Analysis and Experimental Study of EMD and GEMD Steganographic Methods

Om Essad Mohammed Lamiles

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Eastern Mediterranean University
May 2016
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Cem Tanova
Acting Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Master of Science in Computer Engineering.

Prof. Dr. Işık Aybay
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Alexander Chefranov
Co-Supervisor

Asst. Prof. Dr.Gürcü Öz
Supervisor

Examining Committee

1. Assoc. Prof. Dr. Alexander Chefranov
2. Asst. Prof. Dr. Adnan Acan
3. Asst. Prof. Dr. Yiltan Bitirim
4. Asst. Prof. Dr. Gürcü Öz
5. Asst. Prof. Dr. Ahmet Ünveren

ABSTRACT

The aim of this thesis is to analyze and experimentally study two steganographic methods: Exploiting Modification Direction (EMD) and Generalized Exploiting Modification Direction (GEMD). In the known experiments conducted on EMD and GEMD, some quality metrics like Peak Signal to Noise Ratio (PSNR), Mean Square Error (MSE), and the embedding capacity Bit Per Pixel (BPP) are discussed, but implementation important details such as the secret image used, data structures, justification of methods, and the optimal cover images number calculation are not provided. Therefore, in this thesis, the implementation of these methods is explained in details such as the input-output data structures, the justification of the methods and the minimum number of cover images computation are given.

The main idea in EMD is that a separate *n*-pixel group of a cover image is used for embedding of each next digit of (2*n*+1) -ary *k*-digit number representation of the next L-bit block from a binary input stream and only one pixel in the *n*-pixel group could be modified by ±1. In GEMD, L-bit blocks, L=n+1, from the input stream are embedded in the next *n*-pixel group, and at least one pixel value in each group could be changed by ±1.

In the implementation, four grayscale 512×512 secret images, and two cover image sizes, 512×512 and 1024×1024, are used. According to our analysis, for the both cover image sizes results, PSNR of EMD is greater than that of GEMD by 0.06%. For MSE, EMD has less MSE than that of GEMD by 0.5%. On the other hand, GEMD is better than EMD in embedding capacity, BPP is greater by 0.33%. GEMD is also better than EMD in memory and time consumption by 0.006% and 0.06% respectively for the 512x512 cover image size, while for the second size, 1024x1024 by 0.004% and 0.13% respectively. In addition, where each method is compared with different cover image sizes, for both methods, greater cover image size has less time consumption by 0.33% for EMD and by 0.38% for GEMD. For memory consumption, using grater size required more memory for both methods, by 0.02%. For comparison with known experiments with 512×512 cover size, we got the practically the same values for PSNR, MSE, and BPP.

Keywords: Steganography, EMD algorithm, GEMD algorithm, (2*n*+1)-ary number, Peak Signal to Noise Ratio (PSNR), Mean Square Error (MSE), Embedding Capacity, Bit Per Pixel (BPP), memory consumption, time consumption.

ÖZ

Bu tezin amacı, Exploiting Modification Direction (EMD) ve Generalized Exploiting Modification Direction (GEMD) steganographik yöntemlerini analiz etmek ve deneysel olarak incelemektir. Literatürde, EMD ve GEMD yöntemlerinin incelendiği referans çalışmada, Peak Signal to Noise Ratio (PSNR), Mean Square Error(MSE) ve gömme kapasitesi Pit Per Pixel (BPP) gibi kalite ölçütleri tartışılmıştır, ancak, kullanılan gizli görüntüler, veri yapıları, yöntemlerin ıspatı ve kaplama resimlerinin optimal sayısının hesaplanması gibi bazı detayları verilmemiştir. Bu nedenle, bu tezde, giriş/çıkış veri yapıları, belirtilen yöntemlerin ıspatı ve kaplama resimlerinin optimal sayısının hesaplanması gibi uygulama detayları açıklanmıştır.

EMD yönteminin ana fikri, her biri (2*n*+1)-ary k-digit sayı ile belirtilen ikili giriş akışının her bir *L*-bit bloğunun, *n*-piksel gruptan oluşan kaplama görüntüsüne, her n-piksel grupta sadece bir pikselinin ±1 olacak şekilde gömülmesidir. GEMD'de ise ikili giriş akışının her bir L-bloğu, *L*=*n*+1, kaplama görüntüsündeki her bir n-piksel gruba, her grupta en az bir piksel degerinin ±1 olacak şekilde gömülmesidir.

Algoritmaların uygulamasında gizli görüntü olarak 512x512 boyutlu, gri tonlu dört farklı görüntü, kaplama görüntüsü boyutu olarak da 512x512 ve 1024x1024 boyutları kullanılmıştır. Deney sonuçlarının analizlerine gore, her iki boyutlu kaplama görüntüsü için, EMD PSNR değeri GEMD PSNR değerinden %0.06 daha büyüktür. Buna ek olarak, EMD MSE değeri GEMD MSE değerinden %0.5 daha azdır. Öte yandan, GEMD BPP değeri, EMD BPP değerine göre %0.33 daha iyidir. Buna ek olarak GEMD'nin bellek ve zaman tüketimi değerleri EMD'ye göre 512x512 kaplama görüntü boyutu için sırasıyla %0.006 ve %0.06 daha iyi olup 1024x1024 için sırasıyla %0.004 ve %0.13 daha iyidir. Bunlara ek olarak, her metod iki farklı kaplama görüntü boyutuna göre karşılaştırıldığında, her iki metod için de büyük kaplama boyutunda zaman tüketimi EMD'de %0.33 GEMD de ise %0.38 daha azdır. Bellek tüketiminde ise büyük boyutlu görüntü kullanımı her iki yöntemde de %0.02'lik bir artış göstermiştir. İki yöntemin 512x512 kaplama boyutu için PSNR, MSE ve BPP ölçü değerleri referans çalışmadaki deney sonuçları ile karşılaştırıldığında tamamen aynı sonuçların elde edildiği görülmüştür.

Anahtar Kelimeler: Steganografi, EMD algoritması, GEMD algoritması, (2*n*+1)-ary sayı, Tepe Sinyal Gürültü Oranı (PSNR), Kare Ortalama Hatası (MSE), Gömülüm kapasitesi, Piksel Başına Bit (BPP), bellek tüketimi, zaman tüketimi

DEDICATION

I dedicate my dissertation work to my family, and a special thank to my loving parents for their love and support throughout my life. I also dedicate this work and give special thanks to my husband for his support, encouragement, and contribution to the success of my life.

ACKNOWLEDGMENT

Foremost, I would like to sincerely thank my supervisors Assoc. Prof. Dr. Alexander Chefranov, and Asst. Prof. Dr.Gürcü Öz for their guidance and support throughout this study. Also I have to thank my department chairman Prof. Dr. Işık Aybay for providing the facilities to students. I am also grateful to all of the department faculty members for their help and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

BPP	Bit Per Pixel
EMD	Exploiting Modification Direction
GEMD	Generalized Exploiting Modification Direction
LSB	Least Significant Bit
MSE	Mean Square Error
PSNR	Peak Signal to Noise Ratio

Chapter 1

INTRODUCTION

Steganography is a technique used to protect messages from unauthorized access, by embedding data into other media forms such as text, image, video, sound, etc., where the hidden data likely will not be detected [1]. Two main directions in steganography are hiding secret data in spatial domain and in frequency domain [1]. The last direction uses digital cosine transformations that is more time consuming compared with the spatial domain methods but provides more security. In steganography, image file is the most common media form used because the human visual system is not sensitive to small variation in colors. Furthermore, they could be easily used as cover media without any doubt as they are commonly used on the Internet [2].

We consider here spatial domain methods. There are many steganographic schemes based on direct replacement like Least Significant Bit (LSB) [3] [4] or based on indirect replacement such as Exploiting Modification Direction (EMD) [7] [13] [14], and Generalized Exploiting Modification Direction (GEMD) [5] [6] [7] schemes; the latter ones will be discussed in this thesis in details which similar to frequency domain methods provide greater security by the use of data transformations but in the spatial domain. The known experiments conducted on EMD and GEMD, and resulting quality metrics like Peak Signal to Noise Ratio (PSNR), Mean Square Error (MSE), and the embedding capacity Bit Per Pixel (BPP) are discussed in [7], but they do not provide sufficient information for their implementation such as the secret image used, data structure, justification of methods, and the number of cover images for one secret image embedding. Therefore, in this thesis data structures and the implementation of these methods are explained in details such as the input-output data structures, the justifications of the methods are provided. A major characteristic of the EMD method is that it uses a separate n-pixel group of a cover image to embed the next digit of (2*n*+1)-ary *k*-digit number representing the next L-bit block from the secret image input and only one pixel in the group can be changed by ±1. In GEMD scheme the next L-bit block, L=*n*+1, is hidden in the next *n*-pixel group, and more than one pixel value in a group may be changed by ±1, so the image quality for it may be lower than that for EMD.

Experiments are conducted with four secret images for different *n* values, and for two different sizes of cover image 512×512 and 1024×1024. The comparison between both methods in case of using two different sizes are taken as the average for each metric, because the comparison for each metric over *n,* as it is done in [7], is not valid since *n* has different meaning for each method, in EMD means the number of pixels required to embed one digit while in GEMD means the number of pixels required to embed one block. According to our analysis, for size 512×512 comparison results, EMD stego image quality PSNR is better than that of GEMD by 0.06%, and also for MSE, EMD is better than that of GEMD by 0.5%. On the other hand, GEMD is better than EMD in embedding capacity, BPP is greater by 0.33%, and in memory and time consumption by 0.006% and 0.06% respectively.

For 1024× 1024 cover size, the results for metrics PSNR, MSE, and BPP are the same of the size 512×512, but for time and memory consumption both methods take less time and more memory consumption. GEMD is better than EMD in memory and time consumption by 0.004% and 0.13% respectively.

In addition, the comparison results using two sizes are taken for each method separately, since as we use grater cover size then we have less time consumption by 0.33% for EMD and 0.38% for GEMD. For memory consumption, using grater size required more memory for the both methods, by 0.02%. For comparison with known experiments with 512×512 cover size, we got the practically the same values for PSNR, MSE, and BPP.

