

Introduction to Web Science

Assignment 7

Prof. Dr. Steffen Staab

staab@uni-koblenz.de

René Pickhardt

rpickhardt@uni-koblenz.de

Korok Sengupta

koroksengupta@uni-koblenz.de

Olga Zagovora

zagovora@uni-koblenz.de

Institute of Web Science and Technologies
Department of Computer Science
University of Koblenz-Landau

Submission until: December 14, 2016, 10:00 a.m.

Tutorial on: December 16, 2016, 12:00 p.m.

Please look at the lessons 1) **Similarity of Text** & 2) **Generative Models**

For all the assignment questions that require you to write code, make sure to include the code in the answer sheet, along with a separate python file. Where screen shots are required, please add them in the answers directly and not as separate files.

Team Name: hotel

Andrea Mildes - mildes@uni-koblenz.de

Sebastian Blei - sblei@uni-koblenz.de

Johannes Kirchner - jkirchner@uni-koblenz.de

Abdul Afghan - abdul.afghan@outlook.de

1 Modelling Text in a Vector Space and calculate similarity (10 points)

Given the following three documents:

D_1 = this is a text about web science

D_2 = web science is covering the analysis of text corpora

D_3 = scientific methods are used to analyze webpages

1.1 Get a feeling for similarity as a human

Without applying any modeling methods just focus on the semantics of each document and decide which two Documents should be most similar. Explain why you have this opinion in a short text using less than 500 characters.

Answer:

D_1 and D_2 should be most similar, because four words which occur in D_1 also occur in D_2 (is, text, web, science). None of the words in D_3 appears in either D_1 or D_2 . Also, D_1 and D_2 are most similar in terms of their meaning, compared to D_3 . Both Documents cover the description of an object ('this document is about ...', 'web science is covering ...'), whereas D_3 covers a subject which can be applied onto other objects.

1.2 Model the documents as vectors and use the cosine similarity

Now recall that we used vector spaces in the lecture in order to model the documents.

1. How many base vectors would be needed to model the documents of this corpus?
2. What does each dimension of the vector space stand for?
3. How many dimensions does the vector space have?
4. Create a table to map words of the documents to the base vectors.
5. Use the notation and formulas from the lecture to represent the documents as document vectors in the word vector space. You can use the term frequency of the words as coefficients. You can / should omit the inverse document frequency.
6. Calculate the cosine similarity between all three pairs of vectors.
7. According to the cosine similarity which 2 documents are most similar according to the constructed model.

Answer:

1. You need one base vector for every unique word in each document. Since there are 19 unique words in D_1, D_2 & D_3 , you need 19 base Vectors.
2. Each dimension stands for the existence of a unique word in a Set of documents.
3. There are as many dimensions as there are base vectors, or rather, as there are unique words.
4. Every base vector is a n-dimensional (in this case 19-dimensional) vector, which has only 0's except in the row the word is assigned to. For example, the word 'this'

would be the first row, so it's base vector would be $\begin{pmatrix} 1 \\ 0 \\ \dots \\ 0 \end{pmatrix}$.

	this	is	a	text	about	web	science	covering	the	analysis	
Vector	$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$		
	Abbreviation	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}

	of	corpora	scientific	methods	are	used	to	analyze	webpages	
Vector	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$		
	Abbreviation	w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{19}

$$\vec{D}_1 = 1 \cdot w_1 + 1 \cdot w_2 + 1 \cdot w_3 + 1 \cdot w_4 + 1 \cdot w_5 + 1 \cdot w_6 + 1 \cdot w_7 =$$

$$\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\vec{D}_2 = 1 \cdot w_2 + 1 \cdot w_4 + 1 \cdot w_6 + 1 \cdot w_7 + 1 \cdot w_8 + 1 \cdot w_9 + 1 \cdot w_{10} + 1 \cdot w_{11} + 1 \cdot w_{12} =$$

$$\begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\vec{D}_3 = 1 \cdot w_{13} + 1 \cdot w_{14} + 1 \cdot w_{15} + 1 \cdot w_{16} + 1 \cdot w_{17} + 1 \cdot w_{18} + 1 \cdot w_{19} =$$

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

$$\begin{aligned} 5. \cos(D_1 D_2) &= \frac{D_1 \cdot D_2}{\|D_1\| \cdot \|D_2\|} \\ &= \frac{1 \cdot 0 + 1 \cdot 1 + 1 \cdot 0 + 1 \cdot 1 + 1 \cdot 0 + 1 \cdot 1 + 1 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 1}{\sqrt{7} \cdot \sqrt{9}} \\ &= \frac{4}{\sqrt{7} \cdot \sqrt{9}} \end{aligned}$$

$$\begin{aligned}\cos(D_2 D_3) &= \frac{D_2 \cdot D_3}{\|D_2\| \cdot \|D_3\|} \\ &= \frac{0}{\sqrt{9} \cdot \sqrt{8}} \\ &= 0\end{aligned}$$

$$\begin{aligned}\cos(D_1 D_3) &= \frac{D_1 \cdot D_3}{\|D_1\| \cdot \|D_3\|} \\ &= \frac{0}{\sqrt{7} \cdot \sqrt{8}} \\ &= 0\end{aligned}$$

$$6. \Theta_{D_1 D_2} = \cos^{-1}\left(\frac{4}{7}\right) = 59,738^\circ$$

$$\Theta_{D_2 D_3} = \cos^{-1}(0) = 90^\circ$$

$$\Theta_{D_1 D_3} = \cos^{-1}(0) = 90^\circ$$

The smaller the angle is, the more similar two documents are. Therefore, D_1 and D_2 are most similar.

1.3 Discussion

Do the results of the model match your expectations from the first subtask? If yes explain why the vector space matches the similarity given from the semantics of the documents. If no explain what the model lacks to take into consideration. Again 500 Words should be enough.

Answer: The results match our expectations because we assumed D_1 and D_2 to be most similar, based off the similar words in the documents. The vector space uses basically the same method by counting which documents have the same words.

2 Building generative models and compare them to the observed data (10 points)

This week we provide you with two probability distributions for characters and spaces which can be found next to the exercise sheet on the WeST website. Also last week we provided you with a dump of Simple English Wikipedia which should be reused this week.

2.1 build a generator

Count the characters and spaces in the Simple English Wikipedia dump. Let the combined number be n . Use the sampling method from the lecture to sample n characters (which could be letters or a space) from each distribution. Store the result for the generated text for each distribution in a file.

2.2 Plot the word rank frequency diagram and CDF

Count the resulting words from the provided data set and from the generated text for each of the probability distributions. Create a word rank frequency diagram which contains all 3 data sets. Also create a CDF plot that contains all three data sets.

2.3 Which generator is closer to the original data?

Let us assume you would want to create a test corpus for some experiments. That test corpus has to have a similar word rank frequency diagram as the original data set. Which of the two generators would you use? You should perform the Kolmogorov Smirnov test as discussed in the lecture by calculating the maximum pointwise distance of the CDFs.

How do your results change when you generate the two text corpora for a second or third time? What will be the values of the Kolmogorov Smirnov test in these cases?

2.4 Hints:

1. Build the cumulative distribution function for the text corpus and the two generated corpora
2. Calculate the maximum pointwise distance on the resulting CDFs
3. You can use `Collections.Counter`, `matplotlib` and `numpy`. You shouldn't need other libs.

Answer: Answer to 2.3 "Which generator is closer to the original data":

The calculated distances are:

Maximum pointwise distance of ZIPF: 0.03700503562224909

Maximum pointwise distance of Uniform: 0.05072979215332735

The ZIPF probability is closer to the original data because the maximum point wise distance is smaller. So if we had to choose between the two generators we would use the ZIPF generator.

When calculating the maximum pointwise distance a second and third time, the results are very similar, but slight changes can occur.

