SCHOOL OF ELECTRICAL ENGINEERING, UNIVERSITY OF THE WITSWATERSRAND

ELEN3015 - Data and Information Management

Lecturer: Ling Cheng

Lab 1 Classical Cryptography

16 February 2018

Tyson Cross 1239448

***Abstract*— This report details a series of laboratory exercises demonstrating classical cryptanalysis with basic cryptographic tools using Mathworks' Matlab, to perform frequency analysis on letter distribution in samples of text from English literature. The vulnerabilities of substitution and transposition ciphers are examined and discussed.**

INTRODUCTION

Frequency analysis is a fundamental technique in classical cryptanalysis to reveal patterns or weaknesses in mono-alphabetic encryption using substitution ciphers. Poly-alphabetic encryption can flatten the frequency distribution to help conceal pattern matching techniques from recovering the plaintext trivially. The index of coincidence is a useful parameter for measuring and comparing the probabilities of letter occurrences (frequency distribution variance) in texts.

The sample text (from the Calgary Corpus) is a section of Thomas Hardy's *Far From the Madding Crowd*, comprised of 15760 characters selected in a contiguous block of text, stripped of punctuation and white-space. One of the possible pitfalls of small samples in frequency analysis is the deviation of actual measured character frequencies from the expected frequency distribution of letters in the larger English Language.

EXERCISE 1

Exercise 1 a)

Table I shows the measured frequency distribution of letters in the sample text. Table I was generated from the matlab instructions shown in Code Listing no. 1 in Appendix I.

TABLE I: Calculated letter frequency

Letter	Frequency	Rank	Letter	Frequency	Rank
e	0.1243	1	u	0.02407	14
t	0.08743	2	w	0.02366	15
a	0.08435	3	m	0.02267	16
n	0.07487	4	f	0.02255	17
o	0.07092	5	p	0.02006	18
i	0.06947	6	y	0.01622	19
h	0.06545	7	b	0.01587	20
s	0.06516	8	k	0.007848	21
r	0.05673	9	v	0.006685	22
d	0.04464	10	j	0.001221	23
l	0.04116	11	x	0.001163	24
c	0.02831	12	z	0.0005232	25
g	0.02424	13	q	0.0004069	26

Exercise 1 b)

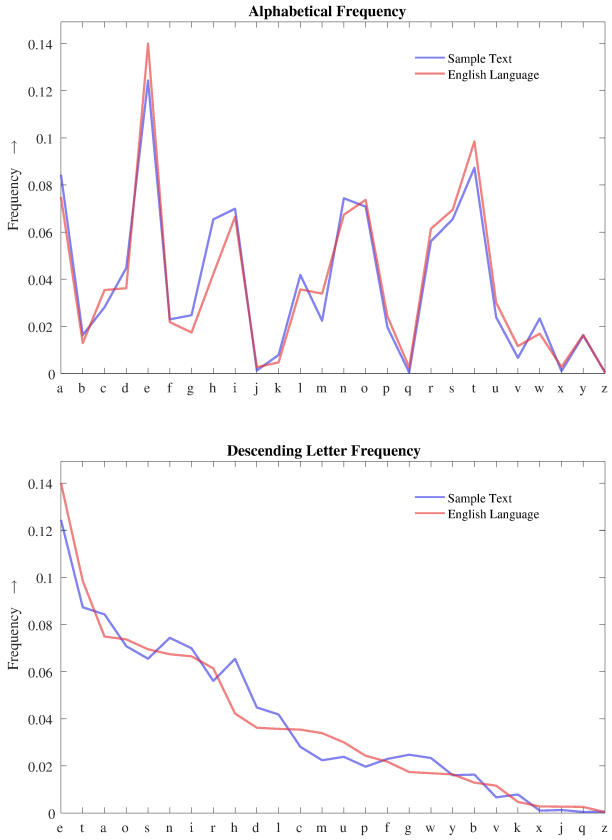

Fig. 1: Letter frequency distribution vs expected values

Figure 1 shows the measured frequency distribution of letters in the sample text, overlaid with the expected distribution, generated from the values included in the `freqmatch` function. These results were found using Code Listing no. 2 in Appendix I.