The rest of the thesis is organized as follows. Chapter 2 presents the related work, the experiments on EMD and GEMD, and problem definition. Data structures for EMD and GEMD implementation discussed with details and justification for the both methods are given in Chapter 3. Chapter 4 introduces the implementation of EMD and GEMD algorithms. Chapter 5 shows the experimental results and their comparison versus the known experiments. Finally, Chapter 6 concludes the thesis and discusses the future work.

Chapter 2

RELATED WORK AND PROBLEM DEFINITION

2.1 Overview of Steganography

Steganography is a word of two syllables, its origin came from the Greek language, the first syllable "stegano" means the "covered" or the "secret", while the second one, "graphy", means the "drawing" or the "writing" ; this word is used nowadays for a technique of information hiding. Such technique was used in Greece since the 5th century BC, where the people used it for hiding information on their slave's head [1]. First, a slave is chosen, then, his head is shaved, and a message is written on his head. They waited till the slave's hair grew to make sure that the message is hidden. Then the slave is sent to another place with the message on his head, where his head is then shaved again to get the confidential message. At the same time in Greece, steganography technique was used by Spartans against their enemy Xerxes. The secret message was written on a wood wax tablet and covered to form a new plane layer of wax and due to this wax looked like a blank. Steganography technique was used in the World War II to hide the secret information written on a paper using invisible ink: the paper looks like blank to any person in natural light. Where the organic compounds are the simplest examples of invisible ink which turn dark when held invisible ink a flame, such as lemon juice, milk, or urine. Finally, information was retrieved by using liquids such as water, fruit juices or vinegar. When the wet paper in the liquid was heated, the paper became dark and the message written on it using invisible ink becomes visible and readable.

2.2 Categories of Steganography

There are many types of techniques used for steganography; they could be divided mainly into three groups as follows [3]:

2.2.1 Text Steganography

The hiding information in a text is one of the preferred methods of steganography. In this type of steganography, there are many techniques used, such as extra white space method, by appending extra white space between words or at the end of lines and paragraphs.

2.2.2 Protocol Steganography

The technique used for embedding data within a message which used in the network transmission is called "the protocol steganography". Hiding of data in the header of a TCP/IP where some fields or places are either optional or never used is an example.

2.2.3 Audio/Video/Image Steganography

A secret message is hidden in an audio/video/image file. The binary sequence of audio/video/image file is a bit differing from the main file which is hard to be detected by the normal human eyes. In Audio/Video/ Image Steganography the most generally used is Least Significant Bit (LSB) method, where the Least Significant Bit of each pixel of cover file is replaced with the binary data of secret message stream, so the changes that are made in least significant bit are too small to be detected by human eyes.

2.3 Related Work

We briefly survey spatial domain methods [1]. Steganographic algorithms are quite so many; each one has its own security and complexity, since the main aim for all of them is to embed large amount of secret data with less effects on the cover file, it means more embedding capacity Bit Per Pixel (BPP) with good image quality. One of the most common techniques is the LSB replacement method, where it is simple, fast, and has good stego image quality [2]. In this method the binary secret image is divided into blocks having L bits, and then embedding each L- block in L LSB's of each pixel of the cover image, where $1 \leq L \leq 8$. In general, this method can achieve a good image quality when $L \leq 3$, but for $4 \leq L \leq 8$, the image quality severely decreased [8].

To improve LSB replacement, many steganographic methods were proposed. In 2001 Wang, & Lin proposed a method that uses an optimal LSB replacement and genetic algorithm [12], where the genetic algorithm is presented to solve the problem of hiding data in the L LSBs of the cover image when L is large in order to improve the image quality and embedding capacity.

In 2002 Yu- Chee proposed a secure data hiding scheme for binary image [10], that uses a binary cover image to embed as many as $\log_2(mn+1)$bits of secret message into $m \times n$ block of binary cover image by changing at most two bits in the block , so this method has good image quality and embedding capacity.

In 2003 Wu & Tsai proposed a new method called Pixel Value Differencing (PVD) [13]. In this method, the cover image is divided into non-overlapping blocks of two adjacent pixels. A difference value is calculated from the values of the two pixels in each block. All possible difference values are classified into a number of ranges. The difference value then is replaced by a new value to embed the value of a sub-stream of the secret message. The number of bits which can be embedded in a pixel pair is decided by the width of the range that the difference value belongs to [13]. This method provided a better way to embed larger amount of secret data.

In 2005 Wu et al proposed a method based on LSB replacement and PVD methods [14]. First, a difference value from two adjacent pixels by PVD method is obtained, where small difference value can be located on a smooth area and the large one is located on an edged area. In the smooth areas, the secret data is hidden into the cover image by LSB method while using the PVD method in the edged areas. This method provided double embedding capacity of PVD method with a good stego image quality PSNR.

In 2006 Mielikainen proposed a modification to LSB method that uses a pair of pixels from the cover image as a group [4], where the secret bits are carried in LSB's of two pixels. Therefore this method has the same payload as LSB replacement method, but with fewer changes to the cover image pixels. So the performance of this method is better than LSB replacement, and the direction of modification to the cover pixels is exploited for data hiding, but there exist two different modification-directions corresponding to a same pair of secret bits to be embedded, meaning that the exploitation is incomplete [10].

Also in 2006 Zhang and Wang proposed a new method called Exploiting Modification Direction (EMD) [15]. The main idea of the EMD method is to use a separate n-pixel group of a cover image to embed the next digit of $(2n+1)$-ary k-digit number representing the next L-bit block from the secret image input and only one pixel in the group can be changed by ±1. Therefore, this method has very good image quality and better embedding capacity, but embedding capacity decreases as increasing n.

To improve EMD method Lee et al. proposed Improved EMD (IEMD) method in 2007 [9]. This method uses two pixels from the cover image as group and 8-ary extraction function. It has greater embedding capacity than EMD, but it uses only two pixels in a group and cannot use more.

To enhance the hiding capacity of EMD and IEMD methods, a novel information concealing method based on Exploiting Modification Direction was proposed in 2011 [16]. This method embeds $2x$ secret digits in the 5-ary notational systems into each group of $(2x + 1)$ cover pixels, where x is a positive integer. Thus, the proposed method can provide better hiding capacity.

In 2013 Kuo and Wang provided GEMD method [5], where it uses n-pixels from the cover image to embed n+1 bits , and at least one pixel value in each group could be changed by ±1. Also in this method there is no need for transformation, GEMD maintained good image quality and good embedding capacity, and also it can adjust the n-pixel size.

Frequency domain uses the transform coefficients to embed secret data. Moreover, frequency domain techniques are very robust against attacks. In frequency domain the cover image is transformed into the frequency domain coefficients before embedding secret messages in it, where the main techniques used are: Discrete Cosine Transform (DCT) [11], and Discrete Wavelet Transform (DWT), in Discrete Cosine Transform [17].

DCT method is used extensively with video and image compression e.g. JPEG compression, since for each color component the JPEG image format uses a discrete cosine transform to transform successive 8 × 8 pixel blocks of the image into 64 DCT coefficients each [11].

In DWT method [17], the cover image is divided into four sub-images such as approximation coefficients (CA), horizontal detail coefficients (CH), vertical detail coefficients (CV) and diagonal detail coefficients (CD). Similarly, the secret image is decomposed into four sub-images. These sub-images are divided into non-overlapping blocks. The blocks of approximation coefficients of cover image are subtracted from approximation coefficient of secret image. The differences of these coefficients are called error blocks. The replacement of an error block is being done with the best matched CH block [17].

Though frequency domain methods are more difficult and slower than spatial domain methods, yet they provide more security [1]. In this work two spatial domain methods EMD and GEMD will be discussed in details which similar to frequency domain methods provide greater security by the use of data transformations but in the spatial domain. In addition, in [7] Kuo and Wang provided a comparison between EMD and GEMD methods over different values of *n*-pixel group using the metrics Peak Signal to Noise Raito (PSNR), Mean Square Error (MSE), and embedding capacity Bit Per Pixel (BPP). Since they considered the embedding capacity for GEMD is better than EMD, but the comparison in this case is not valid because the parameter *n* has different meaning for EMD and GEMD. For EMD *n* is the number of pixel required for one digit among *k* digits in one block, but for GEMD it is the total number of pixels required for one block, where in EMD total number of pixels required for one block is *n.k* not just *n* as in GEMD. So we need to analyze and experimentally study these methods as they exploited the modification of direction with bit differences in embedding and extracting processing, also we compare the performance of them using the averages of the metrics PSNR, MSE, and BPP in addition to the memory and time consumption.

2.3.1 EMD Method

Proposed in [14], it uses the next *n*-pixel group of a cover image to embed one digit of (2*n*+1)-ary *k*-digit number representing the next L-bit block of the secret message binary stream, and only one pixel value may be changed by ±1.

EMD Embedding Algorithm

Begin

Inputs: cover image, *CI (M,N)*; *M* is the number of rows; *N* is the number of columns; integer, *n* >0, pixel group size; integer, *L* >1, input binary stream block size; binary secret message, *S*.

Output: stego image, *SI(M,N)*.

Step 1. Get next binary secret message block having *L* bits, and convert it to

(2*n*+ 1)-ary *k*-digit number, where *k* is defined from the next relations

$$2^{L} \leq (2n+1)^{k}$$

$$L \leq \log_2 (2n+1)^{k}$$

$$L \leq \lfloor k.log_2(2n+1) \rfloor$$

$$k = \left\lceil \frac{L}{log_2(2n+1)} \right\rceil \quad , \tag{2.1}$$

where ⌊x⌋ and ⌈x⌉ are floor and ceil functions.

Step 2. For each digit s_i, i=1, …., k,

Begin

Get next pixel group from cover image, CI, $X=(x_1,x_2,...,x_n)$, and calculate

$$t=ef(x) = \sum_{i=1}^{n} x_i \,.i \; mod \; (2n+1) \tag{2.2}$$

Calculate

$$d=(s_i - t) \bmod (2n+1) \qquad (2.3)$$

Set

$$X'=X \qquad (2.3.1)$$

If $d=0$, nothing is made.