1: Maximum pointwise distance of ZIPF: 0.03700503562224909

Maximum pointwise distance of Uniform: 0.05072979215332735

2: Maximum pointwise distance of ZIPF: 0.03687568360913529

Maximum pointwise distance of Uniform: 0.05073612121006859

3. Maximum pointwise distance of ZIPF: 0.0369625491214863

Maximum pointwise distance of Uniform: 0.05074700111141321

Code:

```
1: # assignment 7 task 2
2: # Andrea Mildes - mildes@uni-koblenz.de
3: # Sebastian Blei - sblei@uni-koblenz.de
4: # Johannes Kirchner - jkirchner@uni-koblenz.de
5: # Abdul Afghan - abdul.afghan@outlook.de
6:
7: # matplotlib to plot the required data
8: import matplotlib.pyplot as plt
9: # logging for debug purposes
10: import logging
11: # random to generate random numbers for the sampling process
12: import random
13: # operator for creating sorted lists from dictionaries
14: import operator
15: # time for debug purposes
16: import time
17: # regex to reliably separate a string into words
18: import re
19:
20: # Configure logging and set start time
21: logging.basicConfig(filename='charCounter.log', level=logging.DEBUG)
22: start_time = 0
23:
24: # Provided probabilities
25: zipf_probabilities = {' ': 0.17840450037213465, '1': 0.004478392057619917, '0': 0.0011831834225755678, '2': 0.0026051307175779174, '5': 0.0011831834225755678, '3': 0.0011831834225755678, '4': 0.0011831834225755678, '6': 0.0011831834225755678, '7': 0.0011831834225755678, '8': 0.0011831834225755678, '9': 0.0011831834225755678}
```



```
27:         '4': 0.0011108979455528355, '7': 0.001079651630435706, '6': 0.
28:         '9': 0.0026071152282516083, '8': 0.0012921888323905763, '_' :
29:         'a': 0.07264712490903191, 'c': 0.02563767289222365, 'b': 0.
30:         'e': 0.09688273452496411, 'd': 0.029857183586861923, 'g': 0.
31:         'f': 0.017232219565845877, 'i': 0.06007894642873102, 'h': 0.
32:         'k': 0.006067466280926215, 'j': 0.0018537015877810488, 'm':
33:         'l': 0.03389465109649857, 'o': 0.05792847618595622, 'n': 0.
34:         'q': 0.0006185966212395744, 'p': 0.016245321110753712, 's':
35:         'r': 0.05221605572640867, 'u': 0.020582942617121572, 't': 0.
36:         'w': 0.013964469813783246, 'v': 0.007927199224676324, 'y': 0.
37:         'x': 0.0014600810295164054, 'z': 0.001048859288348506}
38: uniform_probabilities = {' ': 0.1875, 'a': 0.03125, 'c': 0.03125, 'b': 0.03125, 'e':
39:         'g': 0.03125, 'f': 0.03125, 'i': 0.03125, 'h': 0.03125, 'k':
40:         'm': 0.03125, 'l': 0.03125, 'o': 0.03125, 'n': 0.03125, 'q':
41:         's': 0.03125, 'r': 0.03125, 'u': 0.03125, 't': 0.03125, 'w':
42:         'y': 0.03125, 'x': 0.03125, 'z': 0.03125}
43:
44:
45: # Read the given file into a string
46: def read_file(file):
47:     with open(file) as f:
48:         data = f.read().replace('\n', '')
49:     return data
50:
51:
52: # Write a file
53: def write_file(filename, generated_string):
54:     with open(filename, 'w') as f:
55:         f.write(generated_string)
56:
57:     t = str(round((time.time() - start_time), 2))
58:     logging.info "[" + t + "]" +
59:         "Finished writing \"" + filename + "\" file. \n\n")
60:     return
61:
62:
63: # Iterate through the string and count the characters as well
64: # as the occurrence of each character
65: def count_chars(data):
66:     count = 0
67:     char_dict = {'a': 0, 'b': 0, 'c': 0, 'd': 0, 'e': 0, 'f': 0,
68:         'g': 0, 'h': 0, 'i': 0, 'j': 0, 'k': 0, 'l': 0,
69:         'm': 0, 'n': 0, 'o': 0, 'p': 0, 'q': 0, 'r': 0,
70:         's': 0, 't': 0, 'u': 0, 'v': 0, 'w': 0, 'x': 0,
71:         'y': 0, 'z': 0, ' ': 0, '0': 0, '1': 0, '2': 0,
72:         '3': 0, '4': 0, '5': 0, '6': 0, '7': 0, '8': 0, '9': 0}
73:
74:     for i in range(0, len(data)):
75:         try:
```

```
76:         # Increase count by one
77:         char_dict[data[i].lower()] += 1
78:         count += 1
79:     except KeyError:
80:         # Ignore chars that are not part of the dictionary
81:         continue
82:
83:     t = str(round((time.time() - start_time), 2))
84:     logging.info "[" + t + "]" + "Finished counting the chars. Found "
85:         + str(count) + " chars.\n\n")
86:     return count, char_dict
87:
88:
89: # Calculate the relative values of a given total
90: # and a dictionaries of absolute counts
91: def calculate_percentage(count, char_dict):
92:     rel_char = {}
93:
94:     for key, value in char_dict.items():
95:         rel_char[key] = value / count
96:
97:     return rel_char
98:
99:
100: # Cumulative sum up relative values
101: def cumulative_distribution(rel_char):
102:     c_distri = []
103:     sum_percentage = 0
104:
105:     # Sort the rel_char dictionary by value
106:     sorted_rel_char = sorted(rel_char.items(),
107:                             key=operator.itemgetter(1), reverse=True)
108:
109:     for key, value in sorted_rel_char:
110:         x = sum_percentage + value
111:         c_distri.append((key, x))
112:         sum_percentage = x
113:
114:     return c_distri
115:
116:
117: # Sample the data
118: def sample_data(count, rel_char):
119:     generated_string = ""
120:     c_distri = cumulative_distribution(rel_char)
121:
122:     for i in range(0, count):
123:         r = random.random()
124:         for key, value in c_distri:
```

```
125:         # Find lowest value bigger than r
126:         if r <= value:
127:             generated_string += key
128:             break
129:
130:         # Occasionally log progress
131:         if i % 100000000 == 0:
132:             t = str(round((time.time() - start_time), 2))
133:             logging.info "[" + t + "]" + " " + "Calculated " + str(i) + " chars")
134:
135:         t = str(round((time.time() - start_time), 2))
136:         logging.info "[" + t + "]" + " " + "Finished sampling process\n\n")
137:
138:     return generated_string
139:
140:
141: # Calculate the word rank by counting the occurrence of each word
142: def count_word_rank(s):
143:     regex = re.compile("\w+")
144:     s_list = regex.findall(s)
145:     word_rank = {}
146:
147:     for s in s_list:
148:         try:
149:             # Increase count by one
150:             word_rank[s] += 1
151:         except KeyError:
152:             # Create new dictionary entry for chars that
153:             # are not part of the dict. yet
154:             word_rank[s] = 1
155:
156:     sorted_word_rank = sorted(word_rank.items(),
157:                               key=operator.itemgetter(1), reverse=True)
158:
159:     return sorted_word_rank
160:
161:
162: # Calculate word rank probability as well as the cumulative probability
163: def wr_probability(word_rank):
164:     s = 0
165:     word_rank_probability = []
166:     c = 0
167:     cumulative_word_rank_probability = []
168:
169:     for word in word_rank:
170:         s += word[1]
171:
172:     # Calculate probability
173:     for word in word_rank:
```

