Sabahattin Mert Daloglu
UFID: 94325940

PHZ 3152: Computational Astrophysics
Project 1: Cryptography

Description of the code architecture:

Input Data Management:
As usual the required libraries, numpy, pandas, math, etc. were imported at first. For this project, the text for “War and Peace” was used to find how often particular letters in the English language follow one another. The text in .npz file was loaded and the value corresponding to ‘text’ key was extracted from the library. The defined variable ‘text’ is a numpy array type (0-dimension) and it’s scalar value was accessed via .item function as well as converting it to a string type object. All the characters in this string object were later changed to lower case and assigned to the ‘transitions’ list. The rank function calculates the Unicode code of characters, relative to Unicode of lowercase a, in the input list. The T array was initialized and assigned to the relative Unicode code of the characters in the ‘transitions’ list. A for loop iterating over the values of T was used to filter out the special characters and append the letters of alphabet to the T_new array. This step finalizes the filtering and ordering of the input data file.

Next step was to create a zero matrix, M, of size 26x26 to initialize the transition matrix for the Markov Chain. A for loop iterating over zip(T_new,T_new[1:]), which maps each character in the text to the following character, was used to fill the transition matrix M. Later, the elements of the matrix M were divided by their corresponding row sum to normalize the matrix M since the row values should add up to 1 for a transition matrix. Now we have the transition matrix in terms of numpy arrays corresponding to the likelihood that one letter is followed by another sequentially. This can be visualized in pandas data frame shown with the variable df.

Generating a key(generate_key) and calculating the P values(P_calculation):
The generate_key function was used to map the letters of alphabet to another randomly generated sequence of letters. This function returns the key dictionary which is the dictionary of the corresponding mapping. A key was generated by running this function with the input of lower-case letters in the alphabet. This key will be our initial guess for the cryptograph and is randomly generated. The following steps were followed to randomly swap the indices of the key to optimize the value of P calculated from the probability matrix M. The P_calculation function calculates the total probability of how well the order of letters in the decrypted phrase matches the English language structure given by the transition matrix. Within this function, the first step was to decrypt our phrase according to the mapping given by the input key. Once the decrypted phrase is created inside the mapped_into array, a for loop was used to iterate over each letter of the decrypted phrase and the letter following. Using the rank function to calculate the relative Unicode code of the indices, one can access the probability of a specific letter following another by slicing the rows and columns of the matrix M. These probabilities for each letter of the decrypted phrase were summed to calculate the total probability P.

Swapping the solution key to optimize P(optimal_swap):
The natural step to proceed for our Markov Chain is to swap two indices in our key that are chosen randomly. The optimal_swap function was created for this purpose with the use of simulated annealing to optimize the finding of the global maximum for P. The inputs for the optimal_swap function is the key whose indices will be swapped, and the tau variable which is the time constant for the simulated annealing. The use of unconventional variable names such as T, T_max, and T_min is because the rate of cooling of materials mimics the computational work to find the highest order state, probability P in our case. Simulated annealing can be used both to find the global minimum and maximum with a slight difference in methods between two finding to be mentioned later. The key point in this function is that swapping of the indices was treated as a system with an exponential cooling schedule (as suggested by Newman in Chapter 10.4). Hence, a while loop was used to proceed swapping during cooling, until the temperature of the system reaches to the minimum temperature. Inside this while loop, the time variable ‘t’ is incremented as well as lowering the temperature, until the system stops changing, or a global maximum is found. The choice of tau variable is, hence, significant to optimize the trade-off between the computational time and the accuracy of the global maximum found. The value 1e4 was found the be ideal for tau which is discussed in the next section. First, probability was calculated with the function P_calculation and assigned it to variable P before the swap. Later, two randomly chosen

keys of the mapping 'key' was used to swap the values. The probability of the decrypted phrase mapped by the swapped key was calculated and assigned to P_new. The next step is to accept or reject this swap. An if statement inside the while loop was used to reject the swap with the Metropolis probability. The difference between finding the global maximum and minimum for a system is distinguished here with a minus sign in front of the deltaP variable. The optimal_swap function returns the number of successful swaps N, the new total probability P_new, and the new key new_key obtained by the optimized number of swaps.

