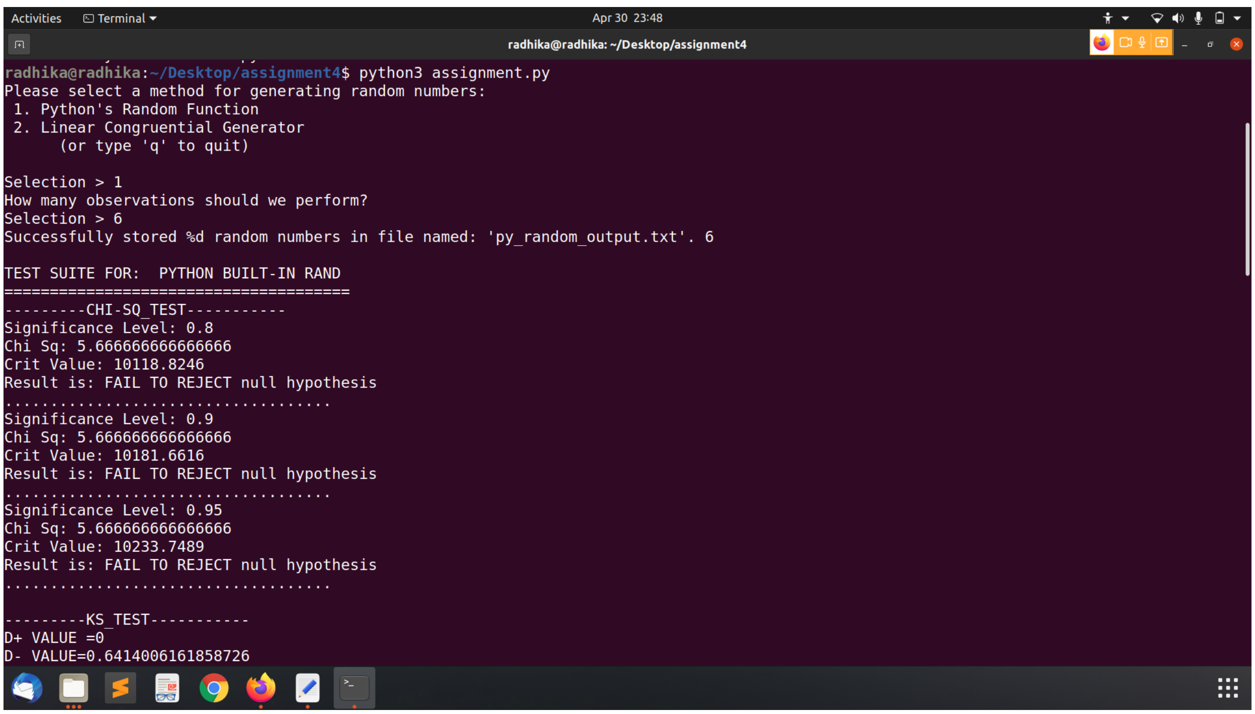
Name : Radhika Singh
Roll No. : 18075046
Dept. : CSE

Assignment 4

Screenshots:



Activities Terminal Apr 30 23:48

radhika@radhika: ~/Desktop/assignment4

```
D VALUE (max): 0.6414006161858726



Alpha Level is: 0.1
D_statistic is: 0.6414006161858726
Critical value is: 0.122
Result is: REJECT null hypothesis
............................
Alpha Level is: 0.05
D_statistic is: 0.6414006161858726
Critical value is: 0.136
Result is: REJECT null hypothesis
............................
Alpha Level is: 0.01
D_statistic is: 0.6414006161858726
Critical value is: 0.16299999999999998
Result is: REJECT null hypothesis
............................
Kolmogorov-Smirnov Test Result for D-Value: 0.6414006161858726

Please select a method for generating random numbers:
 1. Python's Random Function
 2. Linear Congruential Generator
      (or type 'q' to quit)

Selection > 2
How many observations should we perform?
Selection > 3
Successfully stored 3 random numbers in file named: 'lgc_output.txt'.