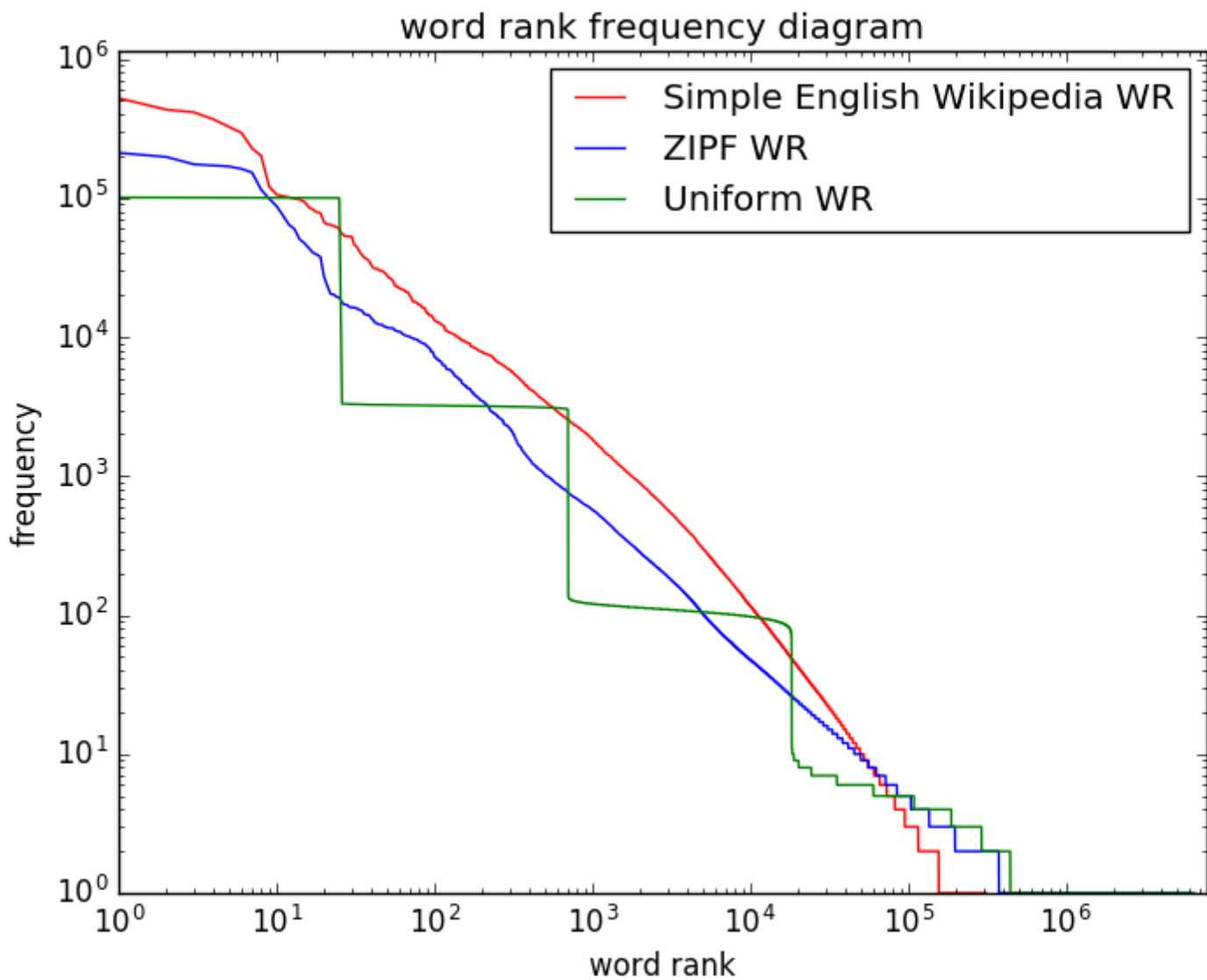
```
174:         word_rank_probability.append((word[0], (word[1]/s)))
175:
176:     # Cumulative probability
177:     for word in word_rank_probability:
178:         c += word[1]
179:         cumulative_word_rank_probability.append((word[0], c))
180:
181:     return word_rank_probability, cumulative_word_rank_probability
182:
183:
184: # Determine the maximum pointwise distance
185: def maximum_pwd(wr_x_probability, wr_data_probability):
186:     max_pwd = 0
187:
188:     for i in range(0, min(len(wr_x_probability), len(wr_data_probability))):
189:         pwd = abs(wr_data_probability[i][1] - wr_x_probability[i][1])
190:         if pwd > max_pwd:
191:             max_pwd = pwd
192:     return max_pwd
193:
194:
195: # Draw the plots
196: def draw_wrf(word_rank_data, word_rank_zipf, word_rank_uniform, slogx):
197:
198:     temp_word_rank_data = []
199:     temp_word_rank_zipf = []
200:     temp_word_rank_uniform = []
201:
202:     for word in word_rank_data:
203:         temp_word_rank_data.append(word[1])
204:
205:     for word in word_rank_zipf:
206:         temp_word_rank_zipf.append(word[1])
207:
208:     for word in word_rank_uniform:
209:         temp_word_rank_uniform.append(word[1])
210:
211:     if not slogx:
212:         plt.loglog(temp_word_rank_data, 'r',
213:                    label='Simple English Wikipedia WR')
214:         plt.loglog(temp_word_rank_zipf, 'b',
215:                    label='ZIPF WR')
216:         plt.loglog(temp_word_rank_uniform, 'g',
217:                    label='Uniform WR')
218:         plt.title("word rank frequency diagram")
219:         plt.xlabel("word rank")
220:         plt.ylabel("frequency")
221:     else:
222:         plt.semilogx(temp_word_rank_data, 'r',
```

```
223:             label='Simple English Wikipedia WR')
224:         plt.semilogx(temp_word_rank_zipf, 'b',
225:             label='ZIPF WR')
226:         plt.semilogx(temp_word_rank_uniform, 'g',
227:             label='Uniform WR')
228:         plt.title("CDF Plot")
229:         plt.xlabel("word rank")
230:         plt.ylabel("cumulative relative frequency")
231:
232:     plt.legend(loc=0)
233:     plt.margins(0.2)
234:     plt.show()
235:
236:
237: def main():
238:     global start_time
239:
240:     # Delete content of log file
241:     with open('charCounter.log', 'w'):
242:         pass
243:
244:     start_time = time.time()
245:
246:     t = str(round((time.time() - start_time), 2))
247:     logging.info "[" + t + "]" Starting :) \n\n"
248:
249:     # Count characters and spaces in the simple english wikipedia
250:     file = "simple-20160801-1-article-per-line"
251:     data = read_file(file)
252:     count, char_dict = count_chars(data)
253:
254:     # Sample the ZIPF distribution and store the result in an file
255:     generated_string_zipf = sample_data(count, zipf_probabilities)
256:     write_file('generated_zipf.txt', generated_string_zipf)
257:
258:     # Sample the Uniform distribution and store the result in an file
259:     generated_string_uniform = sample_data(count, uniform_probabilities)
260:     write_file('generated_uniform.txt', generated_string_uniform)
261:
262:     t = str(round((time.time() - start_time), 2))
263:     logging.info "[" + t + "]" START WORD RANK CALCULATING \n\n"
264:
265:     # Count the resulting words from the provided data set and
266:     # from the generated text
267:     word_rank_data = count_word_rank(data)
268:     t = str(round((time.time() - start_time), 2))
269:     logging.info "[" + t + "]" " +
270:         str(word_rank_data[:30]) + "\n\n"
271:
```