Calculating the fraction of correct letters(accuracy_check):
The last function accuracy_check is used to print out the decrypted phrase, using the mapping provided by the input new_key variable. It is also significant to check the fraction of correct letters at the right position and the accuracy of our algorithm. A for loop is used to iterate over the characters of the real_answer variable, the correct phrase, and compare it with the letters of the obtained decrypted phrase. This fraction is also printed out as a percentage.

Description of the parameters tested and diagnostic plots of the impact of varying each of the parameters

Annealing (with different time constants τ): Simulated annealing is a significant part of the cryptograph algorithm. The total probability P for the solution key was treated as the 'state' of the system. Thus, with the ideas from statistical physics, one can treat the finding of the maximum total probability as the problem of maximizing the state of the system. The catch here is that in statistical physics we would be interested in finding the lowest state of a system while cooling to absolute zero hence the minimum, but this is similar to finding the maximum state by considering the negative of our function. The global maximum of the state of our system, total P calculated from matrix M, hence can be estimated by randomly swapping two indices of the solution key during the process of 'cooling', until the temperature of the system reaches to the minimum temperature. In order to simulate this annealing process of cooling down, variables t, T, T_max and T_min were defined. T_max variable symbolizes the initial temperature of the system whereas T_min symbolizes the lowest temperature at which cooling ends, close to zero. The variable t denotes the time of the system, which is incremented by one after each swapping attempt, determining the current temperature of the system T. The rate of cooling for simulated annealing was chosen to be the exponential cooling schedule expressed in the following equation:

$$T = T_0 e^{-\frac{t}{\tau}}$$

The variables for the annealing were chosen such that the computational calculation does not take more than a minute. This corresponds to values of T_max = 100.0, T_min = 1e-3. If one decides to choose a value much closer to zero for T_min, algorithm will run for much longer which is inconvenient for our purposes. The independent variable chosen for this analysis is the time constant τ, which changes the rate of cooling. One would expect to observe better results with slow cooling, since this tends to give results closer to the global maximum of the system. However, we should also consider the computational intensity this improvement would bring. The choice of tau variable is, hence, significant to optimize the trade-off between the computational time and the accuracy of the global maximum found from the probability matrix M. To test our expectations, the accuracy of the decrypted phrase, rate, was calculated for the values of $\tau = 10^1, 10^{1.2}, 10^{1.4}, 10^{1.6} \ldots 10^{4.2}$. The upper limit was kept at 1e(4.2) since higher orders required longer computational time. The graph in Figure 1 demonstrates that the overall general trend is with higher τ values, the fraction of the correct letters obtained from the phrase, accuracy rate, increases. The zigzag behaviors and the dips at higher τ values naturally arise from the fact that Monte Carlo simulations utilizes random selection for each swap and there might be cases which the algorithm is stuck in the local maximum due to random choices of swaps or very bad initial mapping, key. Nevertheless, the accuracy rate was observed to converge at 25.581% for higher values and thus the choice of τ was kept at 1e4 for the rest of the analysis.