TEST SUITE FOR:  LINEAR CONGRUENTIAL GENERATOR
======================================
         CHI SQ TEST
```

Activities Terminal Apr 30 23:48

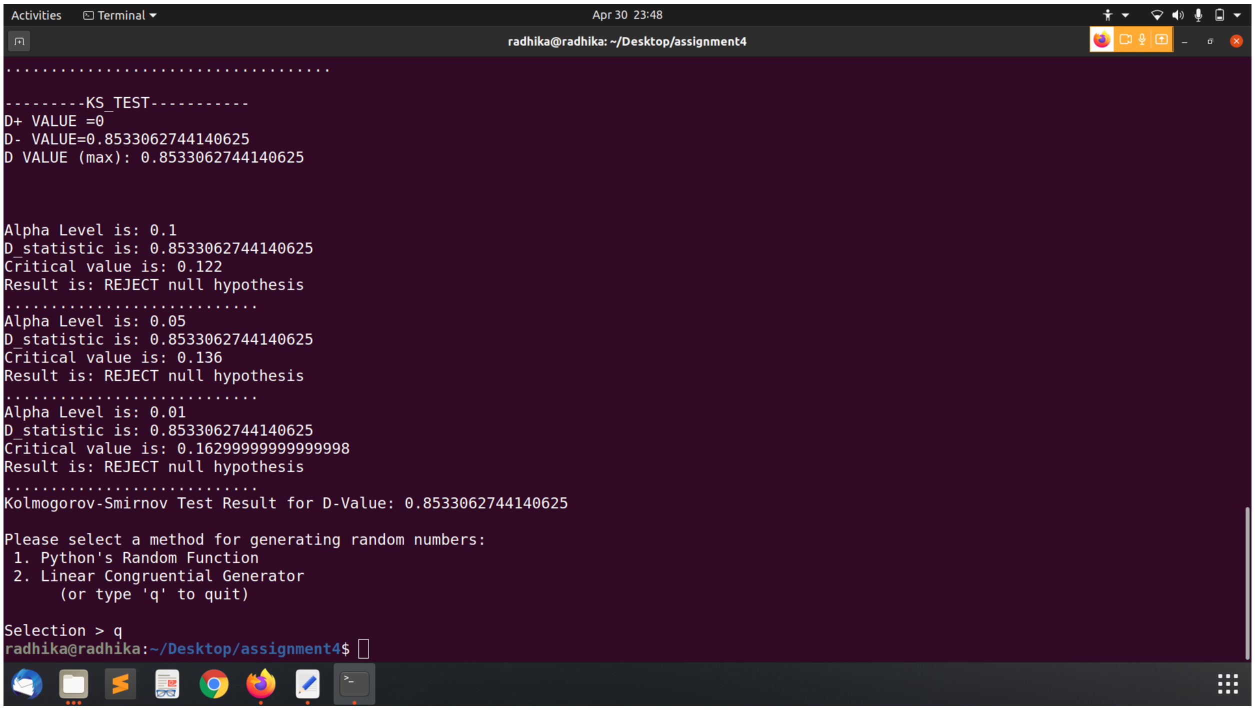
radhika@radhika: ~/Desktop/assignment4

```
======================================
---------CHI-SQ_TEST-----------
Significance Level: 0.8
Chi Sq: 13.666666666666666
Crit Value: 10118.8246
Result is: FAIL TO REJECT null hypothesis
....................................
Significance Level: 0.9
Chi Sq: 13.666666666666666
Crit Value: 10181.6616
Result is: FAIL TO REJECT null hypothesis
....................................
Significance Level: 0.95
Chi Sq: 13.666666666666666
Crit Value: 10233.7489
Result is: FAIL TO REJECT null hypothesis
....................................

---------KS_TEST-----------
D+ VALUE =0
D- VALUE=0.8533062744140625
D VALUE (max): 0.8533062744140625



Alpha Level is: 0.1
D_statistic is: 0.8533062744140625
Critical value is: 0.122
Result is: REJECT null hypothesis
............................
Alpha Level is: 0.05
D_statistic is: 0.8533062744140625
Critical value is: 0.136
Result is: REJECT null hypothesis
```



Source code:

```
# Name - Radhika Singh
# Roll no.- 18075046
# OBJECTIVE:To implement any two PRNGs in a language of my choice.

from itertools import islice
import random as rnd
import numpy as np

"""
PRNGs implemented in this file:
1. Mersenne Twister (PyRand) - python library function
2. Linear Congruential Generator (LCG)
Tests performed:
1. Chi-squared for Uniformity
2. Kolmogorov-Smirnov Test for Uniformity
"""

def main():

        test_selection = ""
        while (test_selection != "q" ):
        select_test()
        test_selection = input("Selection > ").strip()
```

```
        if test_selection == "q":
        exit()

        select_number_of_observations()
        number_observations = input("Selection > ").strip()
        number_observations = int(number_observations)

        # If user selects python rand function,
        if int(test_selection) == 1:
        python_rand( number_observations )
        run_test_suite(test_selection, number_observations)

        # If user selects lfsr function,
        elif int(test_selection)==2:
        generate_lcg(number_observations)
        run_test_suite(test_selection,number_observations)

        else:
        print ("Please select a number from 1 to 2.")