Exercise 1 c)

The disparity between the measured and expected frequencies of certain letters (such as H,D,U,W and M) makes it likely that these letters will not be correctly matched when using `freqmatch` to decrypt a monoalphabetically encrypted version of the sample text, using the default, "expected" values from the English Language.

TABLE II: Translation table

Original	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Recovered	J	B	Q	A	Y	C	D	E	F	M	R	N	X	V	L	G	O	S	W	Z	H	I	T	U	K	P

Exercise 1 d)

Using the default translation table (shown in Table II) the sample text was encrypted using `enmono`, and then decrypted with the suitable default reverse-translation table. The ciphertext was correctly decrypted, using the instructions shown in Code Listing no. 3 in Appendix I.

However, using the default English Language letter frequency distributions, `freqmatch` could only recover a message that was 45.76% correct using Code 3 in Appendix I. The simplistic matching technique of sorted descending translation tables means that at best individual letters are swapped, or the entire hierarchy of frequencies can be offset by a letter occurring more often in the sampled text than in the wider English language corpus.

Exercise 1 e)

Using a more refined technique such as comparing the translation ciphers by least squares errors and matching by considering a confidence factor that considers minimum error deviation instead of a simple naive ordering (which can offset all future value-matches from a single error in matching individual letter frequencies). Examining the ratios of error between individual letters and the canonical language frequency distributions would likely lead to minimising total translation errors across the text.

Exercise 1 f)

Once the cipher text breakdown process is repeated, using the specific frequency distribution calculated for the actual sample text, the recovery process is 100% correct, with the recovered string matching the original message perfectly. The code confirming this can be seen in Code Listing no. 4 in Appendix I.

EXERCISE 2

Using the `enpoly` command to encrypt the sample text with polyalphabetic Vigenere substitution of varying key lengths, the index of coincidence values shown in Table III was generated using the matlab instructions shown in Code Listing no. 5 in Appendix I. Compared to the provided Lab IC values, the calculated values on the sample text are very close, as is expected. The increase in key length flattens the frequency distribution further, up until a maximally flat distribution of the 26 letters as an input key (where the IC is close to 0.038 in both provided and calculated values). With a full key, there is a difference between the measured and given IC values of 0.0048, around 10% margin of difference.

TABLE III: Index of Coincidence table

Alphabet	1	2	3	4	5	10	26
IC	0.066	0.053	0.047	0.044	0.043	0.039	0.038
Ref. IC	0.068	0.052	0.047	0.044	0.044	0.041	0.038
Difference	0.0024	0.00072	0.00021	0.00043	0.0014	0.0016	0.00048

EXERCISE 3

Exercise 3 a)

The code in Code Listing no. 6 Appendix I uses the function `encolumn` to perform columnar transposition on a text of 22 characters, with 7 columns. The plaintext is padded by 'X' to reach the required length to fill all columns (28 characters). The `decolumn` function correctly recovers the original message, with the appended padding. The results of the frequency distribution variance are shown in Table IV

TABLE IV: Index of Coincidence results

	# of chars	**IC**
Original	22	0.043
Encolumned	28	0.066
Decolumned	28	0.066

The IC for the original plaintext differs from the encolumned IC, due to the additional padding, that changes the frequency distribution because of the letters being added. As expected, the decolumned text has the same IC the encolumned text, as they both have the same letters (only rearranged) resulting in identical frequency distributions. In order to avoid padding, and not modify the frequency distribution between the plaintext and encolumned text, the plaintext length would need to be even, and use a number of columns that is a factor of the plaintext length.

One minor problem with padding the message is that it introduces additional information that will need to be removed, post-decryption, involving additional information of message length or padding character.