Sabahattin Mert Daloglu
UFID: 94325940

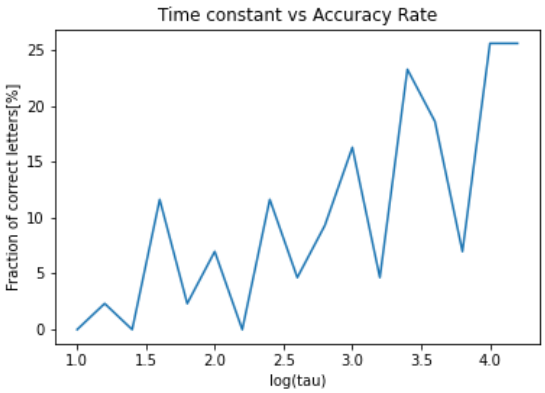

Figure 1: Accuracy rate with different annealing time constants

Only accepting positive proposals:

One of the key elements of the Markov Chain is to accept slightly bad swaps based on some probability so that getting stuck in local maximum or minimum can be avoided. Hence, it is expected that only accepting positive proposals would result in much lower accuracy rate since the optimization of total P would not be able to escape local maximum. To test our hypothesis, only the cases where the new calculated total probability is greater than the previous total probability, were accepted. The previously defined else-if condition to reject negative proposals based on the Metropolis probability was replaced with else condition swapping back the indices whenever P_new > P is not satisfied. Since this analysis does not have a continuous set of independent variable but rather a condition change of accepting positive proposals, the effect was observed on different range of τ values as above. This time only positive swaps were accepted over a range of different τ values. The Figure 2 below demonstrates constant accuracy rate over different range of τ values. Comparing this to the trend of Figure 1, the optimization of τ was not successfully achieved since for all cases there exists a key with high enough accurate mapping rate, around 9.3%, at which all values get stuck at. During the cooling time of our simulated annealing, this local maximum was found for all cases and algorithm was not able to escape since the next swap from this point would mean P_new < P.

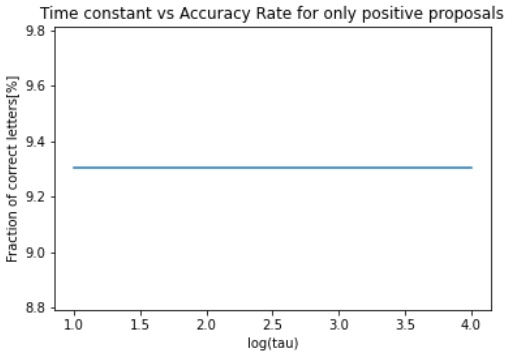

Figure 2: Accuracy rate when only positive swaps are accepted over different annealing time constants

Varying the probability by which negative proposals are accepted:

As seen above, it is crucial to accept negative proposals with some probability to avoid local minimum/maximum. In this section, the probability at which the negative proposals are accepted is varied and its effect is observed. So far, our algorithm decides whether to accept or reject negative proposals, P_new < P, based on the Metropolis probability given as:

Sabahattin Mert Daloglu
UFID: 94325940

$$P_{acceptance} = \begin{cases} 1 & if\ E_j \le E_i\ , \\ e^{-\beta(E_j - E_i)} & if E_j > E_i. \end{cases}$$

This implies that if P_new ≥ P, the swap is accepted always. However, if P_new < P, the acceptance is based on P_new – P. This relation comes from fact that the probability of a state in statistical mechanics is given by the Boltzman formula:

$$P(E_i) = \frac{e^{-\beta E_i}}{Z}, \qquad where\ Z = \sum_i e^{-\beta E_i}$$

In our application of Monte Carlo Markov Chain, the choice of a new state is determined by the set of transition probabilities T_{ij} which are the elements of our transition matrix M. As suggested by Newman in Chapter 10, the acceptance probability of negative proposals was chosen based on the probability of a random number between 0 and 1 to be less than the ratio of two transition probabilities before and after the swap:

$$\frac{T_{ij}}{T_{ji}} = \frac{P(E_j)}{P(E_i)} = \frac{\frac{e^{-\beta E_j}}{Z}}{\frac{e^{-\beta E_i}}{Z}} = e^{-\beta(E_j - E_i)}$$