# THREE PRNGS:
#  1.  Standard random number generator in Python(seed=123456789)
#  2.  LCG Implementation(seed=123456789)
#        o Where:  a=101427; c=321, m=(2**16)
#        o Obtain each number in U[0,1) by diving X_i by m


###   PRNG FUNCTIONS   ###

def python_rand( num_iterations ):
        """
        Run the built-in python random number generator and output a number of data points
        specified by the user to a file.
        num_iterations:  The number of data points to write to file
        """
        # Initialize seed value
        x_value = 123456789.0
        rnd.seed(x_value)

        # counter for how many iterations we've run
        counter = 0

        # Open a file for output
```

```
        outFile = open("py_random_output.txt", "w")

        # Perform number of iterations requested by user
        while counter < num_iterations:
        x_value = rnd.random()
        # Write to file
        writeValue = str(x_value)
        outFile.write(writeValue)
        outFile.write("\n")
        counter = counter + 11

        outFile.close()
        print("Successfully stored %d random numbers in file named: 'py_random_output.txt'.",
num_iterations)


def generate_lcg( num_iterations ):
        """
        LCG - generates as many random numbers as requested by user, using a Linear
Congruential Generator
        LCG uses the formula: X_(i+1) = (aX_i + c) mod m
        num_iterations: int - the number of random numbers requested
        """
        # Initialize variables
        x_value = 123456789.0
        a = 101427
        c = 321
        m = (2 ** 16)

        # counter for how many iterations we've run
        counter = 0


        outFile = open("lgc_output.txt", "w")

        #Perfom number of iterations requested by user
        while counter < num_iterations:
        x_value = (a * x_value + c) % m

        #Obtain each number in U[0,1) by diving X_i by m
        writeValue = str(x_value/m)

        # write to output file
        outFile.write(writeValue + "\n")
```

```
        counter = counter+1

        outFile.close()
        print("Successfully stored " + str(num_iterations) + " random numbers in file named:
'lgc_output.txt'.")


#### RANDOMONESS TESTS ####
#### STATS TESTS #####
        # STATISTICAL TESTS
        # Check for uniformity at 80%, 90%, and 95% level. Note that some tests are one-sided,
others two sided
        # x 1. Chi-Square Frequency Test for Uniformity
        #       - Collect 10,000 numbers per generation method
        #       - Sub-divide[0.1) into 10 equal subdivisions
        # x 2. Kolmogorov-Smirnov Test for uniformity
        #       - Since K-S Test works better with a smaller set of numbers, you may use the first
100
        #       out fo the 10,000 that you generated for the Chi-Square Frequency Test

def chi_square_uniformity_test( data_set, confidence_level, num_samples ):
        """
        Null hypothesis:  Our numbers distributed uniformly on the interval [0, 1).
        This function uses the chi-square test for uniformity to determine whether our numbers
        are uniformly distributed on the interval [0,1).
        Formula is: "sum[ (observed-val - expected-val)^2 / expected val ], from 0 to
num_samples"
        This gives us a number which we can test against a chi-square value table.
        :return: A chi-squared value
        """
        chi_sq_value = 0.0
        degrees_of_freedom = num_samples - 1

        # We're doing 10 equal subdivisions, so need to divide our number samples by 10,\
        expected_val = num_samples/10.0

        for observed_val in data_set:
        chi_sq_value += ( pow((expected_val - data_set[observed_val]), 2)/expected_val )

        return chi_sq_value
```

```
def kolmogorov_smirnov_test( data_set, confidence_level, num_samples ):
    """
    Kolmogorov-Smirnov test for uniform distribution of Random numbers
    :data_set: The set of data to analyze. Should be floating point numbers [0,1) in a .txt file
    :confidence_level: with how much confidence should we test?