If $d \leq n$, increase the d^{th} pixel in the pixel group by 1:

$$x'_d = x'_d + 1 \qquad (2.3.2)$$

Otherwise, decrease $((2n+1)-d)^{\mathrm{th}}$ pixel in the pixel group by 1:

$$x'_{(2n+1-d)} = x'_{(2n+1-d)} - 1 \qquad (2.3.3)$$

End of step2.

Step 3. Go to Step 1 until the secret message is embedded.

Step 4. End.

Figure 2.1 shows the flow chart diagram of EMD embedding, while Figure 2.2 shows the flow chart diagram for Step 2 in EMD embedding.

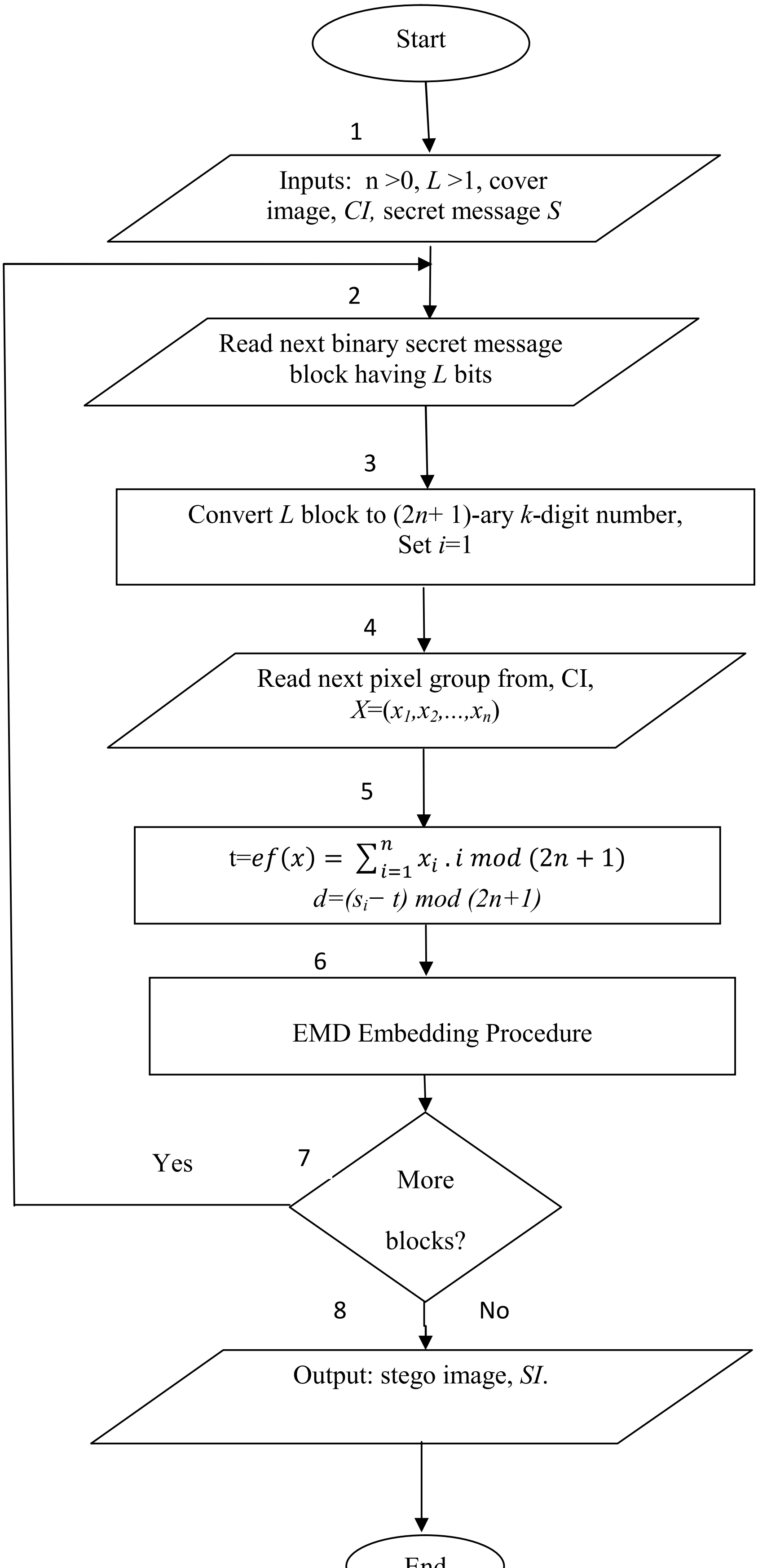

Start
1
Inputs: n >0, L >1, cover image, CI, secret message S
2
Read next binary secret message block having L bits
3
Convert L block to (2n+ 1)-ary k-digit number, Set i=1
4
Read next pixel group from, CI, $X=(x_1,x_2,\ldots,x_n)$
5
t=$ef(x) = \sum_{i=1}^{n} x_i \,.\, i \; mod \; (2n+1)$
$d=(s_i - t) \; mod \; (2n+1)$
6
EMD Embedding Procedure
Yes
7
More blocks?
8
No
Output: stego image, SI.
End

Figure 2.1: Flow chart diagram of EMD embedding

EMD Embedding Procedure:

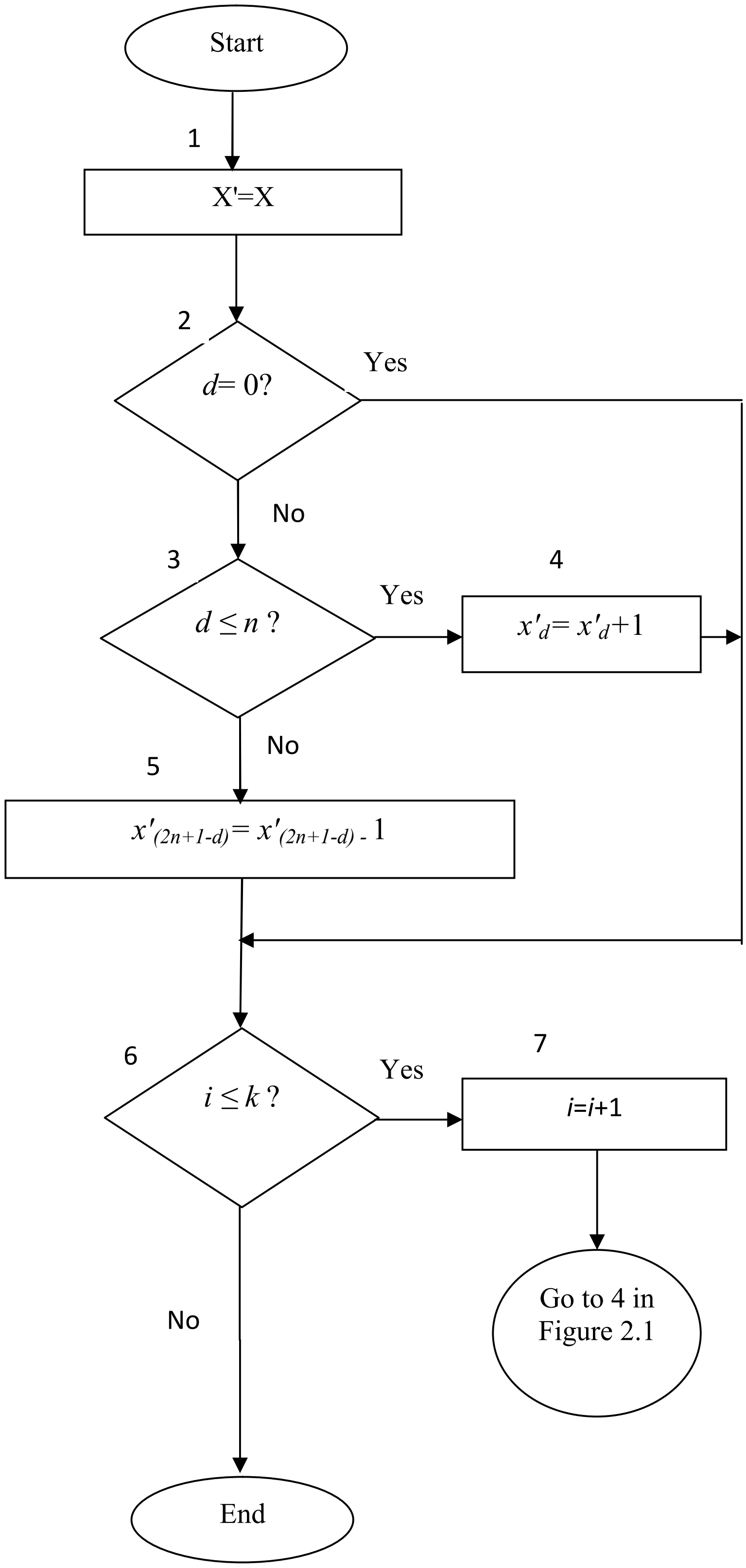


Figure 2.2: Flow chart diagram for EMD Embedding Procedure (Block 6 in Figure 2.1)

EMD Extraction Algorithm

Begin

Inputs: stego image, *SI(M,N)*; *M* is the number of rows; *N* is the number of columns; *integer, n>0* pixel group size.

Outputs: binary secret message, *S*.

Step 0: Set *S*={ };*//empty set.*

Step 1. Obtain the next *n*-pixel block X'=($x'_1,x'_2,\ldots,x'_n$) from stego image, *SI*.

Step 2. Calculate

$$s=ef(x'_1, x'_2, \ . \ . \ ., x'_n) = \sum_{i=1}^{n} x'_i \, . \, i \ mod \ (2n+1) \qquad (2.4)$$

Step 3. Transform *s* into L-bit binary block and append it to the secret data stream, *S*. Go to **Step 1**.

Step 4. End.

