Augustine Stav
Professor Onuttom Narayan
May 4, 2016

Physics 242 Homework 5

1. (a) Source:

```
package hw5prob1a;
import java.util.Random;

public class Hw5Prob1a {
    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*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: 0.387189 ± 0.001732
```

1. (b) Source:

```
package hw5prob1b;
import java.util.Random;

public class Hw5Prob1b {
    static double function(double x) {
        return 1/(1 + (Math.sin(x)*Math.sin(x)));
    }
    static double prob(double y) {
        return -Math.log(1-y); //p(x) = e^(-x), so x = -ln(1-y) with 0<y<1
    }

    public static void main(String[] args) {
        int trials = 50000; //N
        Random rand = new Random();
        double x;
        double y;
        double funcSum = 0;
        double sumAve;
```

```
            for (int i = 0; i < trials; i++) {
                y = rand.nextDouble();
                x = prob(y);
                funcSum += function(x);
            }
            sumAve = funcSum/trials;
            System.out.println("Monte Carlo Integration of 1/(1 + sin^2(x))) "
                    + "from x = 0 to infinity:");
            System.out.format("Integral: %f%n", sumAve);
        }
    }
```

Output:

```
Monte Carlo Integration of 1/(1 + sin^2(x))) from x = 0 to infinity:
Integral: 0.758564
```

2. Source:

```
package hw5prob2;
import java.util.Random;

public class Hw5Prob2 {
    static void siftDown (double[] ra, int l, int r) { //ra is the heap
        int i, iOld;
        iOld = l;
        double a = ra[l];
        i = 2*l+1;
        while (i <= r) {
            if (i < r && ra[i] < ra[i+1]){
                i++;
            }
            if (a >= ra[i]){
                break;
            }
            ra[iOld] = ra[i];
            iOld = i;
            i = 2*i+1;
        }
        ra[iOld] = a;
    }

    static void heapSort(double[] ra){
        int i, n=ra.length;
        for (i = n/2-1; i >= 0; i--){
            siftDown(ra, i, n-1);
        }
        for (i = n-1; i > 0; i--) {
            double raI = ra[i];
            ra[i] = ra[0];
            ra[0] = raI;
            siftDown(ra, 0, i-1);
        }
    }

    //return the index of the value immediately before searchVal
    static int binarySearch(double[] heap, double searchVal) {
        int i0 = 0;
        int i1 = heap.length;
        int i2;

        while (i1 - i0 > 1){
            i2 = (i0 + i1)/2;
            if (heap[i2] > searchVal){
                i1 = i2;
            } else {
                i0 = i2;
            }
```

```
            }
            return i0;
        }

        public static void main(String[] args) {
            int indexPrev, N = 1000000;
            double searchVal = 0.7;
            double[] heap = new double[N];
            Random rand = new Random();
            long startTime, endTime, totalTime1, totalTime2;

            for (int i = 0; i < N; i++){
                heap[i] = rand.nextDouble();
            }

            startTime = System.currentTimeMillis();
            heapSort(heap);
            endTime   = System.currentTimeMillis();
            totalTime1 = endTime - startTime; //calculate time elapsed to sort
                                             //N-sized list

            System.out.println("i.");
            System.out.println("Heap sort: first 5 sorted:");
            for (int i = 0; i < 5; i++){
                System.out.format("%10.8f%n", heap[i]);
            }
            System.out.println("\nHeap sort: last 5 sorted:");
            for (int i = N-5; i < N; i++){
                System.out.format("%10.8f%n", heap[i]);
            }
            System.out.println("\nii.");
            indexPrev = binarySearch(heap, searchVal);
            System.out.format("%.1f lies between elements %d and %d%n",
                    searchVal, indexPrev, indexPrev+1);

            //calculate the time elapsed for sorting a list 10 times as large
            int N2 = N*10;
            double[] heap2 = new double[N2];

            for (int i = 0; i < N2; i++){
                heap2[i] = rand.nextDouble();
            }

            startTime = System.currentTimeMillis();
            heapSort(heap2);
            endTime   = System.currentTimeMillis();
            totalTime2 = endTime - startTime;
            System.out.println("\niii.");
            System.out.format("%10s %20s%n", "N", "sorting time (ms)");
            System.out.format("%10d %20d%n", N, totalTime1);
            System.out.format("%10d %20d%n", N2, totalTime2);
        }
    }
```