A much more serious problem is the resulting increase in the frequency distribution's index of coincidence, which could be an indication of using frequency analysis to isolate padded character. By measuring the distances of this character's occurrences in the text, the column width to decrypt the message can be deduced. To conceal the vulnerability that padding with a single character introduces, the padding character should vary, choosing characters that help flatten the frequency distribution of the overall text.

Exercise 3 b)

Encrypting a 91 character length text with columnar transposition cipher with a width of 7, followed by a second transposition using a width of 13, results in the recovery of the original plaintext. The repeated operations on this specific plaintext using two columnar operation (whose widths are the factors of the plaintext length) results in the transposition of the original matrix of letters, essentially recovering the plaintext, as seen in Figure 2.

```
COMPUTI
NGEQUIP
MENTISU
SELESSW
ITHOUTH
ESOFTWA
RETHATU
SERSEXP
ECTTHER
EAREMAN
YPROGRA
MMINGLA
NGUAGES
```

First (Width 7)

```
CNMSIERSEEYMN
OGEETSEECAPMG
MENLHOTRTRRIU
PQTEOFHSTEONA
UUISUTAEHMGGG
TISSTWTXEARLE
IPUWHAUPRNAAS
```

Second (Width 13)

Fig. 2: Double columnar transposition

To avoid this inadvertent revealing of the original message, one needs to avoid the use of column widths for repeated columnar operations when the widths are factors whose product is the total length (i.e. $7 * 13 = 91$) The code used to test this operation is in Code Listing no. 7 Appendix I.

A general rule to avoid the use of factors would be, to measure the length of the text, and then determine the factors giving that value. The number of `encolumn` operations should differ from the number of factors, and the widths should use values that are different from the factors.

Exercise 3 c)

Using the code shown in Code Listing no. 8 Appendix I, the results shown in Table following results were obtained.

TABLE V: Multiple columnar encipherment

Text length	Column Width	# Operations
20	1	1
20	2	18
20	4	9
20	5	9
20	10	18
20	20	1

Once the text length is an even number (through padding if required) then the message is recoverable through repeated columnar operations with a column width equal to a factor of this text length.

CONCLUSION

Using Matlab code, the vulnerabilities of substitution and transposition ciphers were examined. The frequency distribution and the variance in plain and encrypted texts were discussed as important parameters in cryptanalysis. Vulnerabilities in substitution ciphers can be revealed through frequency analysis of letters with monoalphabetic keys. Calculating the Index of Coincidence to measure the frequency distribution variance in a text can help indicate how vulnerable a text or cipher is to character matching through frequency distribution. Transposition ciphers have a weakness when using columns widths corresponding to a factor of the total message length. Transposition ciphers should aim for maximum diffusion, and not use a number of columns equal to factors of the total length, to avoid trivial recovery of the plaintext through repetition of transposition.

APPENDIX I

EXERCISE 1

```
% ELEN3015
% Tyson Cross
% 1239448

% Formatting and clearing
clear;
clc;
format short;
format loose;

% Read in textfile, stripped all non alphabetical chars, UPPERCASE
text_strip = strip(readfile('sample_text.txt'),1);
% Get the the letter frequency distributions of the text
frequency = freqget(text_strip);
letter=['a':'z'];
% Sort the frequencies in descending letter rank
[sorted_frequency, sortIndex] = sort(frequency,'descend');
sorted_letter = letter(sortIndex);
% Display a table of the results
T = table(sorted_letter', sorted_frequency');
T.Properties.VariableNames ={'Letter','Frequency'};
disp(T)
```

