Augustine Stav
Professor Onuttom Narayan
April 27, 2016

Physics 242 Homework 4

1. (a) $< X >= \sqrt{\frac{12}{N}} \sum_{i=1}^{N} (< x_i - \frac{1}{2} >) = \sqrt{\frac{12}{N}} \sum_{i=1}^{N} (< x_i > - \frac{1}{2})$

$where \ < x_i >= \int_0^1 x_i dx_i = \frac{1}{2} \rightarrow < X >= 0$

$\sigma_X^2 = \frac{12}{N} \sum_{i=1}^{N} \sigma_{x_i}^2 = 12\sigma_{x_i}^2 \ with \ no \ correlation \ between \ the \ x_i$

$where \ \sigma_{x_i}^2 =< (x_i - \frac{1}{2})^2 > - < x_i - \frac{1}{2} >^2 =< (x_i - \frac{1}{2})^2 >= \int_0^1 (x_i - \frac{1}{2})^2 dx_i = \frac{1}{12}$

$\rightarrow \sigma_X^2 = 1$

1. (b) Source:

```
package hw4prob1;
import java.util.Random;

public class Hw4Prob1 {
    public static void calcMoments(double[] moments, Random rand) {
        double sum = -6.0; //-1/2 times 12
        for (int i = 0; i < 12; i++) {
            sum += rand.nextDouble();
        }
        for (int i = 0; i < moments.length; i++) {
            moments[i] += Math.pow(sum, i+1);
        }
    }

    public static void main(String[] args) {
        double[] moments = new double[6];
        double x;
        int trials = 50000;
        Random rand = new Random();
        for (int i = 0; i < trials; i++) {
            calcMoments(moments, rand);
        }
        System.out.println("The first 6 moments of X:");
        for (int i = 0; i < moments.length; i++) {
            System.out.format("<X^%d> = %f%n", i+1, moments[i]/trials);
        }
    }
}
```

Output:

```
The first 6 moments of X:
<X^1> = -0.008004
<X^2> = 1.004561
<X^3> = -0.023291
<X^4> = 2.939720
<X^5> = -0.097072
<X^6> = 13.913565
```

1. (c):

$let \ y_i = x_i - \frac{1}{2} \ then \ < X^4 >= \frac{144}{N^2} < \sum_{i=1}^{N} y_i >^4$

$$< y_i >= 0,\ < y_i^2 >= \frac{1}{12}, ...,\ only\ even\ moments,$$
$$so\ \ < X^4 >= \frac{12^2}{N^2}[\sum_i < y_i^4 > + \sum_{j,k} < y_j^2 >< y_k^2 >]\ where\ j \neq k$$
There are N terms in the first sum. There are $C_2^4 \times N \times (N-1) \times \frac{1}{2}$ = $3N^2$ - 3N terms in the second sum because there are 4 choose 2 ways to arrange j and k out of 4 factors, e.g. $y_j y_k y_j y_k$ or $y_j y_j y_k y_k$. This is multiplied by N possible choices for picking j times (N - 1) choices for k. The ½ factor accounts for choosing j then k being the same as choosing k then j.
$$\int_{\frac{1}{2}}^{-\frac{1}{2}} y_i^4 dy_i = \frac{1}{80},\ so\ \ < X^4 >= \frac{12^2}{N^2}[\frac{N}{80} + \frac{3N^2 - 3N}{12^2}] = 3 - 6/(5N)$$
$<X^4>$ = 2.9 with N = 12 agrees with part (b).
$$< X^6 >= \frac{12^3}{N^3}[\sum_i < y_i^6 > + \sum_{j,k} < y_j^4 >< y_k^2 > + \sum_{l,m,n} < y_l^2 >< y_m^2 >< y_n^2 >]$$
There are N terms in the first sum. There are $C_2^6 \times N \times (N-1)$ = $15N^2$ - 15N terms in the second sum. Choosing j then k or k then j are different, so there is no ½ factor. There are $\frac{6!}{2!^3} \times N \times (N-1) \times (N-2) \times \frac{1}{3!}$ = $15N^3 - 45N^2$ + 30N terms in the third sum. The 1/3! factor prevents overcounting (the order of choosing l,m,n does not matter).
$$\int_{\frac{1}{2}}^{-\frac{1}{2}} y_i^6 dy_i = \frac{1}{448}$$
$$< X^6 >= \frac{12^3}{N^3}[\frac{N}{448} + \frac{15N^2 - 15N}{80 \times 12} + \frac{15N^3 - 45N^2 + 30N}{12^3}] = 15 - 18/N + 48/(7N^2)$$
$<X^6>$ 13.5476 when N = 12 agrees with part (b).

