Mohammed Mowla Project 2 Submission

**Distributions:** (Contained inside Git branch ‘Distributions’)

Each Java Class (except TchebycheffTheorem.java) in this branch contain the method runAll(). This method runs all nonvoid methods in the class and formats their results to be displayed appropriately.

Program Name: BinomialDistribution

The class was created to handle binomial distribution problems. Such problems have the characteristics of having two outcomes. A success and failure which both have their own respective probabilities. The program allows one to find the probability of success or failure for a given number of trials. This java class has no private variables and therefore the constructor is empty.

Methods:

- The first method is binomialDistribution() and it accepts three integers, ‘n’, ‘r’, and ‘p’. The method returns a BigDecimal. A BigDecimal object is returned because when doing calculations involving factorials the int data type is not large enough to store the resulting number. The variable names are close related to the binomial distribution formula where ‘n’ represents the population, the ‘r’ represents the desired success/failures and ‘p’ represents the probability of either success or failure. What is returned after the calculations are done is the likelihood of success/failure for a ‘r’ given runs.

- The second method is expected() and it accepts two doubles, ‘n’ and ‘p’. It returns a double. The naming expected is another word for the average. Therefore, the calculations done in this method return the average probability of success/failures that are expected for the distribution.

- The third method, variance(), again accepts two doubles ‘n’ and ‘p’. It also returns a double. The method returns the variance of the binomial distribution problem. It is how likely the numbers are to be spread apart for the distribution.

- The fourth method is standardDev(), again accepts two doubles ‘n’ and ‘p’. It also returns a double. The method returns the standard deviation for a binomial distribution problem. This is simply the square root of the variance. In this method the variance() method is called and the variable values passed on to it. The method then returns the square root of what the variance method returns.

Program Name: GeometricDistribution

The class was created to handle geometric distribution problems. Such problems have the characteristics of have two discrete outcomes. Unlike binomial distribution the geometric distribution does not look for the probability of success given several runs. It looks for the probability of a single success at the end of a given number of runs. This java class has no private variables and therefore the constructor is empty.

Methods:

- The first method is geometridDistribution() and it accepts three doubles, ‘p’, ‘q’ and ‘x’. The variable names are closely related to the geometric distribution formula. The ‘p’ is chance of success, ‘q’ chance of failure’ and ‘x’ the number of runs. What is returned after the calculations is the likelihood of a single success.

- The second method is expected() and it accepts one double ‘p’. It returns a double. The method calculates and returns the average number of runs expected before a success.

- The third method is variance() and it accepts one double ‘p’. It returns a double. The method calculates and returns the variance or how much the numbers are likely to be from the average.

- The fourth method is standardDev() and it accepts one double ‘p’. It returns a double. The method calculates and returns the standardDev(). The standard deviation for a hypergeometric distribution is the square root of the variance. Therefore, this method calls the variance() method and returns the square root of what the variance() method returns.

Program Name: HypergeometricDistrbution

The class was created to handle hypergeometric distribution problems. It is used to determine the probability of obtaining a certain number of successes without replacement form a specific group size. This java class has no private variables and therefore the constructor is empty.

Methods:

- The first method is called hypergeometricDistribution() and accepts four integers, ‘bigN’, ‘smallN’, ‘r’ and ‘y’. The variable names are closely related to the hypergeometric distribution formula. The ‘bigN’ is the population, ‘smallN’ is the group, ‘r’ is the desired from the group and ‘y’ ‘r’ and is what the distribution finds. The distribution finds the probability of ‘y’ from the entire population where ‘y’ belongs to the desired from the population ‘r’ in a group of ‘n’ out of ‘N’. After the calculations are done a BigDecimal is returned since factorials are involved in the calculations.

- The second method is called expected() and it accepts three doubles, ‘bigN’, ‘smallN’, and ‘r’. The method calculates the average success for the distribution and then returns it as a double.

- The third method is called variance() and it accepts three doubles ‘bigN’, ‘smallN’, and ‘r’. The method calculates the variance of the distribution and then returns it as a double.