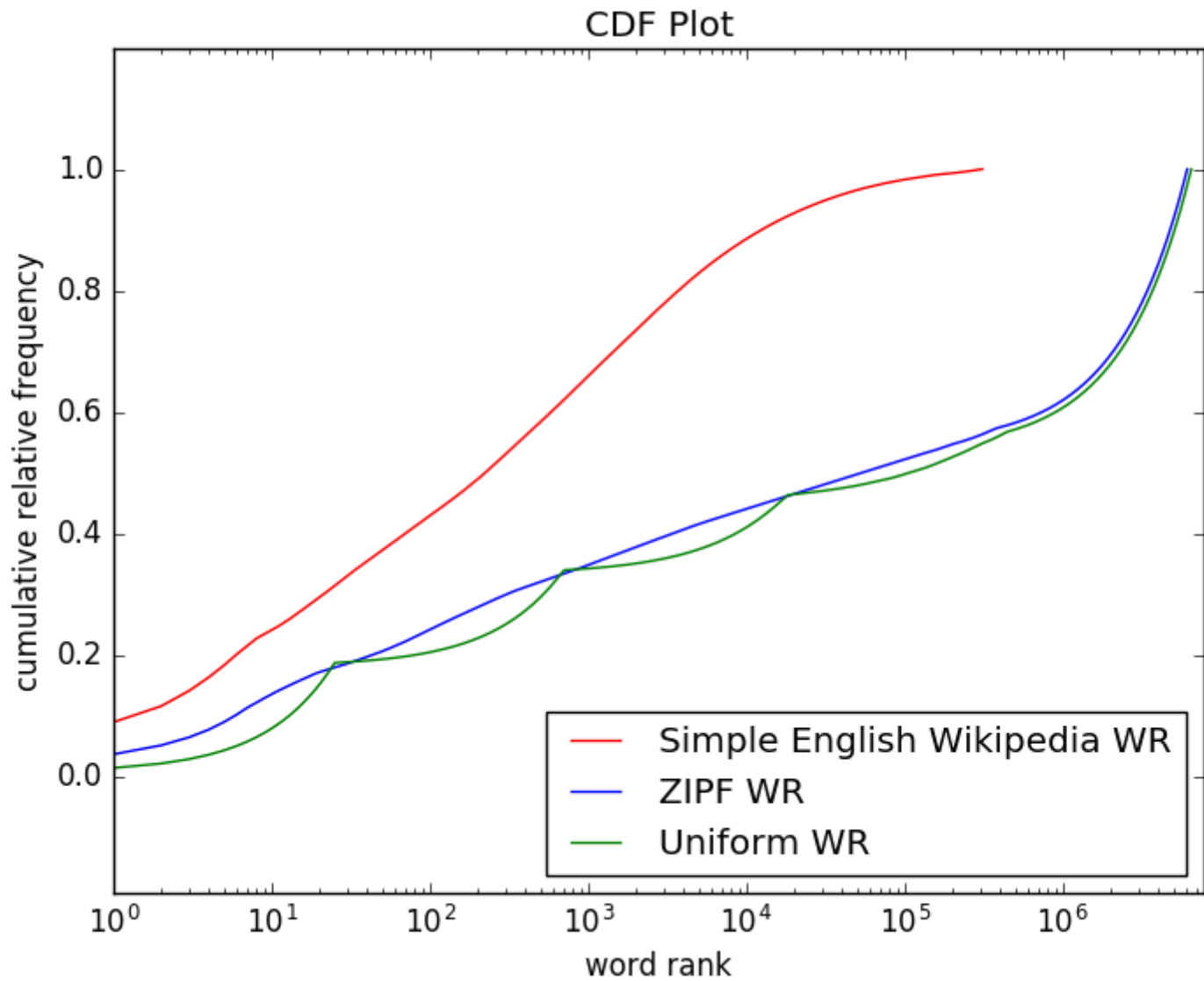
```
272: word_rank_zipf = count_word_rank(generated_string_zipf)
273: t = str(round((time.time() - start_time), 2))
274: logging.info "[" + t + "]" +
275:         str(word_rank_zipf[:30]) + "\n\n")
276:
277: word_rank_uniform = count_word_rank(generated_string_uniform)
278: t = str(round((time.time() - start_time), 2))
279: logging.info "[" + t + "]" +
280:         str(word_rank_uniform[:30]) + "\n\n")
281:
282: # Draw the Word Rank Frequency Diagram
283: t = str(round((time.time() - start_time), 2))
284: logging.info "[" + t + "]" +
285:         "Drawing the Word Rank Frequency Diagram ... \n\n")
286: draw_wrf(word_rank_data, word_rank_zipf, word_rank_uniform, False)
287:
288: # Calculate the word rank and the word probability for each data set
289: t = str(round((time.time() - start_time), 2))
290: logging.info "[" + t + "]" + "Calculating the WR ... \n\n")
291: word_rank_data = count_word_rank(data)
292: wr_data_probability, cwr_data_probability = \
293:     wr_probability(word_rank_data)
294:
295: word_rank_zipf = count_word_rank(generated_string_zipf)
296: wr_zipf_probability, cwr_zipf_probability = \
297:     wr_probability(word_rank_zipf)
298:
299: word_rank_uniform = count_word_rank(generated_string_uniform)
300: wr_uniform_probability, cwr_uniform_probability = \
301:     wr_probability(word_rank_uniform)
302:
303: # Draw the CDF Plot
304: t = str(round((time.time() - start_time), 2))
305: logging.info "[" + t + "]" + "Drawing the CDF Plot ... \n\n")
306: draw_wrf(cwr_data_probability, cwr_zipf_probability,
307:         cwr_uniform_probability, True)
308:
309: # Calculate the maximum pointwise distance
310: t = str(round((time.time() - start_time), 2))
311: logging.info "[" + t + "]" +
312:         "Calculating the maximum pointwise distance ... \n\n")
313: max_pwd_zipf = maximum_pwd(wr_zipf_probability, wr_data_probability)
314: max_pwd_uniform = maximum_pwd(wr_uniform_probability, wr_data_probability)
315:
316: print("Maximum pointwise distance of ZIPF: " + str(max_pwd_zipf))
317: print("Maximum pointwise distance of Uniform: " + str(max_pwd_uniform))
318:
319: t = str(round((time.time() - start_time), 2))
320: logging.info "[" + t + "]" + "FINISHED \n\n")
```

```
321:  
322:  
323: if __name__ == '__main__':  
324:     main()
```

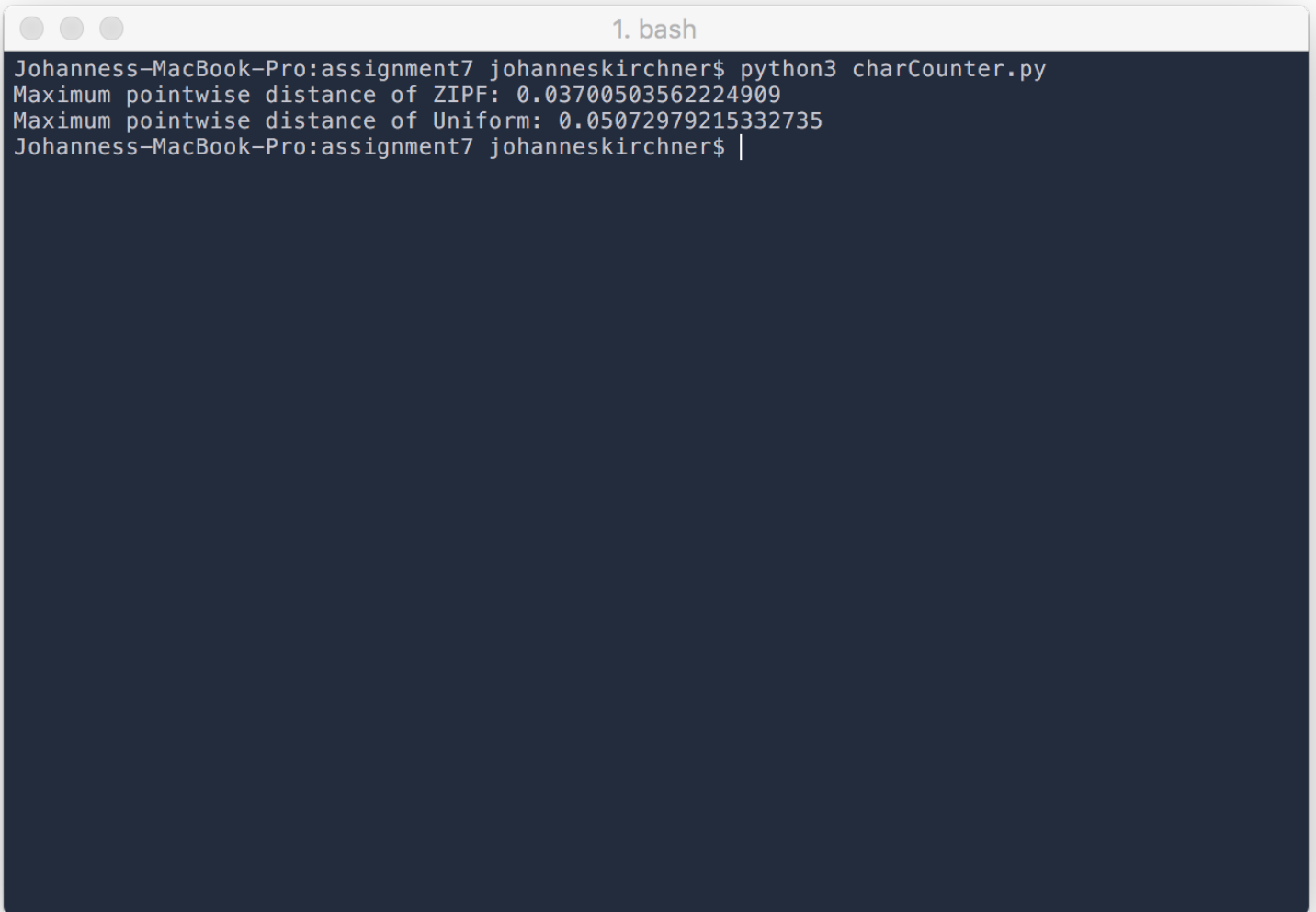
Word Rank Frequency Diagram:



Cumulative Distribution Function Plot:



Terminal output:



```
1. bash
Johanness-MacBook-Pro:assignment7 johanneskirchner$ python3 charCounter.py
Maximum pointwise distance of ZIPF: 0.03700503562224909
Maximum pointwise distance of Uniform: 0.05072979215332735
Johanness-MacBook-Pro:assignment7 johanneskirchner$ |
```

Log Output:

```
1. tail
Johannes-MacBook-Pro:assignment7 johanneskirchner$ tail -f charCounter.log
INFO:root:[0.0] Starting :)

INFO:root:[38.07] Finished counting the chars. Found 91703766 chars.

INFO:root:[38.07] Calculated 0 chars
INFO:root:[46.52] Calculated 10000000 chars
INFO:root:[55.7] Calculated 20000000 chars
INFO:root:[65.15] Calculated 30000000 chars
INFO:root:[74.33] Calculated 40000000 chars
INFO:root:[83.47] Calculated 50000000 chars
INFO:root:[92.52] Calculated 60000000 chars
INFO:root:[101.55] Calculated 70000000 chars
INFO:root:[110.7] Calculated 80000000 chars
INFO:root:[120.03] Calculated 90000000 chars
INFO:root:[121.73] Finished sampling process

INFO:root:[121.9] Finished writing "generated_zipf.txt" file.

INFO:root:[121.9] Calculated 0 chars
INFO:root:[133.12] Calculated 10000000 chars
INFO:root:[144.79] Calculated 20000000 chars
INFO:root:[156.52] Calculated 30000000 chars
INFO:root:[167.93] Calculated 40000000 chars
INFO:root:[179.46] Calculated 50000000 chars
INFO:root:[191.24] Calculated 60000000 chars
INFO:root:[203.09] Calculated 70000000 chars
INFO:root:[214.95] Calculated 80000000 chars
INFO:root:[226.54] Calculated 90000000 chars
```

1. tail

```
INFO:root:[214.95] Calculated 800000000 chars
INFO:root:[226.54] Calculated 900000000 chars
INFO:root:[228.5] Finished sampling process
```

```
INFO:root:[228.66] Finished writing "generated_uniform.txt" file.
```

```
INFO:root:[228.66] START WORD RANK CALCULATING
```

```
INFO:root:[238.84] [('the', 953855), ('of', 523725), ('in', 433112), ('and', 414290), ('a', 368050), ('is', 325266), ('to', 294683), ('was', 228243), ('The', 200841), ('for', 120504), ('on', 106089), ('are', 103150), ('as', 101431), ('that', 99441), ('by', 97500), ('It', 94106), ('with', 86151), ('from', 82800), ('He', 79130), ('s', 78223), ('an', 67047), ('or', 64588), ('it', 64353), ('In', 62824), ('at', 62040), ('his', 59848), ('he', 55253), ('were', 53352), ('also', 53077), ('has', 52991)]
```

```
INFO:root:[253.08] [('e', 282154), ('a', 211695), ('t', 198440), ('i', 175419), ('n', 171091), ('o', 169310), ('s', 162111), ('r', 152283), ('h', 114957), ('l', 98410), ('d', 87627), ('c', 75388), ('m', 64511), ('u', 60115), ('f', 50435), ('p', 47268), ('g', 43652), ('w', 40843), ('b', 39260), ('y', 38134), ('ee', 27409), ('v', 23180), ('ea', 20678), ('ae', 20476), ('te', 19316), ('et', 19304), ('k', 17785), ('ei', 16963), ('ie', 16910), ('ne', 16655)]
```

```
INFO:root:[268.45] [('d', 101436), ('p', 101177), ('m', 101170), ('a', 101119), ('r', 101114), ('w', 101063), ('e', 101054), ('z', 101047), ('g', 101029), ('h', 100923), ('y', 100869), ('o', 100837), ('c', 100827), ('x', 100805), ('n', 100774), ('j', 100731), ('s', 100718), ('f', 100715), ('q', 100619), ('b', 100609), ('t', 100557), ('i', 100491), ('k', 100471), ('v', 100470), ('u', 100329), ('l', 100292), ('zd', 3290), ('ra', 3288), ('su', 3288), ('du', 3287)]
```

1. tail

```
INF0:root:[250.28] [('e', 282903), ('a', 212338), ('t', 197767), ('i', 175469), ('n', 170114), ('o', 169295), ('s', 162204), ('r', 153052), ('h', 114968), ('l', 98678), ('d', 87726), ('c', 74563), ('m', 64626), ('u', 60049), ('f', 50554), ('p', 47447), ('g', 44315), ('w', 40662), ('b', 39162), ('y', 38380), ('ee', 27685), ('v', 22972), ('ea', 20625), ('ae', 20360), ('te', 19450), ('et', 19441), ('k', 17808), ('ei', 17192), ('ie', 16974), ('en', 16469)]
```

```
INF0:root:[264.83] [('d', 101340), ('k', 101142), ('a', 101117), ('h', 100988), ('f', 100935), ('n', 100911), ('i', 100874), ('t', 100849), ('e', 100815), ('b', 100769), ('x', 100741), ('c', 100731), ('j', 100718), ('m', 100668), ('z', 100663), ('q', 100659), ('l', 100624), ('v', 100499), ('p', 100470), ('g', 100453), ('u', 100367), ('w', 100347), ('s', 100336), ('y', 100274), ('o', 100177), ('r', 99957), ('lx', 3322), ('ie', 3303), ('zk', 3291), ('vt', 3291)]
```

```
INF0:root:[264.83] Drawing the Word Rank Frequency Diagram ...
```

```
INF0:root:[289.91] Calculating the WR ...
```

```
INF0:root:[345.51] Drawing the CDF Plot ...
```

```
INF0:root:[429.74] Calculating the maximum pointwise distance ...
```

```
INF0:root:[429.95] FINISHED
```

3 Understanding of the cumulative distribution function (10 points)

Write a fair 6-side die rolling simulator. A fair die is one for which each face appears with equal likelihood. Roll two dice simultaneously n ($=100$) times and record the sum of both dice each time.

1. Plot a readable histogram with frequencies of dice sum outcomes from the simulation.
2. Calculate and plot cumulative distribution function.
3. Answer the following questions using CDF plot:

What is the median sum of two dice sides? Mark the point on the plot.

What is the probability of dice sum to be equal or less than 9? Mark the point on the plot.

4. Repeat the simulation a second time and compute the maximum point-wise distance of both CDFs.
5. Now repeat the simulation (2 times) with $n=1000$ and compute the maximum point-wise distance of both CDFs.
6. What conclusion can you draw from increasing the number of steps in the simulation?

3.1 Hints

1. You can use function from the lecture to calculate rank and normalized cumulative sum for CDF.
2. Do not forget to give proper names of CDF plot axes or maybe even change the ticks values of x-axis.

3.2 Only for nerds and board students (0 Points)

Assuming 20 groups of students. What is the likelihood that at least two groups come up with the same histograms in the case for n ($=100$)?