Figure 2.3 shows the flow chart diagram of EMD extraction**.**

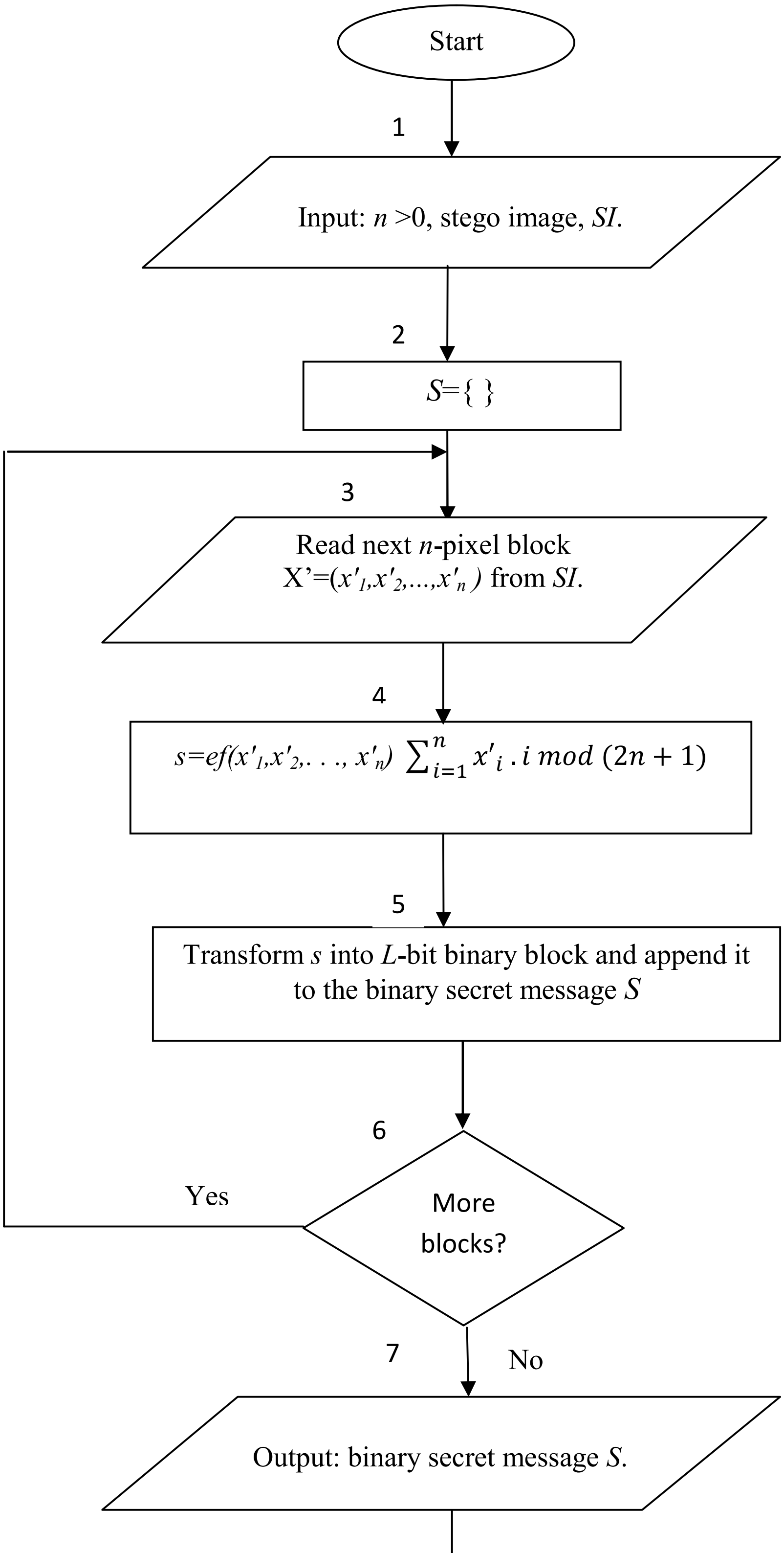

Start
1
Input: n >0, stego image, SI.
2
S={ }
3
Read next n-pixel block
X'=($x'_1,x'_2,\ldots,x'_n$) from SI.
4
$s=ef(x'_1,x'_2,\ldots,x'_n)\ \sum_{i=1}^{n} x'_i\ .\ i\ mod\ (2n+1)$
5
Transform s into L-bit binary block and append it to the binary secret message S
6
More blocks?
Yes
7
No
Output: binary secret message S.
End

Figure 2.3: Flow chart diagram of EMD Extraction

EMD Embedding Example

Let we have the following binary secret message

S= 11100 01101 10101 10101 00011 11100 and CI pixel values are:

CI= 162 163 163 161 162 158 163 161

162 159 155 164 160 155 156 156

If n=2, then we have 2 pixel groups, eg., (x_1, x_2) = (162 163), and L=5 bits in each block of secret message, then we get k =3 from Eq. (2.1). For the first 5-bit block, convert it into (2n+1)-ary k-digit number, so we have in base 5

$$(11100)_2 = (28)_{10} = (103)_5.$$

Next we take 2 pixels to embed the first digit s=1 and we apply the extraction function as in Eq. (2.2)

$$t = (162\times1 +163\times2) \bmod 5 = 3$$

Then calculate the difference d as in Eq. (2.3)

$$d= (1-3) \bmod 5,$$

$$d= -2 \bmod 5 = 3.$$

Since d>n, then modify the pixel at position (2n+1)-d, it means at position 2, so second pixel will decrease by one, and we get

$$\text{group}_1\ (x'_1,x'_2)=(162\ \ 162).$$

We do the same steps for second digit s=0 and second group (x_1, x_2) = (163 161) as follows

$$t = (163\times1 +161\times2) \bmod 5 = 0$$

$$d= (0-0) \bmod 5 = 0$$

Since d=0, then no pixel in a group is modified, then group$_2$ (x'_1,x'_2) = (163 161).

For third digit s=3 and group$_3$ (x_1, x_2) = (162 158) we calculate

$$t = (162\times1 +158\times2) \bmod 5 = 3$$

d= (3-3) mod 5 = 0 then

group$_3$ (x'_1,x'_2)=(162 158).

In the extracting stage if we apply the Eq (2.4) for each group, then we can get our secret digits as follows

s_1= (162×1 +162×2) mod 5 = 1

s_2= (163×1 +161×2) mod 5 = 0

s_3= (162×1 +158×2) mod 5 = 3

So we get 3 digits in base 5 $(103)_5$, which are converted to binary to get our original bits $(11100)_2$.

2.3.2 GEMD Method

This method proposed in [5], uses n-pixel group from the cover image to embed a block of (n+1) bits.

GEMD Embedding Algorithm

Begin

Inputs: Cover image, *CI(M,N)*; *M* is the number of rows; *N* is the number of columns; integer, n>0, defining bit block and pixel group size; binary secret message, *S*.

Output: stego image, *SI (M,N)*.

Step 1. Get next n-pixel group $X=(x_1,x_2,...,x_n)$ from cover image *CI*

Step 2. Get next binary secret message, S, block having (n+1) bits with decimal value s.

Step 3. Compute $ef(x_1,x_2,...,x_n)$ with the pixel groups:

$$t=ef(x_1,\ x_2,\ .\ .\ .,\ x_n) = \sum_{i=1}^{n} x_i\,.\left(2^i - 1\right) mod\ 2^{n+1} \qquad (2.5)$$

Step 4. Compute the difference d

$$d= (s - t)\ mod\ 2^{n+1} \qquad (2.6)$$

```
Step 5. If $d=2^n$ then $R = 1$;
    else if ($d< 2^n$) then $R = 2$; else $R = 3$;
Step 6. Switch ($R$)
Case 1: Let $x'_n = x_n + 1$, $x'_1 = x_1 + 1$ . $x'_i = x_i$ , $i=2,..,n-1$
Case 2: let $d = (d_n\, d_{n-1}\, d_{n-2} \ldots d_1\, d_0)_2$
  for $i = n$ down to 1 do
  Begin
      if ($d_i = 0$ and $d_{i-1} = 1$) then $x'_i = x_i + 1$;
      else if ($d_i = 1$ and $d_{i-1} = 0$) then $x'_i = x_i - 1$;
      else $x'_i = x_i$
  End.
Case 3: Let $d' = 2^{n+1} - d$ . Let $d' = (d_n\, d_{n-1}\, d_{n-2} \ldots d_1\, d_0)_2$
  for $i = n$ down to 1 do
  Begin
      if ($d_i = 0$ and $d_{i-1} = 1$) then $x'_i = x_i - 1$
      else if ($d_i = 1$ and $d_{i-1} = 0$) then $x'_i = x_i + 1$;
      else $x'_i = x_i$
  End.
Step 7. Go to Step1 until secret the message is embedded.
Step 8. End.
```

Figure 2.4 shows the flow chart diagram of GEMD embedding, while Figure 2.5 and Figure 2.6 show the flow chart diagram for Step6-Case 2 and Step6-Case 3 in GEMD embedding respectively.