- The fourth method is called standardDev() and it accepts three doubles ‘bigN’, ‘smallN’, and ‘r’. The method calculates the standard deviation for a hypergeometric distribution. The standard deviation is the square root of the variance. Therefore, the method calls the variance() method and takes the return value which it then finds the square root for. It then returns that value.

Program Name: PoissonDistribution

The class was created to handle Poisson distribution problems. Such problems have the characteristics of finding a probability for a per-unit basis such as an event occurring in space, time, volume, etc. The average value for the number of these events is denoted by . This java program has no private variables and therefore the constructor is empty.

Methods:

- The first method is called poissonDistribution() and accepts two integers, ‘x’ and ‘lambda’. The variable names are closely related to the Poisson distribution formula. The ‘x’ relates to the number of runs and ‘lambda’ is the average. The method calculates the probability of an extremely unlikely event to occur in a number of extremely large trials. The method returns the probability as a BigDecimal since factorials are involved in the calculations.

- The second method is called poissonDistrbutino() and accepts three integers, ‘x’, ‘events’, and ‘units’. This differs from the previous method because lambda is not always given for these types of problems. To handle when the average is not given, this method exists. It takes the number of ‘event’s and the number of ‘units’ and divides events by units to find the average for the specific Poisson distribution. It then passes on the ‘x’ and the calculated average to the previous method for completing the calculates. It then returns a BigDecimal.

- The third methos is called expected() and accepts one integer ‘lambda’. Since the average is sometimes given for this distribution and the method simply returns the integer that was put into the parameter as a double.

- The fourth method also called expected() and accepts two integers, ‘events’ and ‘units’. Since the average is sometimes not given as is the case for the second method of this class then this method exists to remedy that. It calculates the average and returns it as a double.

- The fifth method is called variance() and accepts one integer, ‘lambda’. As stated in the third method the average is sometimes given. For Poisson distribution the variance is also equal to the average, therefore this method simply returns the accepted integer as a double.

- The sixth method is called variance() and accepts two integers, ‘events’ and ‘units’. Extremely similar to the fourth method it does exactly what that method does. The difference is that it calls the expected() method and has it accept the parameters to do the calculations.

- The seventh method is called standardDev() and it accepts one integer, ‘lambda’. Since the standard deviation is the square root of the variance this method calls the expected() method and takes the square root of what is returned. It then returns the square root of what the expected method returned.

- The eight method is called standardDev() and accepts two integers, ‘events’ and ‘units’. It like before is very similar to the fourth and sixth methods in the class. It takes the average using ‘events’ and ‘units’ and the takes the square root of this. It then returns the square root of the average.

Program Name: TchebysheffTheorem

The class was created to find the probability of an event occurring within a certain number of standard deviations away from the average. It employs Tchebysheff’s Theorem to do so. This class can be used for any distribution. This java class has no private variables therefore the constructor is empty.

Methods:

- The first method is called tchebysheffTheorem() and accepts three doubles, ‘k’, ‘expected’, ‘standardDev’. The variable names are related to Tchebysheff’s Theorem. The ‘k’ is the number of standard deviations away from the average to check for. The ‘expected’ is the average. The probability of the data falling in withing a ‘k’ number of standard deviations away from the average is calculated as well as the range in which the numbers will fall. The final return is a String that contains all of the data.

- The second method is called tchebysheffTheoremNoK() and accepts three doubles, ‘within’, ‘standardDev’, ‘expected’. This method differs from the previous method because it calculates the number of standard deviations away from the average, ‘k’ in the previous method. The k value is not always given and must be calculated. Once ‘k’ is calculated the parameter values are passed onto the previous method by call and then the return String is what the previous method returns.

Program Name: DistributionTester

This class simply tests all of the distributions in this class as well as Tchebysheff’s Theorem. It constructs objects of each and then has them run the runAll() method in each class except in the TchebysheffTheorem class.

**Data Writing:** (Contained inside Git branch ‘DataWriting’)

These java classes both write and read from .csv files that they create. Their goal is to output the x and y values of the function and then salt it by randomly adding or subtracting from the y values. Finally it smooths out the y values by taking the average of several of the y values and then returning the average for those y values.