Code 1: lab1_1a.m

```
% ELEN3015
% Tyson Cross
% 1239448

% Formatting and clearing
clear;
clc;
% Read in textfile, stripped all non alphabetical chars, UPPERCASE
text = strip(readfile('sample_text.txt'),1);
% Get the the letter frequency distributions of the text
letter_freq = freqget(text)';
% Canonical values from 'freqmatch' as the English Language distribution
expected_freq = [ 0.0749    0.0129    0.0354    0.0362    0.1400    0.0218    0.0174 ... %  abcdefg
                  0.0422    0.0665    0.0027    0.0047    0.0357    0.0339    0.0674 ... %  hijklmn
                  0.0737    0.0243    0.0026    0.0614    0.0695    0.0985    0.0300 ... %  opqrstu
                  0.0116    0.0169    0.0028    0.0164    0.0004 ]';                     %  vwxyz
% Make a table of the results (alphabetical)
alphabet=['a':'z']';
T = table(alphabet,letter_freq,expected_freq);
T.Properties.VariableNames ={'Letter','Frequency','Expected_Frequency'};
disp(T)

% Sort the results for ordered graph
[sorted_expected_freq, sortIndex] = sort(expected_freq,'descend');
sorted_alphabet = alphabet(sortIndex);
sorted_letter_freq = letter_freq(sortIndex);

%% Display setting and output setup
scr = get(groot,'ScreenSize');                          % screen resolution
ratio = (3/5);
fig1 =  figure('Position',...                           % draw figure
    [1 512 640 880]);
pos = get(fig1,'Position')
set(fig1,'numbertitle','off',...                        % Give figure useful title
    'name','ELEN3015 Lab1: Classical Cryptography',...
    'Color','white');
% set(fig1, 'MenuBar', 'none');                         % Make figure clean
% set(fig1, 'ToolBar', 'none');
% c = listfonts
fontName='CMU Serif';
set(0,'defaultAxesFontName', fontName);                 % Make fonts pretty
set(0,'defaultTextFontName', fontName);
set(groot,'FixedWidthFontName', 'ElroNet Monospace')

%% Plot
% Top
ax1 = subplot(2,1,1);
set(ax1, 'Position', [0.1 0.55 0.85 0.4]);
p1_1 = plot(letter_freq,...
        'Color',[0.18 0.18 0.9 .6],...
```

```
        'LineStyle','-',...
        'LineWidth',2);
hold on
p1_2 = plot(expected_freq,...
        'Color',[0.9 0.18 0.18 .6],...
        'LineStyle','-',...
        'LineWidth',2);
hold on
% Top title
title('Alphabetical Frequency',...
    'FontSize',14,...
    'FontName',fontName);
% Top axes and labels
ylabel('Frequency \rightarrow',...
    'FontName',fontName,...
    'FontSize',14,...
    'Position', [-0.8 .08]);
set(ax1,'xtick',[1:26],...
    'xticklabel',alphabet,...
    'FontSize',14)
axis(ax1,[1 26 0 max(letter_freq*1.2)]);
% Top legend
legend1 = legend({'Sample Text','English Language'});
 set(legend1,...
     'Position',[0.7    0.88    0.1125    0.0403],...
     'Box','off');

% Bottom
ax2 = subplot(2,1,2);
set(ax2, 'Position', [0.1 0.05 0.85 0.4]);
p2_1 = plot(sorted_letter_freq,...
        'Color',[0.18 0.18 0.9 .6],...
        'LineStyle','-',...
        'LineWidth',2);
hold on
p2_2 = plot(sorted_expected_freq,...
        'Color',[0.9 0.18 0.18 .6],...
        'LineStyle','-',...
        'LineWidth',2);
hold on
% Bottom title
title('Descending Letter Frequency',...
    'FontSize',20,...
    'FontName',fontName);
% Bottom axes and labels
ylabel('Frequency \rightarrow',...
    'FontName',fontName,...
    'FontSize',14,...
    'Position', [-0.8 .08]);
set(ax2,'xtick',[1:26],...
    'xticklabel',sorted_alphabet,...
    'FontSize',14)
axis(ax2,[1 26 0 max(letter_freq*1.2)]);
% Bottom legend
legend2 = legend({'Sample Text','English Language'});
 set(legend2,...
     'Position',[0.7    0.38    0.1125    0.0403],...
     'Box','off');
hold off
% export (fix for missing CMU fonts in eps export)
% export_fig Report/letter_frequency.eps
```