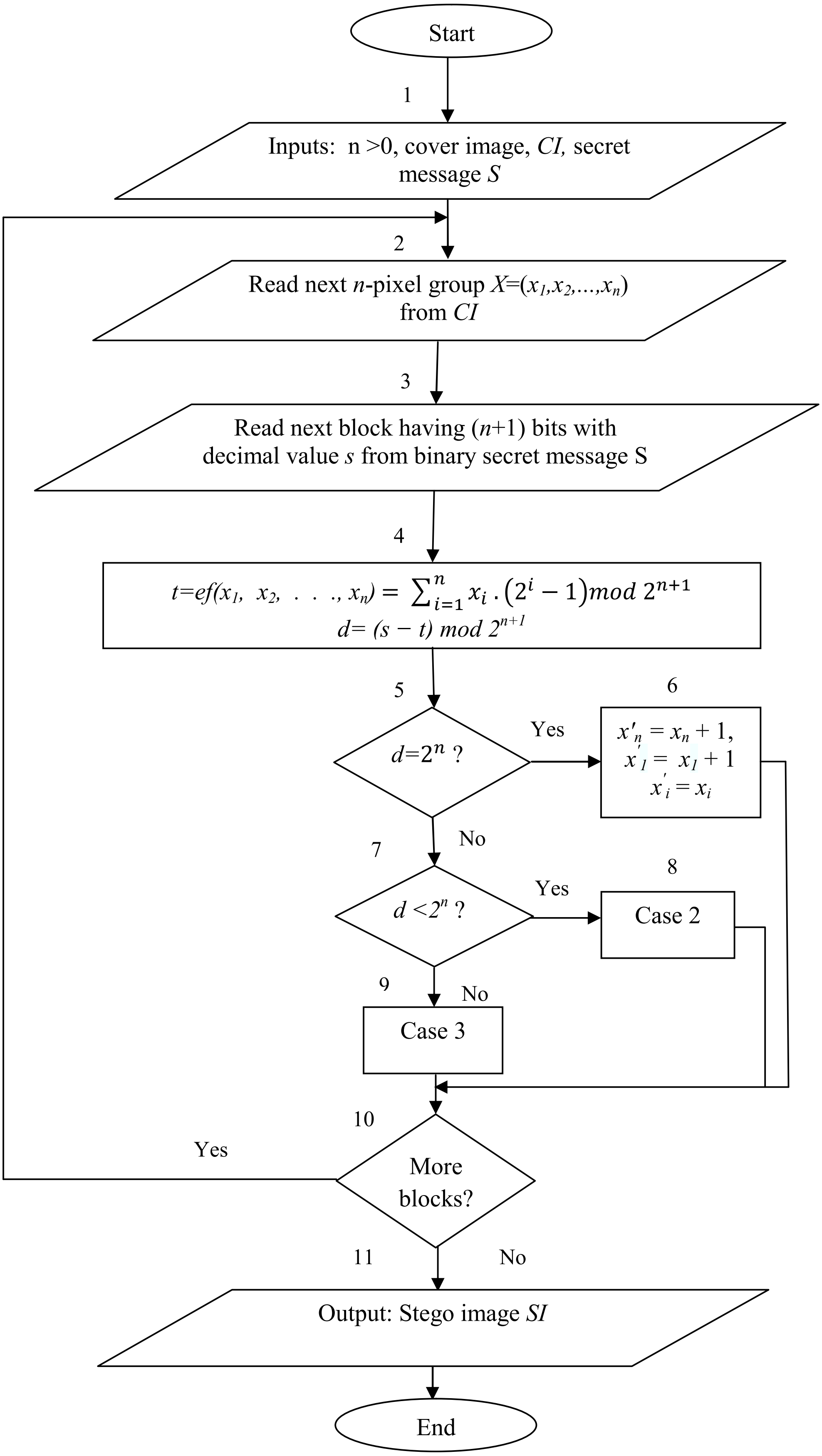

Start
1
Inputs: n >0, cover image, *CI,* secret message *S*
2
Read next *n*-pixel group $X=(x_1,x_2,...,x_n)$ from *CI*
3
Read next block having (*n*+1) bits with decimal value *s* from binary secret message S
4
$t=ef(x_1, x_2, . . ., x_n) = \sum_{i=1}^{n} x_i . (2^i - 1) mod\ 2^{n+1}$
$d= (s - t)\ mod\ 2^{n+1}$
5
$d=2^n$?
Yes
6
$x'_n = x_n + 1,$
$x'_1 = x_1 + 1$
$x'_i = x_i$
No
7
$d < 2^n$?
Yes
8
Case 2
9
No
Case 3
10
Yes
More blocks?
11
No
Output: Stego image *SI*
End

Figure 2.4: Flow chart diagram of GEMD embedding

Case 2:

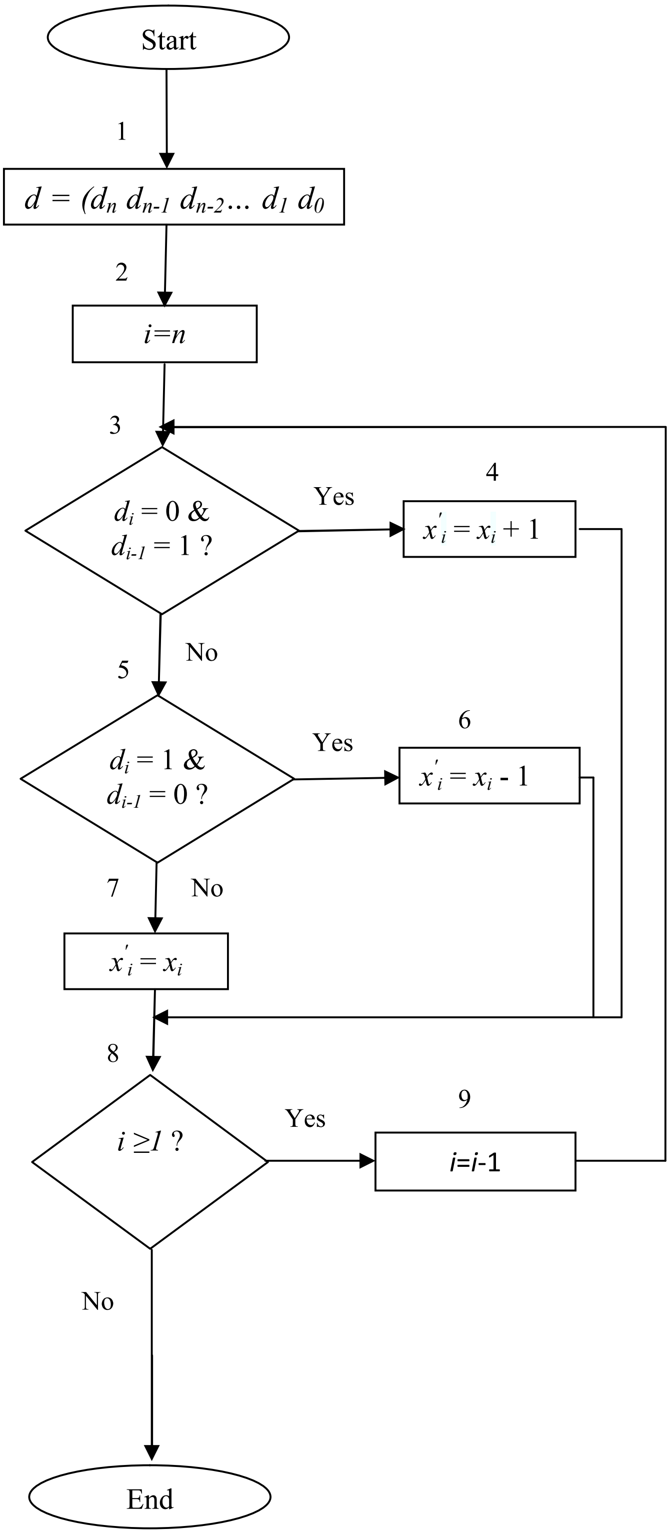


Figure 2.5: Flow chart diagram for Step6-Case 2 in GEMD embedding (Block 8 in Figure 2.4)

Case 3:

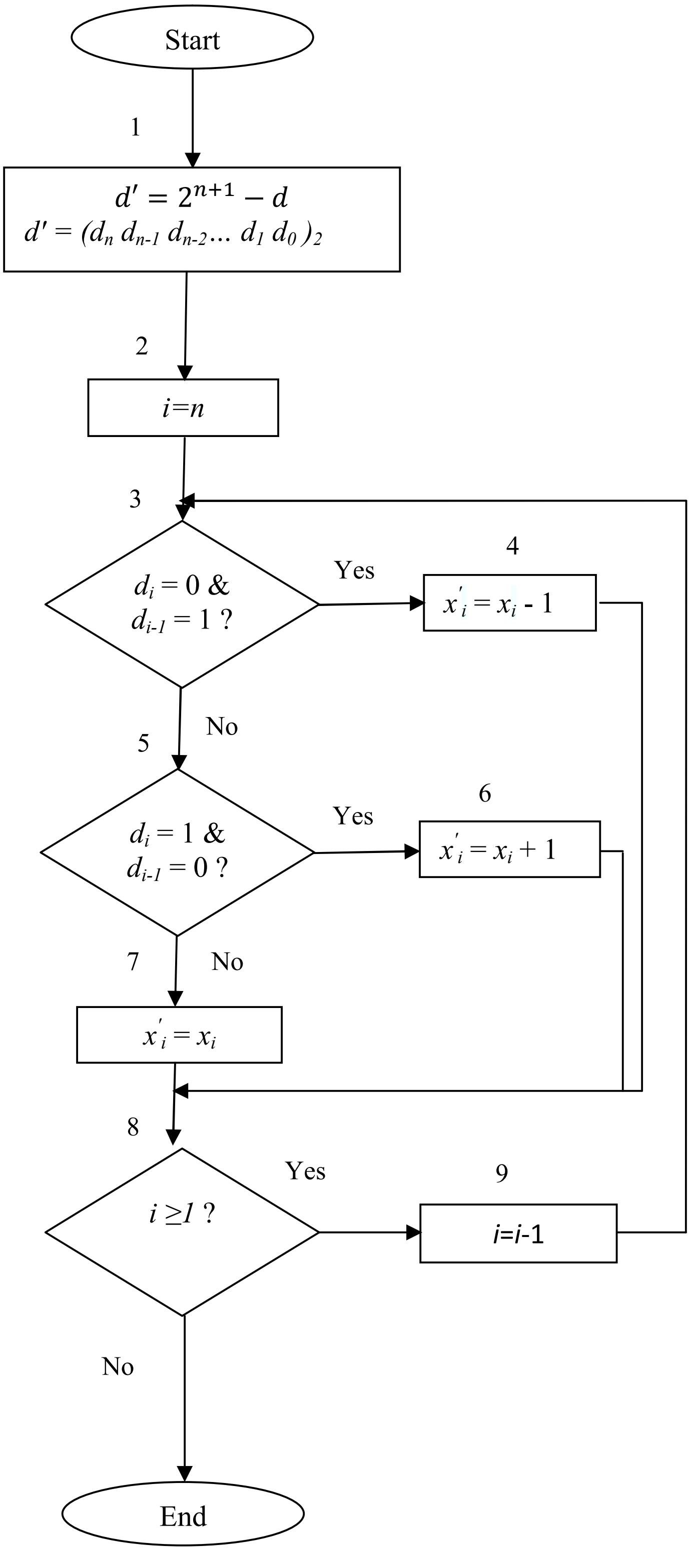


Figure 2.6: Flow chart diagram for Step6-Case 3 in GEMD embedding (Block 9 in Figure 2.4)

GEMD Extraction Algorithm

Begin

Inputs: stego image, *SI(M,N); M* is the number of rows; *N* is the number of columns*;* integer, *n>0,* defining binary block and pixel group size.