Answer: Code:

```
1: # assignment 7 task 3
2: # Andrea Mildes - mildes@uni-koblenz.de
3: # Sebastian Blei - sblei@uni-koblenz.de
4: # Johannes Kirchner - jkirchner@uni-koblenz.de
5: # Abdul Afghan - abdul.afghan@outlook.de
6:
7: import numpy as np
```

```
8: import matplotlib.pyplot as plt
9: import random
10: from datetime import datetime
11:
12:
13: # use sytem time (only microseconds) as seed to generate random nubers[1,6]
14: def roll():
15:     dt = datetime.now()
16:     dt = dt.microsecond
17:     tmp1 = 13.1211*np.log(np.sqrt(dt*0.75456))
18:     tmp2 = round(tmp1)
19:     result = tmp2 % 6
20:     return int(result +1)
21:
22: # do the roll n times and store it in a list
23: def listIt(n):
24:     diceList = []
25:     for k in range(n):
26:         k += 1
27:         t1 = roll()
28:         t2 = roll()
29:         t1 = random.randint(1, 6)
30:         t2 = random.randint(1, 6)
31:         sum = t1 + t2
32:         diceList.append(sum)
33:     return diceList
34:
35: # plot histogram with frequencies of dice sum outcomes
36: def simpleHist(rollResults):
37:     plt.hist(rollResults, bins=11, range=(2,12), facecolor='g', alpha=0.75)
38:     plt.title("frequency of dice sum outcomes")
39:     plt.xlabel("Value")
40:     plt.ylabel("Frequency")
41:
42:     plt.show()
43:
44: # cumulative distribution
45: def cumulative(rollResults):
46:     # count how many occurences of each number
47:     c2 = 0
48:     c3 = 0
49:     c4 = 0
50:     c5 = 0
51:     c6 = 0
52:     c7 = 0
53:     c8 = 0
54:     c9 = 0
55:     c10 = 0
56:     c11 = 0
```

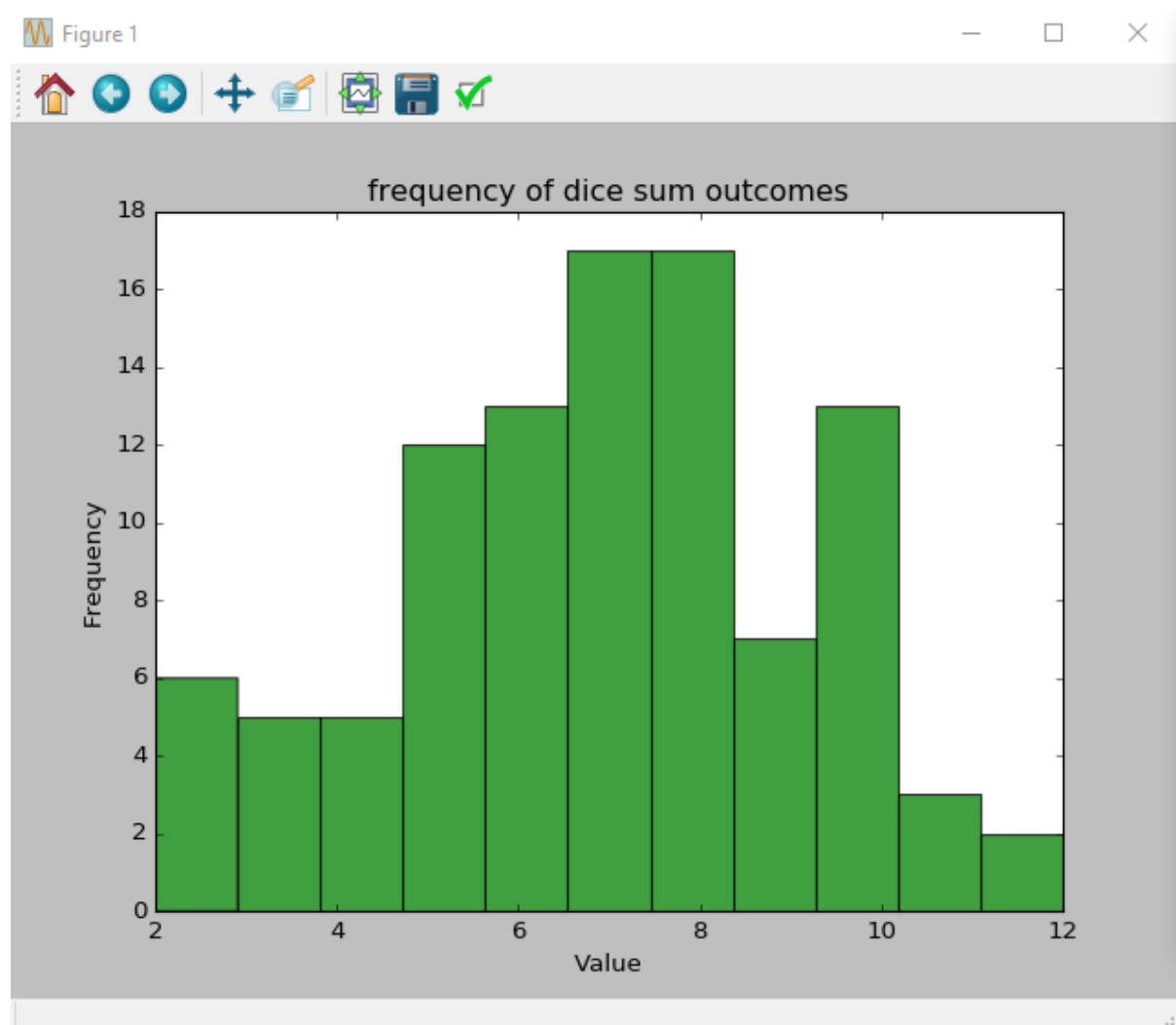
```
57:         c12 = 0
58:
59:     for i in range(len(rollResults)):
60:         if rollResults[i] < 7:
61:             if rollResults[i] < 4:
62:                 if rollResults[i] == 2:
63:                     c2 += 1
64:             else:
65:                 c3 += 1
66:         elif rollResults[i] < 6:
67:             if rollResults[i] == 4:
68:                 c4 += 1
69:             else:
70:                 c5 += 1
71:         else:
72:             c6 += 1
73:     else:
74:         if rollResults[i] < 11:
75:             if rollResults[i] < 9:
76:                 if rollResults[i] == 7:
77:                     c7 += 1
78:             else:
79:                 c8 += 1
80:             elif rollResults[i] == 9:
81:                 c9 += 1
82:             else:
83:                 c10 += 1
84:         elif rollResults[i] == 11:
85:             c11 += 1
86:         else:
87:             c12 += 1
88:
89: #calculate probabilities
90:     prob2 = c2/len(rollResults)
91:     cprob2 = prob2
92:     prob3 = c3/len(rollResults)
93:     cprob3 = cprob2 + prob3
94:     prob4 = c4/len(rollResults)
95:     cprob4 = cprob3 + prob4
96:     prob5 = c5/len(rollResults)
97:     cprob5 = cprob4 + prob5
98:     prob6 = c6/len(rollResults)
99:     cprob6 = cprob5 + prob6
100:    prob7 = c7/len(rollResults)
101:    cprob7 = cprob6 + prob7
102:    prob8 = c8/len(rollResults)
103:    cprob8 = cprob7 + prob8
104:    prob9 = c9/len(rollResults)
105:    cprob9 = cprob8 + prob9
```