    :num_samples: number of samples to analyze
    :return: test statistic
    """
    # Step 1:  Rank data from smallest to largest, such that:
    data_set.sort()

    # Step 2: Compute D+ and D-
    # D+ = max(i/N - R(i))
    d_plus = get_d_plus_value_for_KS_TEST(data_set, num_samples)
    print ("D+ VALUE ="+str(d_plus))

    # D- = max(R(i) - (i -1)/n)
    d_minus = get_d_minus_value_for_KS_TEST(data_set, num_samples)
    print ("D- VALUE="+str(d_minus))

    # Step 3:  Computer D = max(D+,D-)
    d_value = max(d_plus, d_minus)
    print ("D VALUE (max): "+str(d_value))

    print("\n\n")
    # Step 4: Determine critical value, using table
    # Step 5: Accept or reject Null hypothesis
    return d_value



##### Significance Tests #####

def chi_sq_significance_test( chi_sq, signif_level):
    """
    Performs a significance test for df=10000, based on values calculated at:
    https://www.swogstat.org/stat/public/chisq_calculator.htm
    :param chi_sq:  Chi-sq value to test
    :param signif_level: Level of significance we are testing: 0.80, 0.90, or 0.95
    :return: message stating whether we accept or reject null
    """
    result = "FAIL TO REJECT null hypothesis"
    crit_value = 0.0
    if signif_level == 0.8:
```

```
        crit_value = 10118.8246
        elif signif_level == 0.90:
        crit_value = 10181.6616
        elif signif_level == 0.95:
        crit_value = 10233.7489
        else:
        print ("**Invalid Significance Level for Chi Sq***")

        if chi_sq > crit_value:
        result = "REJECT null hypothesis"

        print ("Significance Level: " + str(signif_level))
        print ("Chi Sq: " + str(chi_sq))
        print ("Crit Value: " + str(crit_value))
        print ("Result is: " + result)
        print ("....................................")

        return result

def ks_significance_test( d_statistic, num_observations, alpha_level ):
        """
        Perform Significance test for Kolmogorov-Smirnov
        Uses formulas from table A.7:  Discrete-Event System Simulation, by Banks and Carson,
1984
        :param d_statistic: The d-value we are testing
        :param num_observations: The number of observations in our data set
        :param alpha_level: The level of significance we are testing
        :return: result -- accept or reject
        """
        result = "FAIL TO REJECT null hypothesis"
        critical_value = 0


        if alpha_level == 0.1:
        critical_value = 1.22/np.sqrt(num_observations)
        elif alpha_level == 0.05:
        critical_value = 1.36/np.sqrt(num_observations)
        elif alpha_level == 0.01:
        critical_value = 1.63/np.sqrt(num_observations)
        else:
        print ("Invalid alpha level for KS test. Must be: 0.1, 0.05, or 0.01")

        if d_statistic > critical_value:
        result = ("REJECT null hypothesis")
```

```
    print ("Alpha Level is: " + str(alpha_level))
    print ("D_statistic is: " + str(d_statistic))
    print ("Critical value is: " + str(critical_value))
    print ("Result is: " + result)
    print ("...........................")

    return result


#######################
### Helper Methods ####
#######################

def collect_first_100_samples_in_data_set( data_file ):
    """
    Takes a data file, with real number data points between [0,1) reads the first 100 values,
    then adds them to a dictionary as our return value
    :param data_file: A string - the name of the file to read in our current directory
    :return: A dictionary containing the first 100 values as floats
    """

    first_100_vals_as_FLOATS = []
    # grabs first 100 files, as strings with newline endpoints
    with open( data_file, "r" ) as f:
    first_100_vals_as_STRINGS = list(islice(f, 100))

    # transform all values to floats
    for val in first_100_vals_as_STRINGS:
    val = float(val)
    first_100_vals_as_FLOATS.append(val)

    return first_100_vals_as_FLOATS


def divide_RNG_data_into_10_equal_subdivisions_and_count( data_file ):
    """
    Takes a path to a data file in the current directory.
    Returns a dictionary with keys 1-10, values=num instances in each of
    10 equal intervals from range: [0, 1).
    The function counts how many data points are in each interval, and gives us
    a dictionary so we can manipulate this data more easily, based on count by index.
    :param data_file: Must be in current directory. Pass in the string name.
    :return: A dictionary with counts of how many occurrences our data had for each
    of 10 equal intervals between [0, 1). (Divided into 10ths)
```

```
"""
# For each of our uniformity tests, need to divide our data points in 10 equal subdivisions
subdivisions = {  "1":  0,
            "2":  0,
            "3":  0,
            "4":  0,
            "5":  0,
            "6":  0,
            "7":  0,
            "8":  0,
            "9":  0,
            "10": 0   }
with open(data_file, "r") as f:
# data points is a list containing all numbers we've read in.
data_points = f.readlines()

