Okeefe Niemann
5/9/2014
1281465
PHYS 115

Assignment 5

Question 1

Part a

Comment: See handwritten page on back for analytic solution.

Part b

- Input:

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>


int main()
{
        int N = 12;
        int M = 10000;
        int i, j, k;
        double x[N], Y, moment;

for(k = 1; k < 7; k ++)
{
        moment = 0;

        for( j = 0; j < M; j++)
        {
                Y = 0;

                for(i = 0; i < N; i++)
                {
                        x[i] = rand() / (RAND_MAX + 1.0);

                        Y += sqrt(12 / N) * (x[i] - 0.5);
                }

                moment += pow(Y, k) / M;
         }

         printf("<X^%d> = %f\n", k, moment);
}

return 0;
}
```

- Output:

```
<X^1> = 0.001541
<X^2> = 0.970864
```

```
<X^3> = 0.051299
<X^4> = 3.102080
<X^5> = 0.112035
<X^6> = 14.217256
```

Question 2

Part a/c

See handwritten page for analytic solution

Part b

- Input:

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>

int main()
{
        int i, N = 100000;
        double x[N], value[N], prob, ab;
        srand(time(NULL));

        prob = 0;

        for(i = 0; i < N; i++)
        {
                x[i] = rand() / (RAND_MAX + 1.0);

                value[i] = tan(M_PI * x[i]);

                if(value[i] < 1.0 && value[i] > -1.0)
                        prob += 1.0 / N;
        }

        printf("probability |x| < 1: %f percent\n", prob);
}
```

- Output:

```
probability |x| < 1: 0.500420 percent
```

Comment: Compared with the exact answer calculated on the back sheet ($\frac{1}{7}2$), the numerical answer has precision 10^{-3}.

Question 3

- Input:

```
#include <stdio.h>
#include <time.h>
#include <math.h>
```

```
#include <stdlib.h>

//defines function to be integrated
double f(double x)
{
        return log(x);
}

int main()
{
        double x, a, b, exact, favg, f2avg, error, nsigma;
        int i, NPOINTS;

        NPOINTS = 100000;
        exact = log(4.0) - 1.0;                 //exact answer
        b = 2.0;                                //upper bound
        a = 1.0;                                //lower bound
        srand(time(NULL));                      //initializes random number generator

        favg = 0;
        f2avg = 0;

        for(i = 0; i < NPOINTS; i++)
        {
                x = a + (b - a) * rand() / RAND_MAX;        //computes random values for x
                favg += f(x);                                  //computes f(x)
                f2avg += f(x) * f(x);                          //computes f(x)^2
        }

        favg /= NPOINTS;
        f2avg /= NPOINTS;

        error = sqrt((f2avg - favg*favg)/ (NPOINTS - 1));    //error bar, or standard deviation

        printf("number of points = %d\n", NPOINTS);
        printf("analytic answer = %f\n", exact);
        printf("computational answer = %f\n", (b - a) * favg);
        printf("error bar = %f\n", error);
        printf("# standard deviations = %fsigma\n", fabs((favg - exact) / error));

return 0;
}
```

- Output:

```
number of points = 100000
analytic answer = 0.386294
computational answer = 0.385842
error bar = 0.000625
# standard deviations = 0.724393sigma
```

Comment: The exact answer was about 0.724σ away from the answer calculated by Monte Carlo integration, meaning that the error is acceptable for our purposes. While the standard deviation seems to remain consistent as the number of points increases, the error bar dramatically decreases and our computational answer becomes increasingly more accurate.

Question 4

Part a

Comment: See attached handwritten page for analytic solution.

Part b

- Input:

```
#include <stdio.h>
#include <time.h>
#include <math.h>
#include <stdlib.h>

int main()
{
        double x[10], a, b, exact, favg, f2avg, error, nsigma;
        int i, j, NPOINTS;

        NPOINTS = 100000;
        exact = 155 / 6.0;      //exact answer
        b = 1.0; //upper bound
        a = 0; //lower bound
        srand(time(NULL)); //initializes random number generator

        favg = 0;
        f2avg = 0;

        for(i = 0; i < NPOINTS; i++)
        {
                for(j = 0; j < 10; j++)
                {
                        x[j] = a + (b - a) * rand() / RAND_MAX;    //computes random values for x
                }

         //computes the values for <f> and <f>^2
         favg += pow(x[0] + x[1] + x[2] + x[3] + x[4] + x[5] + x[6] + x[7] + x[8] + x[9], 2);
         f2avg += pow(pow(x[0] + x[1] + x[2] + x[3] + x[4] + x[5] + x[6] + x[7] + x[8] + x[9], 2),2);
        }

        favg /= NPOINTS;
        f2avg /= NPOINTS;

        error = sqrt((f2avg - favg*favg)/ (NPOINTS - 1)); //error bar

        printf("number of points = %d\n", NPOINTS);
        printf("analytic answer = %f\n", exact);
        printf("computational answer = %f\n", (b - a) * favg);
        printf("error bar = %f\n", error);
        printf("# standard deviations = %fsigma\n", fabs((favg - exact) / error));

return 0;
}
```

- Output:

```
number of points = 100000
analytic answer = 25.833333
computational answer = 25.819427
error bar = 0.029112
# standard deviations = 0.477689sigma
```

Comment: The number of standard deviations refer to how many values of σ away the exact answer is from the above Monte Carlo calculation.

Question 5

- Input:

```
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <stdlib.h>

int main()
{
        int i, t, NUMSTEPS;
        srand(time(NULL));
        float array[10] = {10, 50, 100, 200, 300, 400, 500, 1000, 10000, 100000};
        double xavg[10], x2avg[10], epsilon;

        for( i = 0; i < 10; i ++)
        {
                NUMSTEPS = array[i];

                for(t = 0; t < NUMSTEPS; t++)
                {
                        epsilon = rand()/(RAND_MAX + 1.0);

                        if(epsilon <= 0.5)
                                epsilon = -1.0;
                        else
                                epsilon = 1.0;

                        xavg[i] += epsilon;

                        x2avg[i] += epsilon * epsilon;

                }
        }

        printf("# Steps       <x>          <x^2>\n");

        for(i = 0; i < 10; i++)
        {
                printf("%.0f        %.1f    %.1f          %.0f\n", array[i], xavg[i], xavg[i] / array[i
        }

return 0;
}
```

- Output:

# Steps	<x>	<x>/Steps	<x^2>
10	4.0	0.400000	10
50	2.0	0.040000	50
100	0.0	0.000000	100
200	-16.0	-0.080000	200
500	14.0	0.028000	500
1000	-4.0	-0.004000	1000
10000	246.0	0.024600	10000
100000	-48.0	-0.000480	100000
1000000	-438.0	-0.000438	1000000
10000000	-3102.0	-0.000310	10000000

Comment: Though $<x>$ increases, ratio of $<x>$ to the number of steps steadily converges to 0.