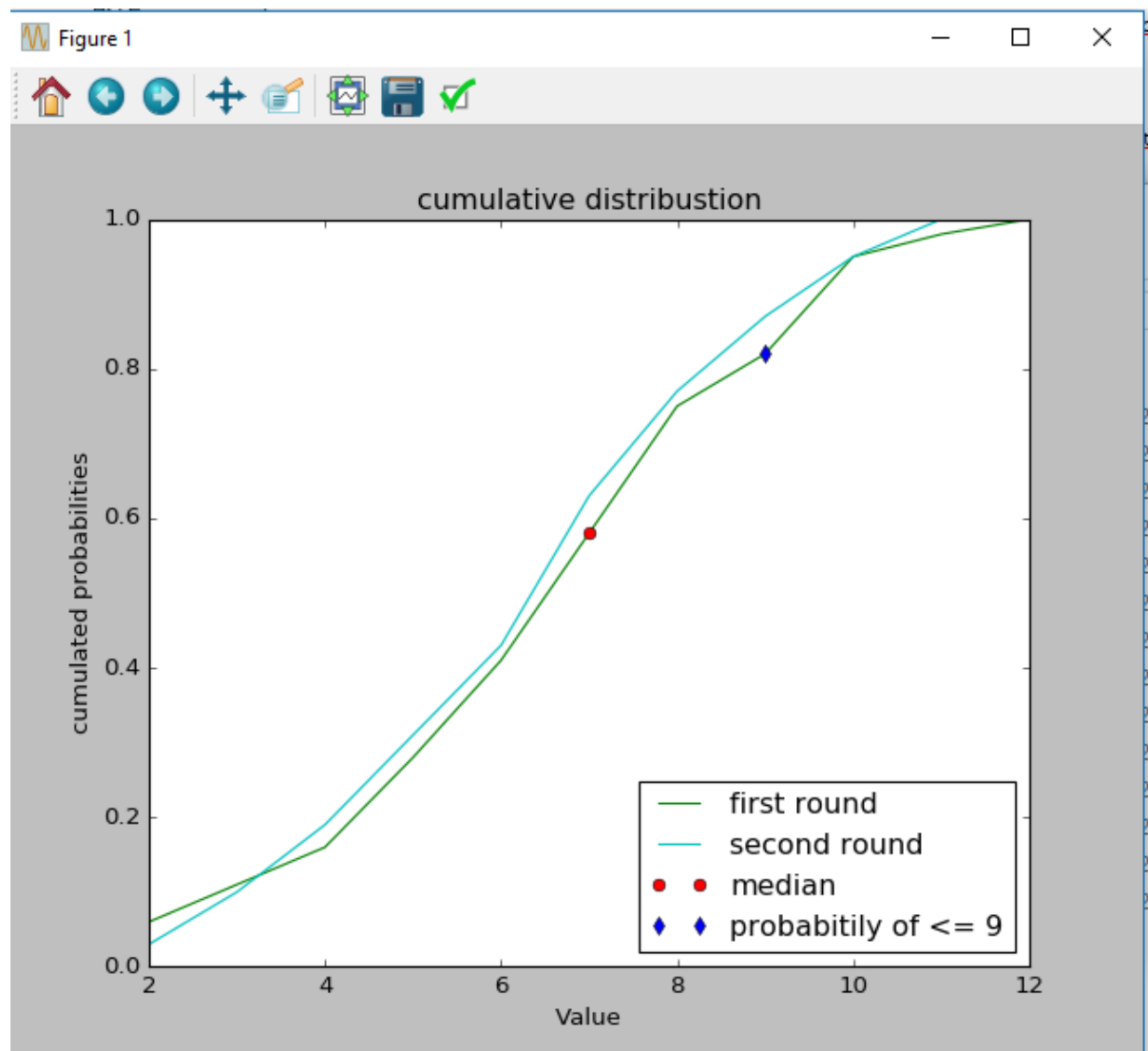
```
106:         prob10 = c10/len(rollResults)
107:         cprob10 = cprob9 + prob10
108:         prob11 = c11/len(rollResults)
109:         cprob11 = cprob10 + prob11
110:         prob12 = c12/len(rollResults)
111:         cprob12 = cprob11 + prob12
112:         cumList = [cprob2, cprob3, cprob4, cprob5, cprob6, cprob7, cprob8, cprob9
113:
114: #probability of dice sum <10
115:         print('Probability of <= 9: ' + str(cprob9))
116:
117: #median
118:         rollResults.sort()
119:         middle = int(np.round(len(rollResults)/2))
120:         medi = rollResults[middle]
121:         print('median: ' + str(medi))
122:
123:         if medi == 2:
124:             ymedi = cporb2
125:         elif medi == 3:
126:             ymedi = cprob3
127:         elif medi == 4:
128:             ymedi = cprob4
129:         elif medi == 5:
130:             ymedi = cprob5
131:         elif medi == 6:
132:             ymedi = cprob6
133:         elif medi == 7:
134:             ymedi = cprob7
135:         elif medi == 8:
136:             ymedi = cprob8
137:         elif medi == 9:
138:             ymedi = cprob9
139:         elif medi == 10:
140:             ymedi = cprob10
141:         elif medi == 11:
142:             ymedi = cprob11
143:         elif medi == 12:
144:             ymedi = cprob12
145:         return cumList, medi, ymedi, cprob9
146:
147:
148:
149: # CDF plot
150: def plotCDF(cumList, medi, ymedi, cprob9):
151:     xlab = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
152:     plt.axis([2, 12, 0, 1])
153:     plt.plot(xlab, cumList)
154:     plt.plot(medi, ymedi, 'ro', label = 'median')
```