Outputs: binary secret message, *S*.

Step 0. Set *S*={};*//empty set*

Step 1. Get next *n*-pixel group, $x=(x_1,x_2,...,x_n)$, from stego image, *SI*.

Step 2. Calculate

$$s=ef(x'_1,x'_2,...,x'_n) = \sum_{i=1}^{n} x'_i \,.\, (2^i - 1) mod\ 2^{n+1} \qquad (2.7)$$

Step 3. Append *s* as (*n*+1)-bit binary block to binary output secret data stream, *S.*

Step 4. If *SI* has not processed blocks, go to step1.

Step 5. End

Figure 2.7 shows the flow chart diagram of GEMD extraction**.**

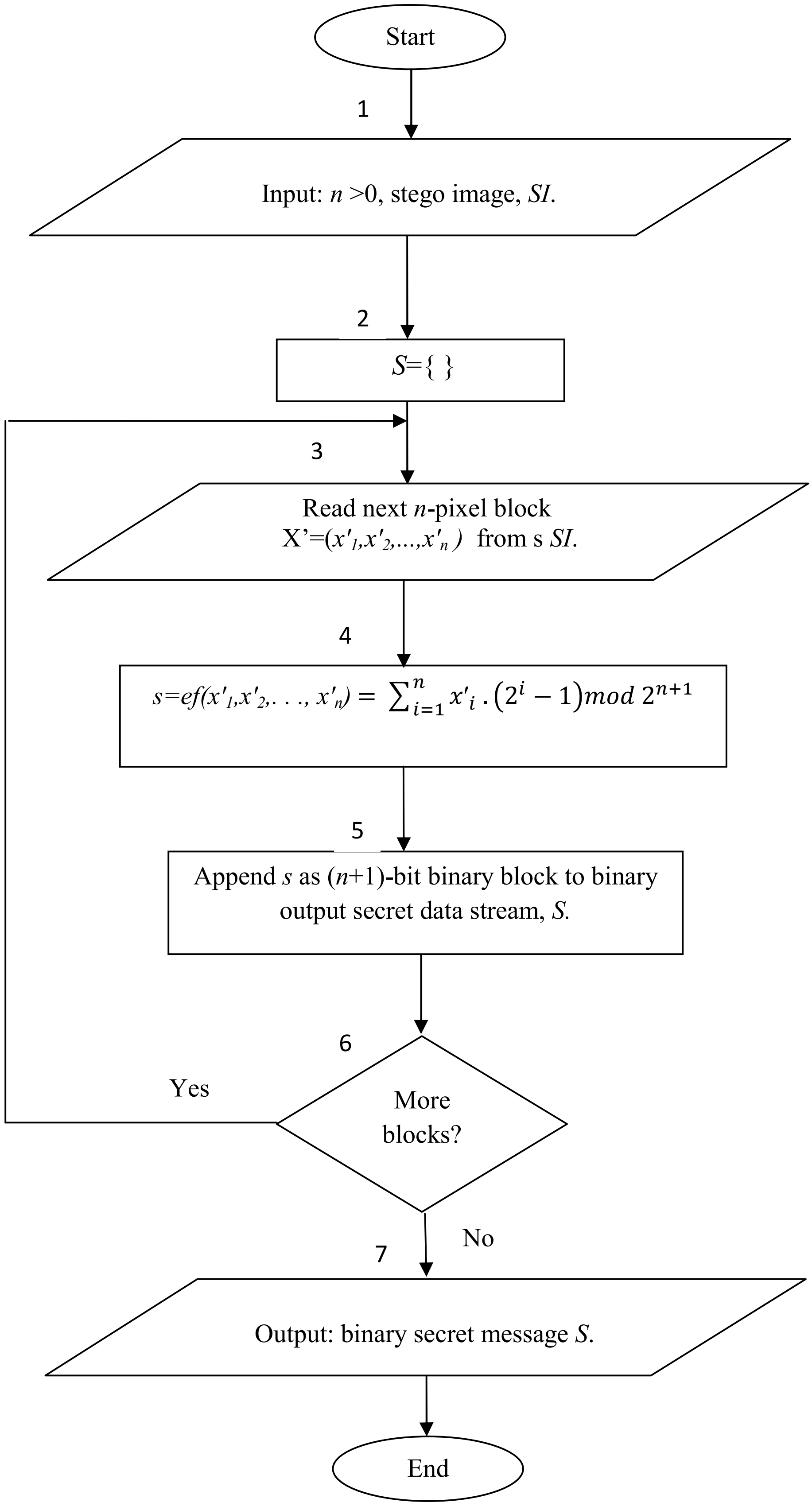

Start
1
Input: n >0, stego image, SI.
2
S={ }
3
Read next n-pixel block
X'=(x'_1,x'_2,...,x'_n) from s SI.
4
$s=ef(x'_1,x'_2,\ldots,x'_n) = \sum_{i=1}^{n} x'_i\,.\left(2^i-1\right) mod\ 2^{n+1}$
5
Append s as (n+1)-bit binary block to binary output secret data stream, S.
6
Yes
More blocks?
No
7
Output: binary secret message S.
End

Figure 2.7: Flow chart diagram of GEMD Extraction

GEMD Embedding Example

In GEMD method we embed (n+1)-bit blocks in the n-pixel groups, so in the condition of Example 2, for n=2, we embed 3 bits in 2 pixels, then

$$\text{Block}_1= (111)_2 = (7)_{10} = s_1$$

$$\text{group}_1\ (x_1,x_2)=(\ 162\ \ 163).$$

From Step 3 in GEMD embedding algorithm

$$t = (162\times1 +163\times3) \bmod 8 =3$$

From Step 4,

$$d = (7\text{-}3) \text{ mod}8=4$$

As d=4=2^n, and from Step 6, Case1, the first and last pixels are increased by 1, then

$$\text{group}_1\ (x'_1,x'_2)=(\ 163\ \ 164).$$

If we apply the extraction function as in Eq. (2.7)

$$s_1= (163\times1 +164\times3) \bmod 8=7$$

So we get the number $(7)_{10}$, which if converted to binary gives our original bits $(111)_2$.

2.4 Known Experiments on EMD and GEMD

In [7] the performance of EMD and GEMD was evaluated using the following quality metrics.

1. Mean Square Error (MSE) is defined as mean squares differences between the original cover image and image after embedding [7]:

$$MSE = \frac{1}{M \times N}\sum_{c=1}^{M}\sum_{r=1}^{N}(CI(r,c)-SI(r,c))^2 \qquad (2.8)$$

where M is the number of rows and N is the number of columns of the cover and stego images. *CI (r,c)* is the original image pixel value and *SI(r,c)* is stego image pixel value .

2. Signal Peak to Noise Ratio (PSNR) is calculated as follows

$$PSNR = 10log_{10}\frac{255\times255}{MSE} \quad \text{dB} \tag{2.9}$$

where 255 is the maximum value of pixels for grey scale images.

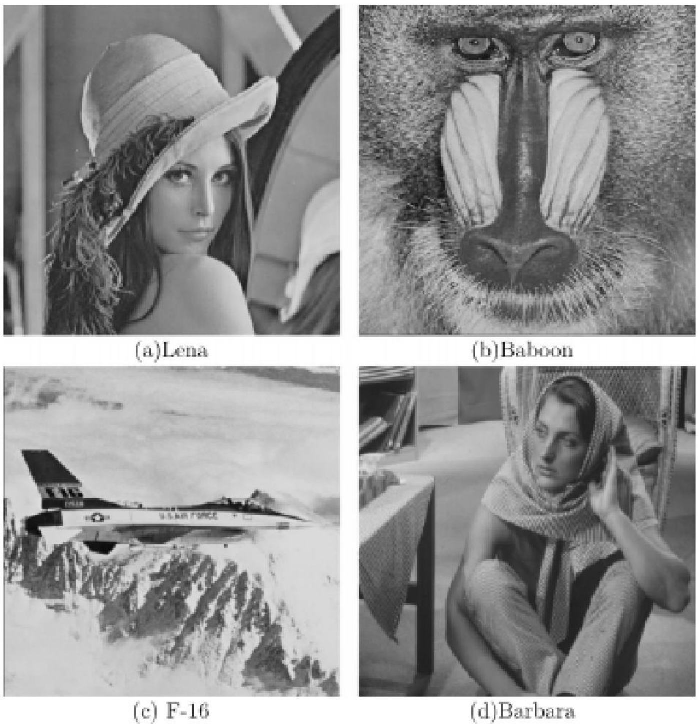
3. Embedding capacity Bit Per Pixel (BPP) is defined as the number of secret bits embedded in each pixel of cover file. For EMD, $log_2(2n+1)$ bits that represent a $(2n+1)$-ary digit embedded in n pixels, while in GEMD $(n+1)$- bit values are embedded in n-pixel group [14]. BPP is calculated for EMD and GEMD as follows [7]:

$$\text{Bpp}_{\text{EMD}} = \frac{log_2\ (2n+1)}{n} \tag{2.10}$$

Where number of bits embedded = $log_2\ (2n+1)$

$$\text{Bpp}_{\text{GEMD}} = \frac{n+1}{n} \tag{2.11}$$

The results taken for PSNR and MSE from [7] are shown in Table 2.1, and the Figure 2.8 shows the four cover images that are used in [7].