2. Source:

```
package hw4prob2;
import java.util.Random;

public class Hw4Prob2 {
    static double function(double x) {
        return Math.log(x);
    }

    public static void main(String[] args) {
        double a = 1.0;
        double b = 2.0;
        double interv = b - a;
        int trials = 50000; //N
        Random rand = new Random();
        double x;
        double func;
        double funcSum = 0;
        double funcSquareSum = 0;
        double sumAve;
        double squareSumAve;
        double stdDev;

        for (int i = 0; i < trials; i++) {
            x = a + rand.nextDouble()*interv;
```

```
                func = function(x);
                funcSum += func;
                funcSquareSum += func*func;
            }
            sumAve = funcSum/trials;
            squareSumAve = funcSquareSum/trials;
            stdDev = Math.sqrt(Math.abs(squareSumAve - sumAve*sumAve)/(trials - 1));
            System.out.println("Monte Carlo Integration of ln(x) from x = 1 to 2:");
            System.out.format("Integral: %f ± %f%n", sumAve, stdDev);
        }
    }
```

Output:

```
Monte Carlo Integration of ln(x) from x = 1 to 2:
Integral: 0.386069 ± 0.000884
```

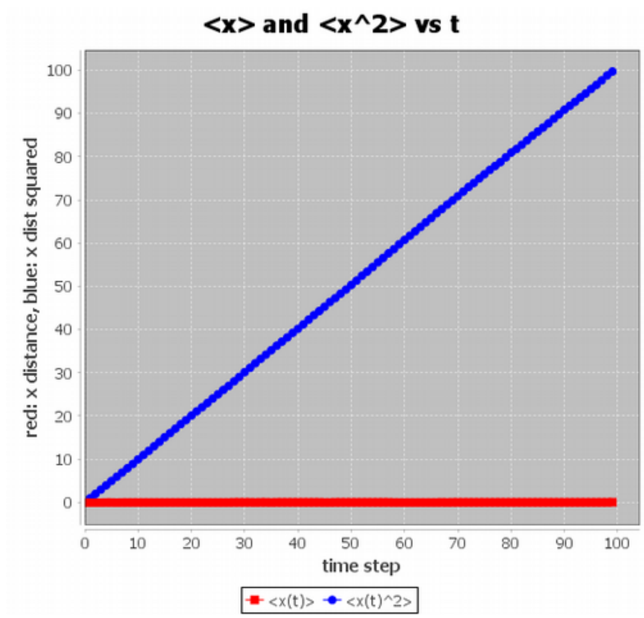
3. Source:

```
package hw4prob3;
import java.util.Random;
import java.awt.Color;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.data.xy.XYSeries;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.xy.XYSeriesCollection;
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;

public class Hw4Prob3 extends ApplicationFrame {
    public Hw4Prob3(String applicationTitle, String chartTitle,
            XYSeriesCollection dataset) {
        super(applicationTitle);
        JFreeChart xylineChart = ChartFactory.createScatterPlot(
                chartTitle , "time step" ,
                "red: x distance, blue: x dist squared" , dataset,
                PlotOrientation.VERTICAL, true ,true ,false);
        ChartPanel chartPanel = new ChartPanel( xylineChart );
        chartPanel.setPreferredSize( new java.awt.Dimension( 560 , 367 ) );
        final XYPlot plot = xylineChart.getXYPlot( );
        XYLineAndShapeRenderer renderer = new XYLineAndShapeRenderer( );
        renderer.setSeriesPaint( 0 , Color.RED );
        renderer.setSeriesPaint( 1 , Color.BLUE );
        plot.setRenderer( renderer );
        setContentPane( chartPanel );
    }
    public static void main(String[] args) {
        int trials = 50000;
        int tMax = 100;
        double x;
        double xSquared;
        double[] xAve = new double[tMax];
        double[] xSquaredAve = new double[tMax];
        Random rand = new Random();

        for (int i = 0; i < trials; i++) {
            x = 0;
            for (int t = 1; t < tMax; t++) {
                if (rand.nextDouble() < 0.5) {
                    x += 1;
                } else {
                    x -= 1;
                }
                xAve[t] += x;
```

```
                xSquaredAve[t] += x*x;
            }
        }

        XYSeries xPoints = new XYSeries("<x(t)>");
        XYSeries xSquaredPoints = new XYSeries("<x(t)^2>");

        System.out.println("x-axis Random Walk:");
        System.out.format("%5s %10s %10s%n", "t", "<x(t)>", "<x(t)^2>");
        for (int t = 0; t < tMax; t++) {
            x = xAve[t]/trials;
            xSquared = xSquaredAve[t]/trials;
            xPoints.add(t, x);
            xSquaredPoints.add(t, xSquared);
            if (t%10 == 0) {
                System.out.format("%5d %10.3f %10.3f%n", t, x, xSquared);
            }
        }

        XYSeriesCollection dataset = new XYSeriesCollection( );
        dataset.addSeries(xPoints);
        dataset.addSeries(xSquaredPoints);
        Hw4Prob3 chart = new Hw4Prob3("<x> and <x^2> vs t",
                "<x> and <x^2> vs t", dataset);
        chart.pack( );
        RefineryUtilities.centerFrameOnScreen( chart );
        chart.setVisible( true );
    }
}
```