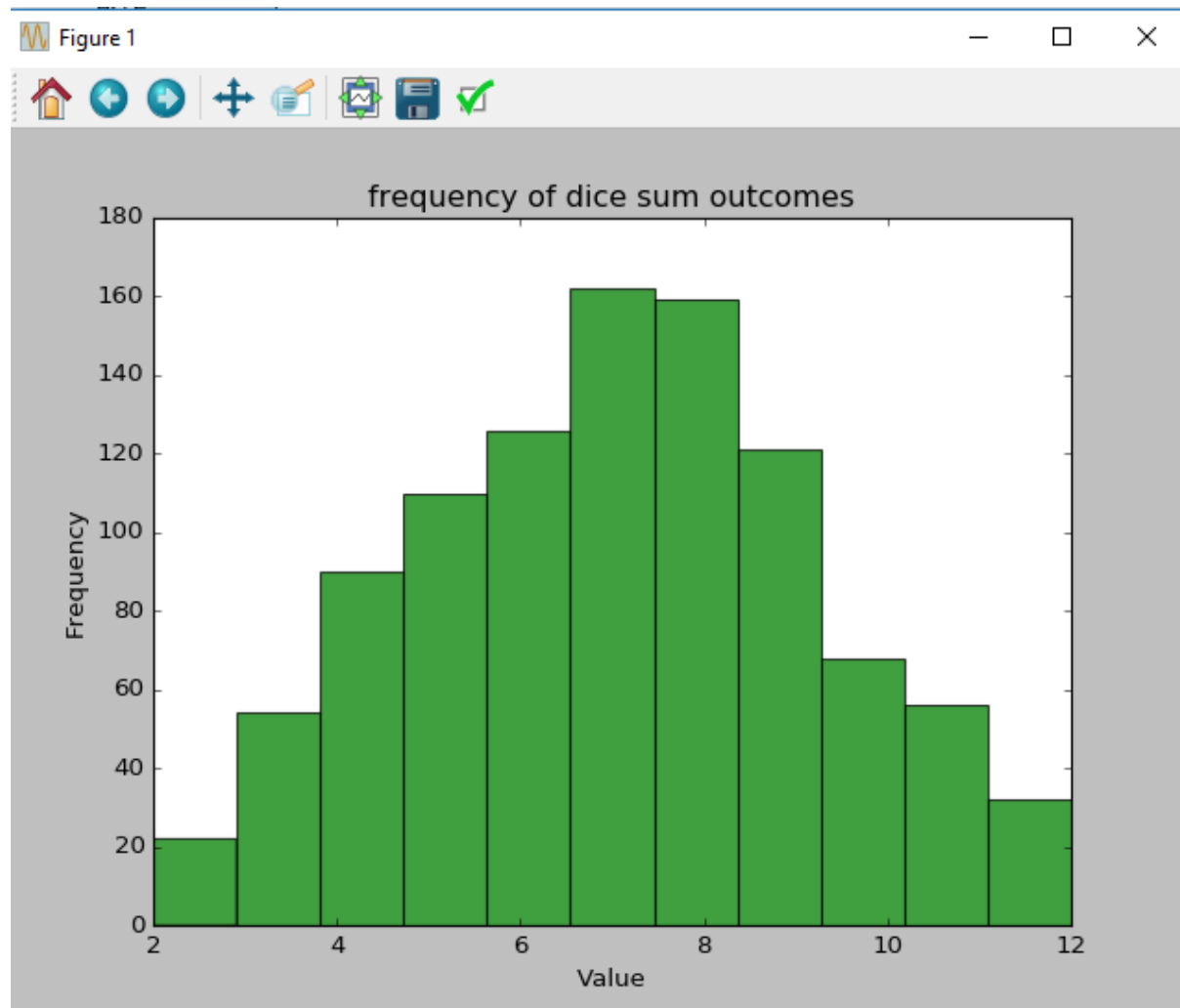


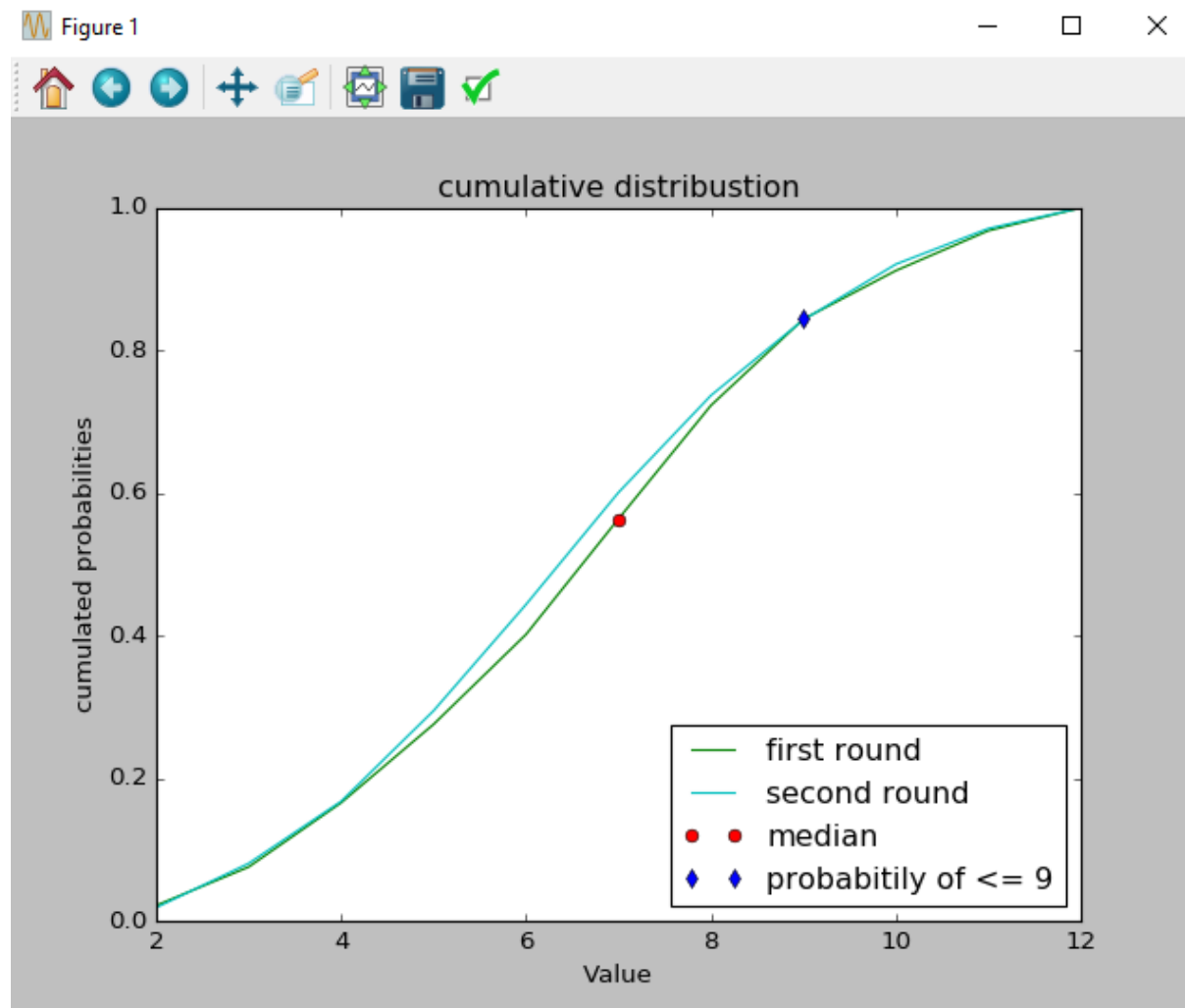
```
155:     plt.plot(9, cprob9, 'gd', label = 'probabilitly of <= 9')
156:     plt.title("cumulative distribustion")
157:     plt.xlabel("Value")
158:     plt.ylabel("cumulated probabilities")
159:     plt.legend(bbox_to_anchor = (1.05, 1), loc = 2, borderaxespad = 0)
160:
161:     plt.show()
162:
163: def plot2CDF(cumList1, cumList2, medi, ymedi, cprob9):
164:     xlab = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
165:     plt.axis([2, 12, 0, 1])
166:     plt.plot(xlab, cumList1, 'g', label = 'first round')
167:     plt.plot(xlab, cumList2, 'c', label = 'second round')
168:     plt.plot(medi, ymedi, 'ro', label = 'median')
169:     plt.plot(9, cprob9, 'bd', label = 'probabilitly of <= 9')
170:     plt.title("cumulative distribustion")
171:     plt.xlabel("Value")
172:     plt.ylabel("cumulated probabilities")
173:     plt.legend(loc = 4)
174:     maxP, pos = maxPointDistance(cumList1, cumList2)
175:     print('maximum point-wise distance = ' + str(maxP))
176:     print('position of maximum point-wise distance = ' + str(pos))
177:     plt.show()
178:
179:
180: # max point-wise distance
181: def maxPointDistance(list1, list2):
182:     maxP = 0
183:     for i in range(len(list1)):
184:         tmp = abs(list1[i] - list2[i])
185:         if tmp > maxP:
186:             maxP = tmp
187:             pos = i + 2
188:     return maxP, pos
189:
190:
191: def main():
192:     # n = 100
193:     firstRound = []
194:     firstRound = listIt(100)
195:     simpleHist(firstRound)
196:     a, b, c, d = cumulative(firstRound)
197:     secondRound = []
198:     secondRound = listIt(100)
199:     e, f, g, h = cumulative(secondRound)
200:     plot2CDF(a, e, b, c, d)
201:
202:     # n = 1000
203:     thirdRound = []
```

```
204:     thirdRound = listIt(1000)
205:     simpleHist(thirdRound)
206:     a3, b3, c3, d3 = cumulative(thirdRound)
207:     fourthRound = []
208:     fourthRound = listIt(1000)
209:     e3, f3, g3, h3 = cumulative(fourthRound)
210:     plot2CDF(a3, e3, b3, c3, d3)
211:
212:
213: if __name__ == '__main__':
214:     main()
```









python dice.py

```
E:\milde\Documents\uni\Master\Web Science\Hotel\workspace\A7\draft>python dice.py
Probability of  $\leq 9$ : 0.8200000000000001
median: 7
Probability of  $\leq 9$ : 0.87
median: 7
maximum point-wise distance = 0.04999999999999993
position of maximum point-wise distance = 7
Probability of  $\leq 9$ : 0.844
median: 7
Probability of  $\leq 9$ : 0.843
median: 7
maximum point-wise distance = 0.04199999999999998
position of maximum point-wise distance = 6
```

6.: The more values we use for computations, the more close are the functions. N is antipropotional to the point-wise distance.

Important Notes

Submission

- Solutions have to be checked into the github repository. Use the directory name `groupname/assignment7/` in your group's repository.
- The name of the group and the names of all participating students must be listed on each submission.
- Solution format: all solutions as *one* PDF document. Programming code has to be submitted as Python code to the github repository. Upload *all* `.py` files of your program! Use **UTF-8** as the file encoding. *Other encodings will not be taken into account!*
- Check that your code compiles without errors.
- Make sure your code is formatted to be easy to read.
 - Make sure you code has consistent **indentation**.
 - Make sure you comment and document your code adequately in English.
 - Choose consistent and intuitive names for your identifiers.
- Do *not* use any accents, spaces or special characters in your filenames.

Acknowledgment

This latex template was created by Lukas Schmelzeisen for the tutorials of "Web Information Retrieval".

LA_TE_X

Currently the code can only be build using **LuaLaTeX**, so make sure you have that installed. If on Overleaf, there's an error, go to settings and change the **L**A_TE_Xengine to **LuaLaTeX**.