Code 2: lab1_1b.m

```
% ELEN3015
% Tyson Cross
% 1239448

% Formatting and clearing
clear all;
clc;
% Read in textfile, stripped all non alphabetical chars, UPPERCASE
message = strip(readfile('sample_text.txt'),1);
% Encrypt plaintext using these default (Caesar Cipher) keys
encrypt_translation_table = 'DEFGHIJKLMNOPQRSTUVWXYZABC';
translation_table = 'XYZABCDEFGHIJKLMNOPQRSTUVW';
encrypted_text = enmono(message);                   % use default key (as above)
decrypted_text = demono(encrypted_text);            % decrypt key with default (reverse key as above)
% Attempt to break the cipher with the default frequency distributions
[recovered_text,recovered_table] = freqmatch(encrypted_text);

% Check results
correct = 0;
```

```
for K=1:length(message)
    if strcmp(message(K),recovered_text(K))
        correct = correct + 1;
    end
end

% Output information based on results
if strcmp(decrypted_text , message)
    disp('The decrypted message matches the original message')
else disp('The decrypted message does NOT match the original message!')
end

if strcmp(recovered_text , message)
    disp('The recovered message matches the original message')
else disp('The recovered message does NOT match the original message!')
    disp(['The recovered message is ', num2str(round(correct/length(message) * 100,2)), '% correct'])
end

% Display results
disp(['Original Table  = ', translation_table])
disp(['Recovered Table = ', recovered_table])
disp(['Index of coincidence = ', num2str(ic(message))])
```

Code 3: lab1_1d.m

```
% ELEN3015
% Tyson Cross
% 1239448

% Formatting and clearing
clear all;
clc;
% Read in textfile, stripped all non alphabetical chars, UPPERCASE
message = strip(readfile('sample_text.txt'),1);
% Encrypt plaintext using (Caesar Cipher) default key4
translation_table = 'XYZABCDEFGHIJKLMNOPQRSTUVW';       % use default key (as above)
encrypted_text = enmono(message);
decrypted_text = demono(encrypted_text);
% Attempt to break the cipher using the calculated frequency
% distribution of the specific plaintext
[recovered_text,recovered_table] = freqmatch(encrypted_text,freqget(message));

% Check results
correct = 0;
for K=1:length(message)
    if strcmp(message(K),recovered_text(K))
        correct = correct + 1;
    end
end

% Output information based on results
if strcmp(decrypted_text , message)
    disp('The decrypted message matches the original message')
else disp('The decrypted message does NOT match the original message!')
end

if strcmp(recovered_text , message)
    disp('The recovered message matches the original message')
else disp('The recovered message does NOT perfectly match the original message!')
    disp(['The recovered message is ', num2str(round(correct/length(message) * 100,2)), '% correct'])
end

% Display results
disp(['Original Table  = ', translation_table])
disp(['Recovered Table = ', recovered_table])
disp(['Index of coincidence = ', num2str(ic(message))])
```

Code 4: lab1_1f.m

EXERCISE 2

```
% ELEN3015
% Tyson Cross
% 1239448

% Formatting and clearing
clear all;
clc;

% Read in textfile, stripped all non alphabetical chars, UPPERCASE
message = strip(readfile('sample_text.txt'),1);
% Generate alphabets and initialise arrays
key = 'a':'z';
number_of_keys = 7;
IC_table = zeros(0,number_of_keys);
difference = zeros(0,number_of_keys);
Given_IC_table = [0.068, 0.052, 0.047, 0.044, 0.044, 0.041, 0.038];

% For alphabets 1:5
for K = 1:5
    IC_table(K) = ic(enpoly(message,key(1:K)));
end
% For alphabet 10
IC_table(6) = ic(enpoly(message,key(1:10)));
% For full alphabet (26)
IC_table(7) = ic(enpoly(message,key(1:26)));

% Calculate difference in values
for K = 1:number_of_keys
    difference(K) = abs(IC_table(K) - Given_IC_table(K));
end

% Display results
alphabet_length = [1:5 10 26]; % Table column labels for the 7 different length keys
T = table (alphabet_length', IC_table', Given_IC_table', difference');
T.Properties.VariableNames ={'Alphabet_Size','Measured_IC','Given_IC','Difference'};
disp(T)
```