Output:

```
i.
Heap sort: first 5 sorted:
0.00000175
0.00000285
0.00000320
0.00000321
0.00000426

Heap sort: last 5 sorted:
0.99999753
0.99999756
0.99999761
```

```
0.99999848
0.99999902

ii.
0.7 lies between elements 699640 and 699641

iii.
         N    sorting time (ms)
   1000000                 729
  10000000                9541
```

3. (a,b,c) Source:

```
package hw5prob3;
import java.util.Random;

public class Hw5Prob3 {
    public static void main(String[] args) {
        int N = 100; //the number of spins
        int[] spins = new int[N];
        int correlSize = 10;
        double stdDev;
        //correl[j+1] = 1/n sum(S_i * S_(i+j))
        double[] correl = new double[correlSize];
        double[] correlTot = new double[correlSize];
        double[] correlSqTot = new double[correlSize];
        int flipsInitital = 100*N; //the number of flips to discard
        int flipsCalcCorrel = 1000000;
        double J = 1.0; //positive number, ferromagnetic
        double temp = 1.0; //the temperature (with k_boltzmann = 1)
        Random rand = new Random();
        //the frequency of having total spin == key

        for (int i = 0; i < spins.length; i++) {
                spins[i] = 1;
        }

        for (int i = 0; i < flipsInitital; i++) {
            flipAccept(spins, temp, rand, J);
        }

        int trials = 0;
        for (int i = 1; i < flipsCalcCorrel+1; i++) {
            flipAccept(spins, temp, rand, J);
            if (i%(10*N) == 0) {
                trials++;
                calcCorrel(spins, correl);
                for (int j = 0; j < correl.length; j++) {
                    correlTot[j] += correl[j];
                    correlSqTot[j] += correl[j]*correl[j];
                }
            }
        }

        //average the correlations and squares of correlations
        System.out.println("C(j) for a circle of Ising Spins:");
        for (int j = 0; j < correl.length; j++) {
            correlTot[j] = correlTot[j]/trials;
            correlSqTot[j] = correlSqTot[j]/(trials*trials);
            stdDev = Math.sqrt(Math.abs(correlSqTot[j] -
                    correlTot[j]*correlTot[j])/(trials - 1));
            System.out.format("C(%2d) = %8.6f ± %8.6f%n", j+1,
                    correlTot[j], stdDev);
        }
    }
```

```
        //return false if the flip is rejected
        //return true if the flip is accepted
        public static boolean flipAccept(int[] spins, double temp, Random rand,
                double J){
            double energy1 = energy(spins, J);
            double energy2;
            int i = rand.nextInt(spins.length);
            spins[i] *= -1; //flip a random spin
            energy2 = energy(spins, J);
            if (energy2 > energy1) {
                if (rand.nextDouble() >= Math.exp((energy1-energy2)/temp)) {
                    spins[i] *= -1; //the spin change is rejected
                    return false;
                }
            }
            return true;
        }

        public static double energy(int[] spins, double J){
            //initialize energy to the end-beginning ring connection contribution
            double ener = -J* spins[spins.length-1] * spins[0];
            for (int i = 1; i < spins.length; i++) {
                ener += -(J * spins[i] * spins[i-1]);
            }
            return ener;
        }

        public static void calcCorrel(int[] spins, double[] correl) {
            int n = spins.length;
            for (int j = 0; j < correl.length; j++) { //j=1 is really 0 here
               double correlRun = 0.0; //running totals for the correlations at
                                       //different j
               for (int i = 0; i < n; i++) {
                   correlRun += spins[i] * spins[(i+j+1)%n];
               }
               correl[j] = correlRun/n;
            }
        }
    }
```

Output:

```
C(j) for a circle of Ising Spins:
C( 1) = 0.762800 ± 0.024122
C( 2) = 0.582720 ± 0.018427
C( 3) = 0.445000 ± 0.014071
C( 4) = 0.338040 ± 0.010689
C( 5) = 0.257400 ± 0.008138
C( 6) = 0.193840 ± 0.006127
C( 7) = 0.145240 ± 0.004589
C( 8) = 0.108320 ± 0.003420
C( 9) = 0.078600 ± 0.002478
C(10) = 0.059760 ± 0.001879
```

I find that $C(j) \approx 0.76 \times C(j-1)$ or $C(j) \approx 0.76^{j}$.