Output:

```
x-axis Random Walk:
    t     <x(t)>   <x(t)^2>
    0      0.000      0.000
   10     -0.009      9.968
   20      0.006     20.103
   30      0.025     30.152
   40      0.031     40.178
   50      0.025     50.324
   60      0.018     60.746
   70      0.028     70.864
   80      0.032     80.934
   90      0.029     90.839
```



4. Source:

```
package hw4prob4;
import java.util.ArrayList;
import java.util.Random;
import java.util.Scanner;

public class Hw4Prob4 {
    static double gasDev(ArrayList<Double> betas, Random rand) {
        double fac, rsq, v1, v2;
        if (betas.isEmpty()) {
            do {
               v1 = 2.0*rand.nextDouble() - 1.0;
               v2 = 2.0*rand.nextDouble() - 1.0;
               rsq = v1*v1 + v2*v2;
```

```java
            } while (rsq >= 1.0 || rsq == 0.0);
            fac = Math.sqrt(-2.0*Math.log(rsq)/rsq);
            betas.add(v1*fac);
            return v2*fac;
        } else {
            return betas.remove(0);
        }
    }

    public static double f(double x) { //force
        //V = x^2 /2, F = -dV/dx = -x
        return -x;
    }

    public static void velocityVerlet(double h, double[] y, double beta)
    {
        double xN = y[0]; //get current x before updating to n + 1 to calculate
                          //v_n+1
        y[0] += h*(y[1] + (h * f(xN) + beta)/2.0)/(1.0 + h/2.0); //x_n+1
        y[1] += h*(f(xN) + f(y[0]))/2.0 - (y[0] - xN) + beta; //v_n+1
    }

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.println("Enter seed:");
        int seed = in.nextInt();
        Random rand = new Random(seed);
        ArrayList<Double> betas = new ArrayList<>();
        double[] y = {0.0, 0.0}; //y[0] = x, y[1] = v
        //initialize x(0)= 0 and v(0) = 0 (particle at rest at origin)
        double h = 0.1;
        double stdDevFac = Math.sqrt(2*h); //beta has variance 2h
        int trials = 10000000;
        //xEvenMom[0] = x^2 sum, xEvenMom[1] = x^4 sum, xEvenMom[2] = x^6 sum
        double[] xEvenMom = new double[3]; //the first three even moments of x

        System.out.format("%n%10s %10s %10s %10s %10s%n", "time step", "x(t)",
                "x(t)^2", "x(t)^4", "x(t)^6");
        for (int i = 0; i <= trials; i++) {
            velocityVerlet(h, y, stdDevFac * gasDev(betas, rand));
            xEvenMom[0] += y[0]*y[0];
            xEvenMom[1] += y[0]*y[0]*y[0]*y[0];
            xEvenMom[2] += y[0]*y[0]*y[0]*y[0]*y[0]*y[0];
            if (i%1000000 == 0) {
                System.out.format("%10d %10.3f %10.3f %10.3f %10.3f%n", i,
                        y[0], y[0]*y[0], y[0]*y[0]*y[0]*y[0],
                        y[0]*y[0]*y[0]*y[0]*y[0]*y[0]);
            }
        }
        System.out.format("%n%9s %7s %7s %7s%n", "", "<x^2>", "<x^4>", "<x^6>");
        System.out.format("%9s %7.3f %7.3f %7.3f%n", "numerical",
                xEvenMom[0]/trials, xEvenMom[1]/trials, xEvenMom[2]/trials);

        //Boltzmann dist.: p(x) = A*exp(-x^2 /2)
        //normalization: A = 1/sqrt(2*pi)
        //<x^2> = integral x^2 * p(x) dx, x=-inf..inf and similarly for higher
        //moments. The analytic values for the even moments follow:
        System.out.format("%9s %7.3f %7.3f %7.3f%n", "analytic",
                1.0, 3.0, 15.0);
    }
}
```

