 rel_error_1  = (np.zeros_like(N)).astype(float)
37 rel_error_2  = (np.zeros_like(N)).astype(float)
38
39 # Loop through different sample values
40 itr = 0
41 for n in N:
42
43     # Sample
44     accept_1 = 0.
45     accept_2 = 0.
46     for i in range(int(n)):
47
48         xi  = random.random()
49         eta = random.random()
50         x   = G(xi, a, b, const)
51
```

```
        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.