Program Name: Plotter

This java class was created to plot the x and y values of onto a .csv file. The constructor for this class initializes imported classes that are needed to create and write to a file. These classes are BufferedWriter and FileWriter. The FileWriter creates the file “RegularData.csv” and the BufferedWriter writes to it. The constructor also initializes two private Strings for the two column titles as well as a two-dimensional array that stores the x and y values. The constructor gives the private Strings and array values by accepting two String parameters and an int parameter. The String parameters set the column title names and the int parameter sets the number of rows for the .csv file that will be created.

Methods:

- The only in the class is called output() and accepts no parameters since the data that is to be written to the .csv file is held in the private variables. The method starts by writing the first column name and writing a column after that. Then comes the second column title. Finally, after a new line is set, the two-dimensional array storing the x and y values is written to the file. Each x value is followed by a comma which is then followed by a y value before moving to a new line. In this way the data is written to the file.

Program Name: GraphSalter

This class is somewhat similar to the previous class except for a few things. The class can read a file as well as create a file and write to it. To read a file the classes File and Scanner are imported. Along with BufferedWriter, FileWriter, ArrayList, and Random, they are initialized in the constructor. Different from the previous class, the x values are read instead of written. They are read from the previous classes created file “RegularData.csv” and the new data is written to “SalterData.csv”. Another difference is that the x and y values are stored in two different ArrayLists.

Methods:

- The only method is called saltDat() and accepts three parameters, two String, ‘columnOne’ and columnTwo’ and one integer, ‘bound’. The strings are aptly named for the titles of the two columns. The ‘bound’ variable is the random number limit from 1 to ‘bound’ in which to add or subtract from a y value. The method takes the data from “RegularData.csv” and salts it by adding or subtracting randomly a random number ranging from 1 – ‘bound’ and then writes it to the new file “SalterData.csv”.

Program Name: GraphSmoother

This class is extremely similar to GraphSalter. It imports all the same classes except the Random class. All of the same imported classes as well as ArrayLists are initialized in the constructor as they were previously. The class is designed to take several a given number of y values and find their average. It then writes them to a with their corresponding x values. The file that this class reads is “SalterData.csv” and writes to “SmootherData.csv”.

Methods:

- The only method is smoothData() and it has three parameters identical to saltData() String ‘columnOne’, String ‘columnTwo’ and int ‘bound’. In this method however ‘bound’ has a different use. It is used to determine how many y values are to be averaged together. After the method averages a ‘bound’ number of them it writes to the new file “SmootherData.csv”.

Program Name: PlotTester

This class is the tester class which creates objects of each class and runs.

**Poker Hand**: (Contained inside Git branch ‘PokerHand’)

This class contains classes that are needed to create a poker deck which it then can pull cards into a hand to check for specific hands such as pairs, three of a kind, etc.

Program Name: Card

This class defines what a card should have. A card has a number value and a suite to which it belongs and thus has two private variables to store the number and suite. The constructor accepts two parameters, a String ‘suite” and integer ‘cardNum’ to be stored in the private variables that define what a card is.

Methods:

- The first method is toString() and accepts no parameters. This method exists so that if the card is numbered 1 it is printed out as Ace, 11 as Jack, 12, as Queen, 13 as King. This is to properly display a card as if it were a real card in a deck.

- The second method is getCardNum() and returns the privately stored card number.

- The third method is getSuite() and returns the privately stored suite of the card.

Program Name: Deck

This class defines what a deck is to be. It stores 52 cards in an ArrayList and of which there are 4 suites and 13 cards of each Suite. The constructor initializes the ArrayList and Random object. The Random object is used to randomly shuffle the cards and put them back in the ArrayList.

Methods:

- The first method is fillDeck() which accepts no parameters. It simply fills the ArrayList with cards to become a deck.

- The second method is shuffle() and accepts no parameters. It randomly picks a card from the deck and then stores it in a temporary ArrayList. When the private ArrayList is empty it is set equal to the temporary ArrayList which stored the random card order. This effectively shuffled the deck.

- The third method is drawCard() and accepts no parameters. It removes the first card in the ArrayList acting as if a card was remove from the top of the deck.

- The fourth method is handStatus() and it accepts no methods. It returns a Boolean that is true if the deck has more than zero cards and false if the deck is empty. It is used to reset the deck if the deck gets too low when running the repeated hand checks.