Output (i):

```
Enter seed:
654867
```

```
   time step       x(t)      x(t)^2      x(t)^4      x(t)^6
           0     -0.005       0.000       0.000       0.000
     1000000      0.500       0.250       0.062       0.016
     2000000     -1.142       1.304       1.702       2.220
     3000000      1.455       2.117       4.481       9.485
     4000000      0.909       0.827       0.683       0.565
     5000000     -1.395       1.947       3.790       7.378
     6000000      0.299       0.089       0.008       0.001
     7000000     -0.519       0.269       0.072       0.019
     8000000      0.768       0.590       0.348       0.205
     9000000      0.363       0.132       0.017       0.002
    10000000      1.453       2.110       4.452       9.395

               <x^2>   <x^4>   <x^6>
  numerical    0.998   2.979  14.815
   analytic    1.000   3.000  15.000
```

Output (ii):

```
Enter seed:
91451

   time step       x(t)      x(t)^2      x(t)^4      x(t)^6
           0      0.022       0.000       0.000       0.000
     1000000     -0.869       0.755       0.570       0.431
     2000000     -0.331       0.109       0.012       0.001
     3000000      1.109       1.230       1.512       1.859
     4000000      0.872       0.760       0.578       0.439
     5000000      1.167       1.361       1.853       2.523
     6000000     -0.303       0.092       0.008       0.001
     7000000      0.800       0.641       0.410       0.263
     8000000     -1.277       1.631       2.660       4.338
     9000000      1.295       1.677       2.811       4.713
    10000000     -0.192       0.037       0.001       0.000

               <x^2>   <x^4>   <x^6>
  numerical    1.000   3.002  15.100
   analytic    1.000   3.000  15.000
```

Output (iii):

```
Enter seed:
74138

   time step       x(t)      x(t)^2      x(t)^4      x(t)^6
           0     -0.013       0.000       0.000       0.000
     1000000      1.698       2.882       8.307      23.944
     2000000      0.422       0.178       0.032       0.006
     3000000      1.390       1.932       3.733       7.213
     4000000      0.707       0.500       0.250       0.125
     5000000      0.187       0.035       0.001       0.000
     6000000      0.582       0.338       0.115       0.039
     7000000      0.188       0.035       0.001       0.000
     8000000     -0.633       0.400       0.160       0.064
     9000000     -0.318       0.101       0.010       0.001
    10000000     -1.070       1.146       1.312       1.503

               <x^2>   <x^4>   <x^6>
  numerical    1.001   3.003  15.011
   analytic    1.000   3.000  15.000
```

Output (iv):

```
Enter seed:
82465
```

```
 time step       x(t)      x(t)^2     x(t)^4     x(t)^6
         0     -0.013      0.000      0.000      0.000
   1000000     -0.626      0.392      0.154      0.060
   2000000     -0.824      0.679      0.462      0.314
   3000000     -1.433      2.053      4.215      8.654
   4000000      0.505      0.255      0.065      0.017
   5000000      0.244      0.059      0.004      0.000
   6000000     -2.035      4.142     17.159     71.080
   7000000     -0.102      0.010      0.000      0.000
   8000000      0.834      0.696      0.484      0.337
   9000000      0.010      0.000      0.000      0.000
  10000000     -1.251      1.565      2.449      3.832

             <x^2>   <x^4>   <x^6>
numerical    0.996   2.970  14.713
 analytic    1.000   3.000  15.000
```

Output (v):

```
Enter seed:
123456789

 time step       x(t)      x(t)^2     x(t)^4     x(t)^6
         0     -0.011      0.000      0.000      0.000
   1000000      1.412      1.993      3.973      7.919
   2000000     -0.829      0.688      0.473      0.325
   3000000      0.604      0.365      0.133      0.049
   4000000      0.124      0.015      0.000      0.000
   5000000      0.741      0.549      0.302      0.166
   6000000      1.526      2.328      5.418     12.610
   7000000      2.754      7.586     57.554    436.631
   8000000      1.111      1.234      1.522      1.877
   9000000     -0.102      0.010      0.000      0.000
  10000000     -1.241      1.539      2.370      3.648

             <x^2>   <x^4>   <x^6>
numerical    0.997   2.983  14.853
 analytic    1.000   3.000  15.000
```