Assignment 2 - Code Evaluation

Mason Nakamura

October 2021

1 Shuffle Class

```
    public class Shuffle {
    public boolean[] shuffle(boolean[] A){
        int n = A.length;
        for(int i = 0; i < n; i++){
            // pick random number from 0 to i uniformly
            int rand = (int)(Math.random() * (i+1));
            boolean temp = A[rand];
            A[rand] = A[i];
            A[i] = temp;
        }
        return A;
    }
}
```

An implementation of the Knuth shuffle on a boolean array. Used in the Testing_Protocol class.

2 Testing_Protocol Class

```
import java.text.DecimalFormat;

public class Testing_Protocol {

    // number of tests needed
    int total_tests = 0;

    /**
     * Gets the first and last name of this Student.
     * @param people The number of people to be tested
     * @param group_size The group size for initial pooled test
     * @param infection_rate The rate at which the population is being infected
     * @param print_expected Prints the expected number of occurrences of each case and number of tests
     */
    public void pooled(int people, int group_size, double infection_rate, boolean print_expected){
        // Define number of groups
        double num_groups = (double)people/group_size;
        boolean isdivisible = people % group_size == 0;
        // Initialize an array of length "people"
        // All values in "infected" are set to false by default
        boolean[] infected = new boolean[people];
        // infect infection_rate*people number of people
        // take floor if infection_rate*people is rational
        // convert double to int
        int infected_end = (int)Math.floor(infection_rate*people);
        for(int i = 0; i < infected_end; i++){
            infected[i] = true;
        }

        // Knuth shuffle array, infected
        Shuffle shuffle = new Shuffle();
        shuffle.shuffle(infected);

        // Check the pools for infected of size "group_size"
        // if pool has infected, split pool in two and test if each pool has infected
        // then do individual test for each two pools that has infected
        // start index for pooled group
```

```java
        int start_index = 0;
        // end index for pooled group
        // delete 1 to group_size to convert from type size to type index
        int end_index = group_size;
        // number of loops of pooled testing in the population of people
        int loops = 1;

        //TESTING
//          infected[0] = true;
//          infected[5] = true;
//          infected[6] = true;
//          infected[7] = true;
//          infected[8] = true;
//          infected[12] = true;

        while(start_index < people){
            loops += 1;
            // test the whole pool
            total_tests += 1;
            // linear search through pool
            for (int i = start_index; i < end_index; i++) {
                // if someone is infected, then split pool in two
                if (infected[i]) {
                    // Subtract half of group_size from end_index to get get an about halved value between start and end index
                    // Use the ceiling method to take into account case when group_size is odd
                    // Need the (int) and (double) to make it compatible with ceiling method
                    // TODO: TEST that the method below works as intended
                    int half_end_index = (int) Math.ceil(end_index - (double) group_size/2);
                    // Test first half of test pool
                    total_tests += 1;
                    for (int j = start_index; j < half_end_index; j++) {
                        // Test if someone is infected in half pool size
                        if (infected[j]) {
//                              //TESTING
//                              System.out.print("["+ start_index + "," + half_end_index+ "] " );
//                              //TESTING
//                              System.out.print("FIRST HALF: ");
                            // Test everyone in the half pool size
                            for (int k = start_index; k < half_end_index; k++) {
                                total_tests += 1;
//                                  //TESTING
//                                  System.out.print(k + " ");
                            }
//                              //TESTING
//                              System.out.println();
                            // break out of outer for loop to stop linear search, since the pool of size group_size/2 has already been tested on
                            break;
                        }
                    }
                    // test second half of test pool
                    total_tests += 1;
                    for (int j = half_end_index; j < end_index; j++) {
                        // test if someone is infected in half pool size
                        if (infected[j]) {
                            // TODO this for-loop only loops three times not four
//                              //TESTING
//                              System.out.print("["+ half_end_index + "," + end_index + "] " );
//                              //TESTING
//                              System.out.print("SECOND HALF: ");
                            // test everyone in the half pool size
                            for (int k = half_end_index; k < end_index; k++) {
                                total_tests += 1;
//                                  //TESTING
//                                  System.out.print(k + " ");
                            }
//                              //TESTING
//                              System.out.println();
                            // break out of outer for loop to stop linear search,
```

```
                        since the pool of size group_size/2 has already been tested on
                    break;
                }
            }
            // break out of outer for loop to stop linear search, since the pool of size group_size has already been tested on
            break;
        }
    }
    // move to next testing pool
    start_index = end_index;
    // Case when people % group_size != 0
    if (group_size * loops > people){
        end_index = people;
    }
    else{
        end_index = group_size*loops;
    }
}

if (print_expected) {
    // Find the expected percentage of occurrences
    double case1 = Math.pow(1 - infection_rate, group_size);
    double case3 = Math.pow(infection_rate, 2);
    double case2 = 1 - case1 - case3;

    // Find the expected number of tests needed in pooled testing
    int test1 = (int) Math.ceil(num_groups * case1);
    // Initialize test2 and test3
    // Since taking the ceiling of a number less than one will be one
    int test2, test3;
    if (num_groups * case2 < 1){
        test2 = (int) Math.ceil(num_groups * case2) * 7;
    }
    else{
        test2 = (int) Math.ceil(num_groups * case2 * 7);
    }
    // since taking the ceiling of a number less than one will be one
    if (num_groups * case3 < 1){
        test3 = (int) Math.ceil(num_groups * case3) * 11;
    }
    else{
        test3 = (int) Math.ceil(num_groups * case3 * 11);
    }

    // For printing a certain number of decimals
    DecimalFormat numberFormat = new DecimalFormat("#.00000");
    System.out.println("Case 1");
    System.out.println("Expected Percentage of Total Occurrences: " + numberFormat.format(case1));
    // take ceiling to err on the side of caution and convert from double to integer
    System.out.println("Expected Number of Tests: " + test1);

    System.out.println("——————————");
    System.out.println("Case 2");
    System.out.println("Expected Number of Occurrences: " + numberFormat.format(case2));
    System.out.println("Expected Number of Tests: " + test2);

    System.out.println("——————————");
    System.out.println("Case 3");
    System.out.println("Expected Number of Occurrences: " + numberFormat.format(case3));
    System.out.println("Expected Number of Tests: " + test3);

    System.out.println("——————————");
    System.out.println("Total Expected Number of Tests: " + (test1 + test2 + test3));
    System.out.println("——————————");

}
}
}
```

