NE 255 HW 6

Adam Glick

Wednesday 30th November, 2016

1. (30 points) Using the direct inversion of CDF sampling method, derive sampling algo- rithms for

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

Direct Inversion Algorithm

$$\hat{x} = F^{-1}(\xi) \tag{1}$$
$$\bar{\Omega} = \Omega_x\hat{x} + \Omega_y\hat{y} + \Omega_z\hat{z} \tag{2}$$
$$\frac{d\Omega}{4\pi} = \frac{\sin(\theta)d\theta d\phi}{4\pi} = \frac{-d(\cos(\theta))d\phi}{4\pi} = \frac{-d\mu d\phi}{4\pi} \tag{3}$$
$$f(\bar{\Omega}) = f_1(\mu)f_2(\phi) = \frac{1}{2}\frac{1}{2\pi} \tag{4}$$
$$F_1(\mu) = \int_{-1}^{\mu} f_1(\mu')d\mu' = \frac{1}{2}(\mu+1) = \xi_1 \Longrightarrow \boxed{\mu = 2\xi_1 - 1} \tag{5}$$
$$F_1(\phi) = \int_{0}^{\phi} f_2(\phi')d\phi' = \frac{\phi}{2\pi} = \xi_2 \Longrightarrow \boxed{\phi = 2\pi\xi_2} \tag{6}$$

(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 R1, R2, and R3, each made of different materials with total cross sections t1, t2, and t3, respectively.

$$n = \Sigma_t(s)s \Longrightarrow dn = \Sigma_t(s)ds \tag{7}$$
$$n_b = \Sigma_t(R_1)R_1 \tag{8}$$
$$n_c = -\ln(1-\xi) \tag{9}$$

To start the algorithm, we first transform all distances into units of mean free path. This helps us deal with inhomogeneous materials and boundaries between materials. The value n_b represents the boundary mean free path length, which is a discrete value and not random. However we generate the collision distance by using random sampling where ξ is a randomly generated value between 0 and 1. The probability density function of the particle, independent of the material is

$$p_c(n_c)dn = e^{-n_c}dn_c \tag{10}$$

The algorithm to find out the distance to the next particle interaction is then:

```
i = 0 % set up a counter
while n_b < n_c do
    n_c = n_c − n_b %This keeps going if the collision distance is larger than the boundary
    n_b = Σ_t(R_i)R_i %Calculate the second boundary distance in mean free path
    i = i+1
end
display(i) % This will tell which region the particle collided in
display(n_c)%This will tell where in the region the particle collided
```

(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,gamma) capture interactions. Assume monoenergetic neutron transport.

$$\Sigma_t = \Sigma_{elastic} + \Sigma_{inelastic} + \Sigma_{(n,2n)} + \Sigma_{(n,\gamma)} \quad (11)$$

$$= \sum_{x=1}^{R} \Sigma_{x,j} \quad (12)$$

$$p_x = \frac{\Sigma_{x,j}}{\Sigma_{t,j}} \quad (13)$$

$$dN_x = N_0 p_x \Sigma_x e^{-\Sigma_x x} dx \quad (14)$$

2. (20 points) Use a rejection Monte Carlo method to evaluate = 3.14159:

from: $\pi = 4\int_0^1 \sqrt{1x^2}dx$

from: $\pi = 4\int_0^1 \frac{1}{1+x^2}dx$

Assuming that = 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 number of trials?

```
piRandE1 =

    2.8000

errE1 =

    0.1087

piRandE2 =

    3.0400

errE2 =

    0.0323

piRandE3 =

    3.1200

errE3 =

    0.0069

piRandE4 =

    3.1488

errE4 =

    0.0023

>>
```

```
piRandE1 =

    3.6000

errE1 =

    0.1459

piRandE2 =

    3.3200

errE2 =

    0.0568

piRandE3 =

    3.2120

errE3 =

    0.0224

piRandE4 =

    3.1200

errE4 =

    0.0069

>>
```

It is important to notice that the more trials that are ran, the closer the estimation of pi converges to the actual value. It is also worth noting that since the values generated are random, it is possible that with a short number of trials we can potentially converge closer to the value of pi than for larger trials. The increase in the number of trials not only converges to the value of pi better than shorter numbers of trials, but the results produced from large number of trials are more consistent. If you run the 10 trial code multiple times it is very unlikely you get a similar error each run.