Code 5: lab1_2.m

EXERCISE 3

```
% ELEN3015
% Tyson Cross
% 1239448

% Formatting and clearing
clear all;
clc;

% Read in textfile, stripped all non alphabetical chars, UPPERCASE
text = strip(readfile('sample_text_22_chars.txt'),1);
column = encolumn(text,7);
recovered_text = decolumn(column,7);

IC_plaintext = ic(text);
IC_column = ic(column);
IC_decolumn = ic(recovered_text);

% Check results
correct = 0;
for K=1:length(text)
    if strcmp(text(K),recovered_text(K))
        correct = correct + 1;
    end
end

% Output information based on results
if ¬strcmp(recovered_text , text)
    disp('The decolumned text does NOT match the original text!')
else
    disp('The decolumned text matches the original text')
end

if ¬isequal(length(text),length(recovered_text))
    disp('The texts have different lengths')

end
```

```
disp(' ')
disp(['Original:   ', text])
disp(['Encolumned: ', column])
disp(['Decolumned: ', recovered_text])
disp(' ')

% Display results
entries = ['Original  '; 'Encolumned'; 'Decolumned'];
lengths = [length(text), length(column), length(recovered_text)];
ICs = [IC_plaintext, IC_column, IC_decolumn];
T = table (entries, lengths', ICs');
T.Properties.VariableNames = {'Text','Length','IC'};
disp(T)
```

Code 6: lab1_3a.m

```
% ELEN3015
% Tyson Cross
% 1239448

% Formatting and clearing
clear all;
clc;

% Read in textfile, stripped all non alphabetical chars, UPPERCASE
text = strip(readfile('sample_text_91_chars.txt'),1);
column = encolumn(text,7);
double_column = encolumn(column,13);
recovered_text = decolumn(decolumn(double_column,13),7);

IC_plaintext = ic(text);
IC_column = ic(column);
IC_column_double = ic(double_column);
IC_decolumn = ic(recovered_text);

% Check results
correct = 0;
for K=1:length(text)
    if strcmp(text(K),recovered_text(K))
        correct = correct + 1;
    end
end

% Output information based on results
if ¬strcmp(recovered_text , text)
    disp('The decolumned text does NOT match the original text!')
else
    disp('The decolumned text matches the original text')
end

if ¬isequal(length(text),length(recovered_text))
    disp('The texts have different lengths')

end

disp(' ')
disp(['Original:          ', text])
disp(['Encolumned:        ', column])
disp(['Encolumned (again):', double_column])
disp(['Decolumned:        ', recovered_text])
disp(' ')

% Display results
entries = ['Original  '; 'Encolumn2x'; 'Decolumned'];
lengths = [length(text), length(double_column), length(recovered_text)];
ICs = [IC_plaintext, IC_column, IC_decolumn];
T = table (entries, lengths', ICs');
T.Properties.VariableNames = {'Text','Length','IC'};
disp(T)
```

Code 7: lab1_3b.m

```
% ELEN3015
% Tyson Cross
% 1239448

% Formatting and clearing
clear all;
clc;
```

```
% Read in textfile, stripped all non alphabetical chars, UPPERCASE
text = strip(readfile('sample_text_20_chars.txt'),1)
column_width = 5;

% Check results
if ¬mod(length(text),column_width)==0
    disp(['The input text of ', num2str(length(text)),...
        ' chars is not recoverable with multiple columnar encipherments using ',...
        num2str(column_width), ' columns'])
    return
end

column = encolumn(text, column_width);
count = 1;
while ¬strcmp(column , text)
    column = encolumn(column, column_width);
    count = count + 1;
end


% Display results
disp(['It requires ', num2str(count), ' applications of columnar transposition with ',...
    num2str(column_width), ' columns to recover a text of ', num2str(length(text)),' chars.'])
```

Code 8: lab1_3c.m