- The fifth method is getCardNum() which accepts no parameters and returns the number of the card on top of the deck.

- The sixth method is getSuite() which accepts no parameters and returns the suite of the card on top of the deck.

- The seventh method is reset() which empties the deck and then refills the deck in order.

Program Name: HandEvaluator

This program exists to draw five shuffled cards into an ArrayList which acts as a pseudo hand. The hand is then check against most poker hands that exist. The constructor initializes the deck and the ArrayList which acts as the hand.

Methods:

- The first method is drawFive() which accepts no parameters. It shuffles the deck and draws five cards into the hand ArrayList. It checks to see if there are already cards in the hand. If there are, they are emptied. Before drawing if there are no cards in the deck then a statement is printed to reset the deck.

- The second method is checkPair() which accepts no parameters. It checks to see if there exist exactly one pair in the hand. A pair is two cards of different suite with the same number value. It returns a Boolean true if there is a pair and false if there isn’t

- The second method is checkPairInt() which accepts no parameters. It does the same as the method before but is used exclusively for the checkFullHouse() method and returns a int of 1 - 6 or 0. One to six if there are pairs and 0 if there is no pair.

- The third method is checkThreeOfKind() which accepts no parameters. It checks for a three of a kind. This is three cards of different suites with the same number value. It returns a Boolean true if there is three of a kind and false otherwise.

- The fourth method is checkThreeOfKindInt() which accepts no parameters. It does the same as the method before but is used exclusively for the checkFullHouse() method and returns a int of 1 or 0. One if there is a three of a kind and zero if there is no three of a kind.

- The fifth method is checkStraight() which accepts no parameters. It checks for a straight in the hand. A straight is when the cards are sequential in number. When ordered from greatest to smallest they sequentially get smaller by one. The method does this by ordering the cards from greatest to smallest using a group of selection sort methods and then checking if they sequentially get smaller. Returns a Boolean true if it finds a straight and false otherwise.

- The sixth method is checkFullHouse() which accepts no parameters. It checks to see if there is a full house in the hand. A full house is a three of a kind and a pair in the same hand. If the methods checkPairInt() and checkThreeOfKind() return a total of 5. Then a straight exists and a Boolean true is returned. Otherwise, a false is returned

- The seventh method is checkFlush() which accepts no parameters. The method checks to see if there is a flush. A flush is a hand where all of the cards have the same suite. The method does this my storing a counter for each suite. It gets the suite of each card in the hand and adds one to each respective counter for each suite found. If one of the suites totals to five, a Boolean true is returned. If no counter reaches five then false is returned.

- The eight method is checkFourOfKind() which accepts no parameters. The method checks to see if there is a four of a kind. A four of a kind occurs when there are four cards of different suites with the same number value. The method has a similar function to the checkPair() method where it checks one card with the entire hand before checking the next card. Except this method only returns true if the counter is exactly six indicating that a four of a kind has been found which will return Boolean true. Otherwise, a Boolean false is returned.

- The ninth method is runAll() and accepts one parameter, an integer ‘runs’. The parameter ‘runs’ is used to determine how many hands are to be checked for each previously stated hand check method. It enumerates a counter array which stores how many times a check has returned true. After the runs are done the values stored in the array are divided by ‘runs’ to determine the likelihood of each hand given ‘runs’ number of runs. The data is formatted and printed out to the user.

- The tenth method is sort() which accepts an int[] ‘array’. This method is part of a selection sort required for checkStraight(). It takes the ‘array’ and sorts it form greatest to smallest integer.

- The eleventh method is maxPosition() and accepts an int[] ‘a’, and integer ‘from’. It finds the greatest index value in the array starting from ‘from’. It then returns the index with the greatest value. This method is part of a selection sort required for checkStraight().

- The twelfth and final method is swap() which accepts an int[] ‘a’, integer ‘i’, and integer ‘j’. It swaps the array value ‘a[i]’ with array value ‘a[j]’. This method is part of a selection sort required for checkStraight().

Program Name: CardDeckTester

This class is the tester class. It creates an object of HandEvauluator and runs the method runAll() which accepts an int parameter of 10000. This runs the HandEvaluator 10,000.