(a)Lena (b)Baboon

(c) F-16 (d)Barbara

Figure 2.8: Cover images used in [7]

Table 2.1: EMD-versus-GEMD known comparison results for PSNR and MSE [7]

	n=2		*n*=3		*n*=4		*n*=5	
	EMD	GEMD	EMD	GEMD	EMD	GEMD	EMD	GEMD
PSNR (dB)	52.11	50.17	53.57	50.79	54.66	51.00	55.53	51.09
MSE	0.40	0.62	0.28	0.54	0.22	0.51	0.18	0.50
(BPP)	1.16	1.50	0.93	1.33	0.79	1.25	0.69	1.20

From Table 2.1, the EMD scheme has very good image quality. Also for EMD method, the largest embedding capacity is 1.16 BPP when *n*= 2 and its capacity is less than 1 BPP when *n*≥3. For GEMD it maintains good stego image quality, and the embedding capacity is greater than 1 BPP when number of pixels in each group of cover image increases [7]. But the comparison in Table 2.1 is not valid because the parameter n has not the same meaning for both methods. In EMD *n* is the number of pixel required for one digit, not for one block as in GEMD.

In the experiments conducted in this thesis, we tried to find the best values of EMD and GEMD parameters that achieved the results mentioned in Table 2.1 with minimum number of cover images, and the comparison between both methods will be taken as the average over the metrics PSNR, MSE, BPP, time and memory consumption.

2.5 Problem Definition

In this research, two steganographic algorithms, EMD and GEMD, are studied. They are selected as representing a perspective direction in steganographic methods combining features from the main two directions: embedding in the space domain (as LSB-like methods embedding secret data directly in the cover image) and embedding in the frequency domain [1]. EMD and GEMD embed in the space domain but similar to frequency domain methods use data transformations. In the papers on EMD and GEMD, justification for the methods is not provided; hence, we prove their correctness. Also, information is not provided such as data structures and the implementation details like the input-output data structures. They are experimentally investigated for image size 512x512, and we extend experiments to 1024x1024 size images.

Data structures for the both methods and the justification of their correctness will be explained in Chapter 3. Implementation details will be explained in Chapter 4. The simulation with the best values for EMD and GEMD parameters that required minimal number of cover images will be discussed and our results will be compared first between the both methods and then with the known experiments in Chapter 5. In Chapter 6, we give conclusions and discuss the future work.

2.6 Summary of Chapter 2

Thus, in this chapter we have presented an overview of steganography and the related work: we explained two algorithms, EMD, and GEMD with an example for each of them. We considered the experiments conducted in [7], and finally we defined the problem.

Chapter 3

EMD AND GEMD DATA STRUCTURES AND JUSTIFICATION OF THE METHODS CORRECTNESS

In this chapter the details of data structures used in the implementation of EMD and GEMD algorithms will be discussed, such as the input-output data structures. Justification of EMD and GEMD correctness are given.

3.1 Data Structure and Justification of EMD Correctness

EMD method is described in Section 2.3.1. Now we consider necessary for its implementation data structures and of EMD correctness.

3.1.1 Data Structure for EMD Embedding

Inputs structure:

1. Integers, $n>0$, pixel group size; L>1*,* bit block size.

2. Binary message, *S,* sized |*S*| bits, we consider as a sequence of blocks B_i sized $|B_i|=L$ bits, $i=[0,1,\ldots H-1]$ where the number of blocks H is defined as follows:

$$H \ = \left\lceil \frac{|S|}{L} \right\rceil \tag{3.1}$$

The last block is padded by zeros when $|\ B_{H-1}\ | \neq L$, $\hat{S}\ =S+$ zeros (L-|S| mod L), where + stands for concatenation, each B_i is converted into *k*-digit number in the (2*n*+1)-ary notational system. Input, S, may be represented as a sequence of digits:

$S_{\text{dig}}= [s_0,s_1,\ldots..s_{i.k+j},\ldots\ldots,\ s_{(H-1).k+(k-1)}]$, $i=[0,1,\ldots H-1]$, $j=[0,1,\ldots k-1]$, where $s_{i.k+j}$ is j^{th} (2*n*+1)-ary digit of i^{th} *k* digits (2*n*+1)-ary number, *k* is specified as in Eq.(2.1).

3. Grayscale cover image number *j* from the set of cover images CI is represented as a matrix $CI_j\ [M,N]$,where *M* is the number of rows and *N* is the number of columns,

$CI_j(r,c) \in [0,1,\dots\dots 255]$. $0 \le r \le M\text{-}1$, $0 \le c \le N\text{-}1$,$0 \le j \le \tilde{N}\text{-}1$,where $\tilde{N}$ is the number of cover images that is necessary to embed secret message S defined by following relation

$$\tilde{N} = |CI| = \left\lceil \frac{H \times k}{C} \right\rceil \tag{3.2}$$

$$C = \left\lfloor \frac{M \times N}{n} \right\rfloor \tag{3.3}$$

where C is the number of n-pixel groups fitting one cover image

Output: Stego images SI_j [M, N], with embedded message, $0 \le j \le \tilde{N}\text{-}1$.

A cover image is represented by one-dimensional array $X_j = \{ x_{j0}, x_{j1}, \dots, x_{j.(M.N-1)} \}$ by scanning each row of image from left to right and from top to bottom *(Row major order C-style)* as in Fig. 3.1:

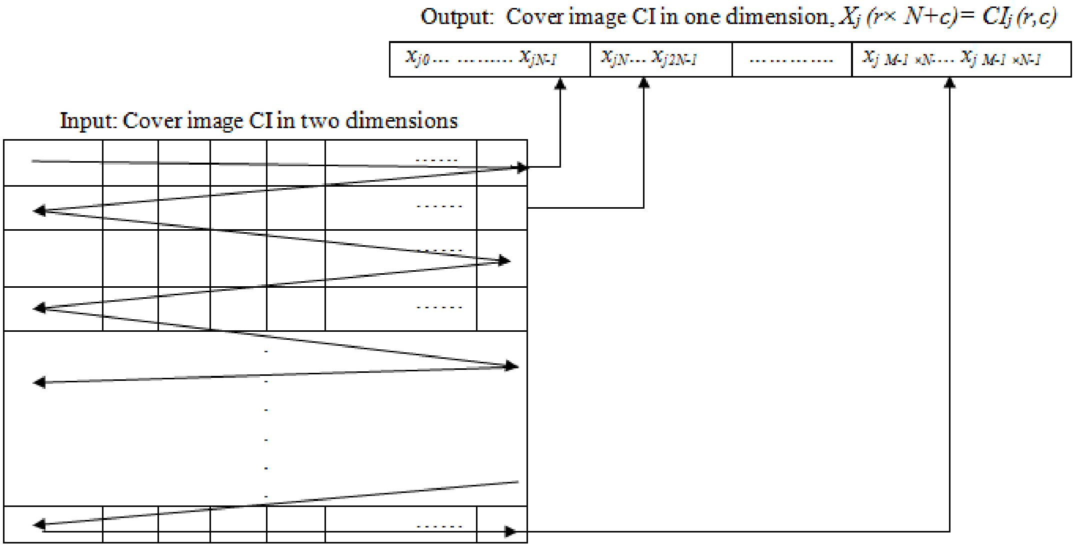

Output: Cover image CI in one dimension, $X_j(r\times N+c) = CI_j(r,c)$
$x_{j0} \dots \dots\dots\dots x_{jN-1}$
$x_{jN} \dots x_{j2N-1}$
…………
$x_{j\,M-1\,\times N} \dots x_{j\,M-1\,\times N-1}$
Input: Cover image CI in two dimensions

Figure 3.1: Reshaping cover image as one dimensional array

The one dimensional cover image X_j is divided into non-overlapping groups of n pixels. Each j-th digit $s_{i.k+j}$ of i^{th} k-digit (2n+1)-ary number from S_{dig}, where S_{dig} is a secret message represented as sequence of *(2n+1)*-ary digits of k-digit numbers, $i=[0,1,\dots H\text{-}1]$, $j=[0,1,\dots k\text{-}1]$, is embedded in a group of n-pixels

$x=[\ x_{(i.k+j)n}, \ldots\ldots\ x_{(i.k+j).n+n-1}],\ i=[0,1,\ldots H-1],\ j=[0,1,\ldots k-1]$ that is illustrated by Figure 3.2.

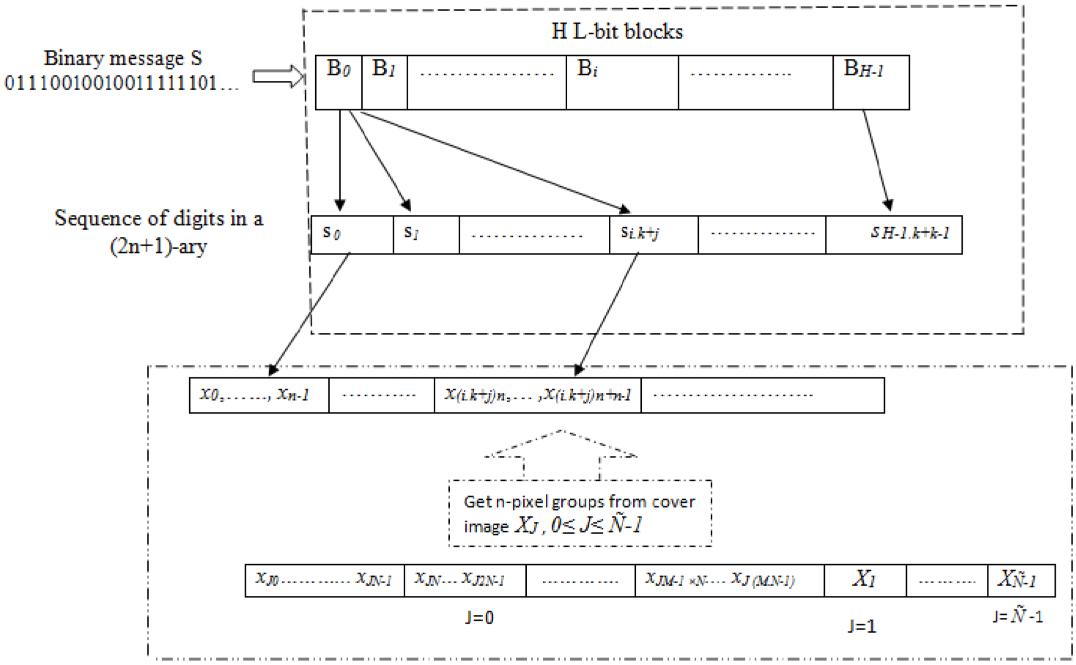


Figure 3.2: EMD data structure for embedding procedure

For instance, let L=5 bits, n=2 pixels in each group, then 5- bit binary blocks are converted to 5-ary numbers with k=3 digits, where k is defined by (2.1), so we need 3 groups to embed these 3 digits and each group has 2 pixels, it means that three out of six pixels will be changed at most.