MATLAB Code:

```
clc
clear
piRandE1 = 0;
errE1 = 0;
piRandE2 = 0;
errE2 = 0;
piRandE3 = 0;
errE3 = 0;
piRandE4 = 0;
errE4 = 0;


rng('shuffle');
for i = 1:10

    dRand = rand(2,1);
    r = sqrt(1-dRand(1,1)^2);
    if dRand(2,1)<r || dRand(2,1)==r
        piRandE1 = piRandE1+1;
    end
end
rng('shuffle');
for i = 1:1E2

    dRand = rand(2,1);
    r = sqrt(1-dRand(1,1)^2);
    if dRand(2,1)<r || dRand(2,1)==r
        piRandE2 = piRandE2+1;
    end
end
rng('shuffle');
for i = 1:1E3

    dRand = rand(2,1);
    r = sqrt(1-dRand(1,1)^2);
    if dRand(2,1)<r || dRand(2,1)==r
        piRandE3 = piRandE3+1;
    end
end
rng('shuffle');
for i = 1:1E4
    dRand = rand(2,1);
```

```
    r = sqrt(1-dRand(1,1)^2);
    if dRand(2,1)<r || dRand(2,1)==r
        piRandE4 = piRandE4+1;
    end
end

piRandE1 = 4*piRandE1/1E1;
errE1 = abs(3.14159-piRandE1)/(3.14159);
display(piRandE1)
display(errE1)

piRandE2 = 4*piRandE2/1E2;
errE2 = abs(3.14159-piRandE2)/(3.14159);
display(piRandE2)
display(errE2)

piRandE3 = 4*piRandE3/1E3;
errE3 = abs(3.14159-piRandE3)/(3.14159);
display(piRandE3)
display(errE3)

piRandE4 = 4*piRandE4/1E4;
errE4 = abs(3.14159-piRandE4)/(3.14159);
display(piRandE4)
display(errE4)
```

```
clc
clear
piRandE1 = 0;
errE1 = 0;
piRandE2 = 0;
errE2 = 0;
piRandE3 = 0;
errE3 = 0;
piRandE4 = 0;
errE4 = 0;


rng('shuffle');
for i = 1:10

    dRand = rand(2,1);
    r = 1/(1+dRand(1,1)^2);
    if dRand(2,1)<r || dRand(2,1)==r
        piRandE1 = piRandE1+1;
    end
end
rng('shuffle');
for i = 1:1E2

    dRand = rand(2,1);
    r = 1/(1+dRand(1,1)^2);
    if dRand(2,1)<r || dRand(2,1)==r
```

```
        piRandE2 = piRandE2+1;
    end
end
rng('shuffle');
for i = 1:1E3

    dRand = rand(2,1);
    r = 1/(1+dRand(1,1)^2);
    if dRand(2,1)<r || dRand(2,1)==r
        piRandE3 = piRandE3+1;
    end
end
rng('shuffle');
for i = 1:1E4
    dRand = rand(2,1);
    r = 1/(1+dRand(1,1)^2);
    if dRand(2,1)<r || dRand(2,1)==r
        piRandE4 = piRandE4+1;
    end
end

piRandE1 = 4*piRandE1/1E1;
errE1 = abs(3.14159-piRandE1)/(3.14159);
display(piRandE1)
display(errE1)

piRandE2 = 4*piRandE2/1E2;
errE2 = abs(3.14159-piRandE2)/(3.14159);
display(piRandE2)
display(errE2)

piRandE3 = 4*piRandE3/1E3;
errE3 = abs(3.14159-piRandE3)/(3.14159);
display(piRandE3)
display(errE3)

piRandE4 = 4*piRandE4/1E4;
errE4 = abs(3.14159-piRandE4)/(3.14159);
display(piRandE4)
display(errE4)
```