Applying this to our system we get the expression $e^{-(-\frac{Pnew-P}{T})} = e^{(\frac{\Delta P}{T})}$. The negative sign in front of ΔP is because we are looking for global maximum instead of minimum. We can change this probability at which the negative proposals are accepted by multiplying ΔP with a constant such that the condition of accepting negative proposals would be higher or lower depending on the constant. This constant was given the following values $i = 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}$. This meant that for higher values of i, random.random() < $e^{\frac{i\Delta P}{T}}$ will be less likely since ΔP is negative for negative proposals, and $e^{\frac{i\Delta P}{T}}$ becomes smaller. Hence, for higher values of i, negative proposals are less likely to be accepted whereas for lower values i, negative proposals are more likely to be accepted. The plot in Figure 3 demonstrates that for both extreme cases, when negative proposals are rarely accepted or almost always accepted, the accuracy rate of the cryptograph moves away from the maximum value of 25.58%. For values of $i = 0.1, 1, 10$ the algorithm has the maximum rate of 25.58%. This trend is expected and makes sense since we want a 'sweet spot' of the probability to accept negative proposal just enough that local minima/maxima are avoided. If this probability is too low, then the previous case of accepting only positive proposals is obtained. On the other hand, if the probability is too high, then there would be almost no distinction between positive and negative proposals, neglecting the effect of the probability obtained from the transition matrix M.

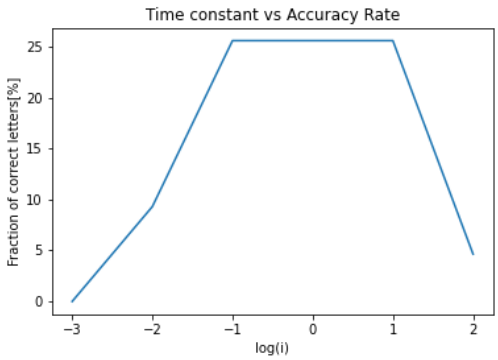

Figure 3: Accuracy rate plotted against log of constants that are multiplied with ΔP

Sabahattin Mert Daloglu
UFID: 94325940

Varying how the probability for the solution is calculated (addition, multiplication, log):

The P_calculation function calculates the total probability of how well the order of letters in the decrypted phrase matches the English language structure given by the transition matrix. This function iterates over each letter of the decrypted phrase and finds the corresponding probability in the matrix M, and adds them to a total value of P. Previously only addition method was used for the calculation of P. In this section, other methods are considered such as multiplying individual probabilities for each letter as well as adding the log of individual probabilities. The plot in Figure 4 illustrates the accuracy rate of different methods used to calculate the total probability P. The addition method gave the highest accuracy of 25.58%. The multiplication method gave 0% accuracy which made sense since the individual probabilities are percentages as small as the order of 1e-3, some are zero, looking at the transition matrix. Multiplying such small numbers for every letter in the phrase would yield a number converging to 0 for the total probability P. This value of P makes it hard to choose between a good swap and a bad swap since both P values for states E_i and E_j converge to 0. On the other hand, taking the logarithm of individual probabilities can be misleading since individual probabilities converging to 0 would give negative infinity as an answer when logarithm is taken. Nevertheless, the addition method of calculating the total probability P for the solution key was found to give the highest accuracy.

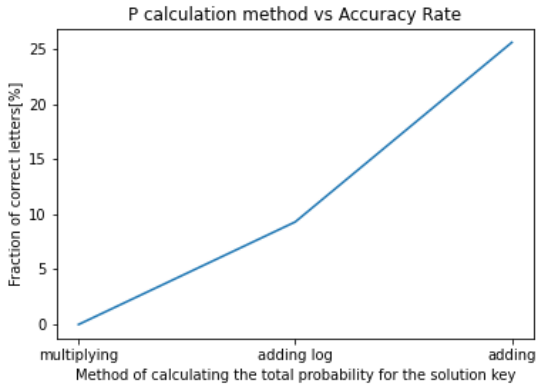

Figure 4: Accuracy rate plotted against the methods of calculating the total probability P