The pool() method takes in the number of people being tested (people), the testing pool sizes (group_size), the infection rate of the virus (infection_rate), and a boolean (print_expected) which outputs the print statements for the expected results. On line 21, I initialize an array of people as all false, indicating that they are not infected. I then infect the first n number of people where n = (number of people * infection rate) in lines 25-28. Then, I implement a Knuth shuffle on the array to randomly shuffle the array of infected and non infected in line 30-32. In line 53, we initialize our while loop which runs until everyone has been tested at least once. In line 58, we test every pool of size group_size to see if someone is infected. If so, we split the test pool in half, add two additional tests for the two subgroups, then check if each subgroup has an infected. If either of the test groups have an infected, we test all the individuals in the subgroup. From lines 124-169, we print the expected number of tests for each case and aggregate them to get the total expected number of tests needed.

3 Testing Class

```
public class Testing {
    public static void main(String[] args) {
        final int ITERATIONS = 100;

        System.out.println();
        System.out.println();
        System.out.println("1,000 PEOPLE");
        System.out.println("——————————");

        Testing_Protocol simulation = new Testing_Protocol();
        simulation.pooled(1000, 8, .02, true);
        System.out.println("Simulated total Tests: " + simulation.total_tests);

        // average the number of tests for n number of simulations/iterations
        int total_total_tests = 0;
        for (int i = 0; i < ITERATIONS; i++){
            Testing_Protocol simulation1 = new Testing_Protocol();
            simulation1.pooled(1000, 8, .02, false);
            total_total_tests += simulation1.total_tests;
//              System.out.println(simulation1.total_tests);
        }
        System.out.println("Averaged Tests over " + ITERATIONS + " iterations: " +
            total_total_tests/ITERATIONS);

        System.out.println();
        System.out.println();
        System.out.println("10,00 PEOPLE");
        System.out.println("——————————");

        simulation.pooled(10000, 8, .02, true);
        System.out.println("Simulated total Tests: " + simulation.total_tests);

        // average the number of tests for n number of simulations/iterations
        total_total_tests = 0;
        for (int i = 0; i < ITERATIONS; i++){
            Testing_Protocol simulation1 = new Testing_Protocol();
            simulation1.pooled(10000, 8, .02, false);
            total_total_tests += simulation1.total_tests;
//              System.out.println(simulation1.total_tests);
        }
        System.out.println("Averaged Tests over " + ITERATIONS + " iterations: " +
            total_total_tests/ITERATIONS);

        System.out.println();
        System.out.println();
        System.out.println("100,000 PEOPLE");
        System.out.println("——————————");

        simulation.pooled(100000, 8, .02, true);
        System.out.println("Simulated total Tests: " + simulation.total_tests);

        // average the number of tests for n number of simulations/iterations
        total_total_tests = 0;
        for (int i = 0; i < ITERATIONS; i++){
            Testing_Protocol simulation1 = new Testing_Protocol();
            simulation1.pooled(100000, 8, .02, false);
```

```
            total_total_tests += simulation1.total_tests;
//            System.out.println(simulation1.total_tests);
        }
        System.out.println("Averaged Tests over " + ITERATIONS + " iterations: " + total_total_tests/ITERATIONS);

        System.out.println();
        System.out.println();
        System.out.println("1M PEOPLE");
        System.out.println("----------------");



        simulation.pooled(1000000, 8, .02, true);
        System.out.println("Simulated total Tests: " + simulation.total_tests);

        // average the number of tests for n number of simulations/iterations
        total_total_tests = 0;
        for (int i = 0; i < ITERATIONS; i++){
            Testing_Protocol simulation1 = new Testing_Protocol();
            simulation1.pooled(1000000, 8, .02, false);
            total_total_tests += simulation1.total_tests;
//            System.out.println(simulation1.total_tests);
        }
        System.out.println("Averaged Tests over " + ITERATIONS + " iterations: " + total_total_tests/ITERATIONS);


    }
}
```

This code shows the expected number of tests needed for differing numbers of people being tested. In this case it's 1,000 people, 10,000 people, 100,000 people, and 1,000,000 people with pool sizes of 8 and infection rate 2%. I also took the average over 100 iterations to compare to the expected value.

4 Binomial vs Hypergeometric

The key difference between the binomial distribution and the hypergeometric distribution is that the binomial distribution's events are independent while the hypergeometric distribution's events depend on the previous events. In essence the binomial distribution draws with replacement while the hypergeometric distribution draws without replacement.

5 Improvements

To improve upon the scalability of the algorithm, we could implement the average false positive of the testing method as well as the false negative, giving a more realistic approach.

6 Results

# of People	1,000	10,000	100,000	1,000,000
Total Expected Number of Tests	249	2378	23714	237129
Averaged Number of Simulated Tests over 100 Iterations	240	2401	23993	239933