# Loop through our data points and count number of data points in each subdivision
# Divide by tenths, from 0.0 to 1.0.
for num in data_points:
num = float(num)
if num < 0.1:
subdivisions["1"] += 1
elif num < 0.2:
subdivisions["2"] += 1
elif num < 0.3:
subdivisions["3"] += 1
elif num < 0.4:
subdivisions["4"] += 1
elif num < 0.5:
subdivisions["5"] += 1
elif num < 0.6:
subdivisions["6"] += 1
elif num < 0.7:
subdivisions["7"] += 1
elif num < 0.8:
subdivisions["8"] += 1
elif num < 0.9:
subdivisions["9"] += 1
elif num < 1.0:
subdivisions["10"] += 1

return subdivisions
```

```
def get_d_plus_value_for_KS_TEST( data_set, num_samples ):
	"""
	Finds the D+ value for a KS test
	:param data_set: 100 values, must be a list of floats
	:return: the D-+Statistic for our data set
	"""
	# D+ = max(i/N - R(i))
	d_plus_max = 0
	value_rank_i = 1

	# iterate through data set
	for value in data_set:
	# Do each D+ calculation, store it
	d_plus_i_value = ( (value_rank_i/num_samples) - value )

	# Check if it is highest D+ value yet
	if d_plus_i_value > d_plus_max:
	d_plus_max = d_plus_i_value

	# increment our "i" value
	value_rank_i = value_rank_i + 1

	# coming out of this loop, D+ = highest D+ value
	return d_plus_max


def get_d_minus_value_for_KS_TEST( data_set, num_samples ):
	"""
	Finds the D- value for a KS test
	:param data_set: 100 values, must be a list of floats
	:return: the D- Statistic for our data set
	"""
	# D- = max(R(i) - (i -1)/n)
	d_minus_max = 0
	value_rank_i = 1.0

	# iterate through data set
	for value in data_set:
	# Do each D+ calculation, store it
	substraction_value = ( (value_rank_i - 1.0)/num_samples )
	d_minus_i_value = value - substraction_value

	# Check if it is highest D+ value yet
	if d_minus_i_value > d_minus_max:
```

```
        d_minus_max = d_minus_i_value

        # increment our "i" value
        value_rank_i = value_rank_i + 1

        # coming out of this loop, D+ = highest D+ value
        return d_minus_max

def select_test():
        """
        Command line prompt for selecting a test
        :return: void - prints a prompt to command line
        """
        print ("Please select a method for generating random numbers: ")
        print (" 1. Python's Random Function")
        print (" 2. Linear Congruential Generator ")
        print ("            (or type 'q' to quit)")
        print ("")

def select_number_of_observations():
        """
        Command line prompt to select the number of observations for a given test
        :return: void - prints a prompt to command line
        """
        print ("How many observations should we perform?")


def run_test_suite( test_selection, number_observations ):
        """
        Runs all of our test suites and prints output to the screen
        :param test_selection:  an int - 1,2, or 3. Corresponds to test selected.
        :param number_observations: the number of data points to test
        :return: void - prints to command line
        """
        input_file = ""
        test_name = ""
        test_selection = int(test_selection)
        if test_selection == 1:
        input_file = "py_random_output.txt"
        test_name = "PYTHON BUILT-IN RAND"

        elif test_selection == 2:
        input_file = "lgc_output.txt"
        test_name = "LINEAR CONGRUENTIAL GENERATOR"
```

```
    else:
    print ("Invalid input. Please try again.")

    print ("")
    print ("TEST SUITE FOR:  %s " % (test_name))
    print ("=====================================")

    # divide our output values in 10 equal subdivisions and run chi-square test
    print ("---------CHI-SQ_TEST-----------")
    data_points = divide_RNG_data_into_10_equal_subdivisions_and_count(input_file)
    chi_sq_result = chi_square_uniformity_test(data_points, 0, number_observations)
    chi_sq_significance_test( chi_sq_result, 0.8 )
    chi_sq_significance_test( chi_sq_result, 0.9 )
    chi_sq_significance_test( chi_sq_result, 0.95 )

    print ("")

    # get first 100 values from sample and run kolmogorov-smirnov test
    print ("---------KS_TEST-----------")
    first_100_values = collect_first_100_samples_in_data_set(input_file)
    first_100_values.sort()
    ks_result = kolmogorov_smirnov_test(first_100_values,1,100)
    ks_significance_test(ks_result,100, 0.1)
    ks_significance_test(ks_result,100, 0.05)
    ks_significance_test(ks_result,100, 0.01)
    print ("Kolmogorov-Smirnov Test Result for D-Value: " + str(ks_result))
    print ("")

if __name__ == "__main__":
    main()
```

Github link: https://github.com/Radhika-singh/Assignment-4