Example 1. If we have a binary secret message with size |S|=100 bits, and L=5 bits in each block, then number of these blocks is

$$H = \frac{|S|}{L} \quad = \frac{100}{5} = 20 \text{ blocks}$$

And S may be represented as sequence of blocks $[B_0, B_1 \ldots\ldots\ldots B_{19}]$.

If the number of pixels n is two for each group, then (2n+1)-ary is 5-ary. To embed S we need $H.k$ groups of n pixels, i.e. number of pixels is $H.k.n$=20.3.2=120 pixels.

3.1.2 Data Structures for EMD Extraction

Input structures:

Integer, n>0; defining pixel group size.

Grayscale stego images SI_j $[M,N]$,where M is the number of rows and N is the number of columns, $SI_j(r,c) \in [0,1,\ldots 255]$, $0\leq r\leq M\text{-}1$,$0\leq c\leq N\text{-}1$,$0\leq j\leq \tilde{N}\text{-}1$, where $\tilde{N}$ is the number of stego images that is defined by Eq. (3.2).

Output: Binary secret message, S.

The stego images are represented as one dimensional arrays as in Figure.3.1. One-dimensional stego image X_j is divided into non-overlapping groups of n pixels. For each n-pixel group of X_j calculate (2.4). Data structure for extraction procedure is illustrated in Figure 3.3.

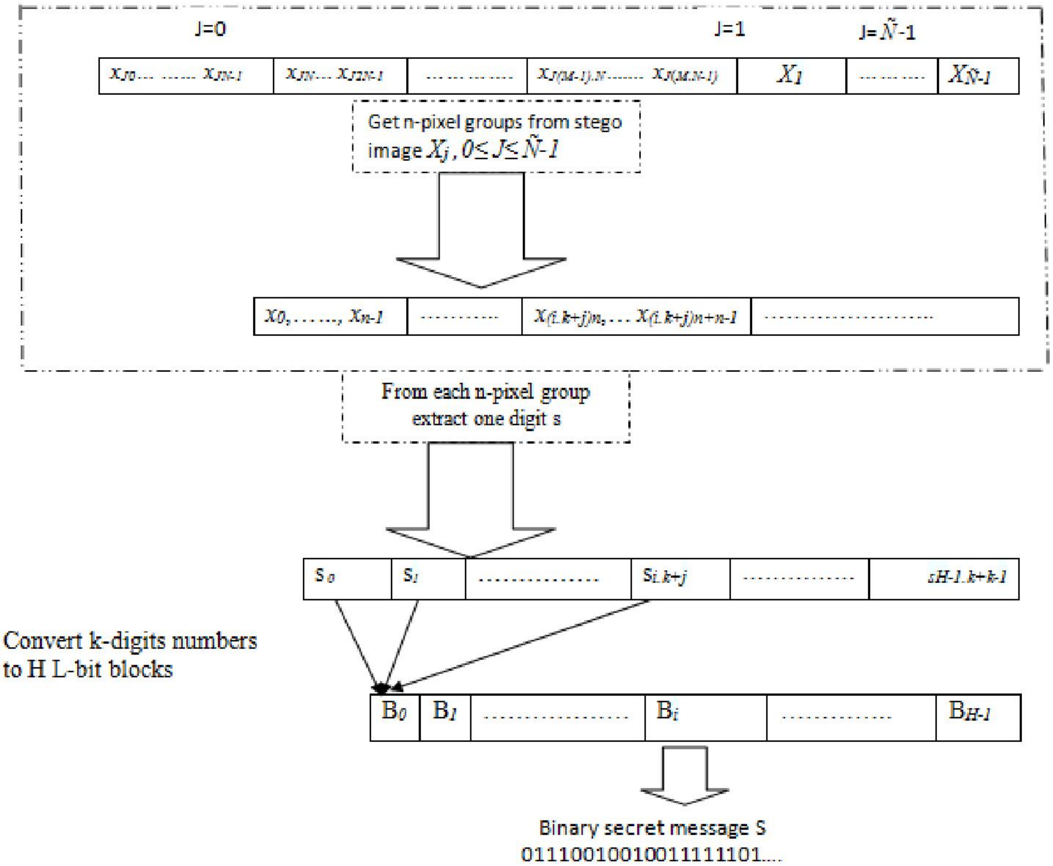


Figure 3.3: EMD data structure for extraction procedure

3.1.3 Justification of EMD Correctness

Statement 1.

Let's assume that EMD algorithm from Section 2.3.1 is applied, and digit s_i is embedded in a group of *n* pixels $x=(x_1,\ldots\ldots,x_n)$ resulting in stego image group $x'=(x'_1,\ldots x'_n)$. Then by applying Step2 of EMD extraction algorithm, *Eq.* (2.4), value of s_i is returned.

Statement 1 Justification.

From (2,4)

$$ef(x') = \sum_{i=1}^{n}(x'_i\,.\,i)\ mod(2n+1\,)$$

Consider three cases for *d* calculated by (2.3).

Case 1. *d=0.* In this case, from (2.3.1), no pixel was modified, and $x'= x$

$$ef(x') = \sum_{i=1}^{n}(x_i\,.\,i)\ mod(2n+1\,) \qquad (3.4)$$

From (2.2) , $ef(x') = t$ (3.5)

From (2.3), s_i= (t+d) mod (2n+1) (3.6)

For *d=0,* from (3.6), $s_i= t$, then from (3.5)

ef(x') =s_i q.e.d.

Case 2. $d \le n$. Then according to (2.3.1), (2.3.2)

$$x'i = \begin{cases} xi+1,\ i=d \\ \ \ xi,\ i \neq d \end{cases} \qquad i = 1,2,..n \qquad (3.7)$$

From (2.4), (3.7),

$$ef(x') = \sum_{i=1}^{n}(x'_i\,.\,i)\ mod(2n+1\,)$$

$$= (\sum_{i=1}^{d-1} x_i\,.\,i + d(x_d+1) + \sum_{i=d+1}^{n} x_i\,.\,i\,)\ mod(2n+1)$$

$$= (\sum_{\mathrm{i=1}}^{\mathrm{n}} x_i\,.\,i\ + 1.d)\ mod(2n+1)$$

$= (t+d)mod(2n+1)$, then from (2.5)

$ef(x') = s_i$ q.e.d.

Case 3. *n*< *d<2n*. Let

$$d' = (2n+1-\mathrm{d}),\ d' \in \{1,\dots,n\} \tag{3.8}$$

$$d = (2n+1-d')mod(2n+1)$$

$$= -d^{'}\,mod(2n+1) \tag{3.9}$$

From (2.3.3),

$$x'i = \begin{cases} xi-1,\ i = d' \\ xi,\ i \neq d' \end{cases} \qquad i = 1{,}2,..n \tag{3.10}$$

Then, from (2.4), (3.9), (3.10)

$$ef(x') = \sum_{\mathrm{i=1}}^{\mathrm{n}} (x'_i\,.\,i\,)\ mod(2n+1)$$

$$= (\sum_{\mathrm{i=1}}^{d\prime-1} x_i\,.\,i + d'(x_{d'}-1) + \sum_{i=d'+1}^{n} x_i\,.\,i)\ \ mod(2n+1)$$

$$= (\textstyle\sum_{i=1}^{n}\ x_i\,.\,i\ - 1.d')mod(2n+1) \tag{3.11}$$

$$= (\textstyle\sum_{\mathrm{i=1}}^{\mathrm{n}}\ x_i\,.\,i\ + 1.d\ )mod(2n+1)$$

$$= (t+d)mod(2n+1) \tag{3.12}$$

Then, from (2.3), (3.12),

$ef(x') = s_i$ q.e.d.

3.1 Data Structure and Justification of GEMD Correctness

In GEMD, (*n*+1) bits are embedded in *n* adjacent pixels and at least one-pixel value in each group could be changed.

3.2.1 Data Structure for GEMD Embedding

Input structures:

1. Integer, *n, n>0,* defining pixel group and bit block size.

2. Binary message S sized $|S|$ bits. It is divided into blocks B_i sized $|B_i|=L$, where $L=n+1$ bits, $i=0,1,\ldots\ldots H-1,$ and H is the number of blocks defined as in Eq.(3.1). The last block is padded by zeros when $|B_{H-1}| \neq L$, $S=S+$ zeros $(L-|S| \bmod L)$.

3. Grayscale cover images $CI_j\,[M,N]$, where M is the number of rows and N is the number of columns, $CI_j(r,c) \in [0,1,\ldots\ 255]$, $0\leq r\leq M-1,$ $0\leq c\leq N-1,$ $0\leq j\leq \tilde{N}-1.$ $\tilde{N}$ is the number of cover images that is defined by Eq. (3.2) , and is represented by one-dimensional array as in Figure 3.1.

Output: Stego images SI_j [M,N], with embedded message S , $0\leq$ j$\leq$ Ñ-1 . Data structure for GEMD embedding is illustrated by Figure 3.4.

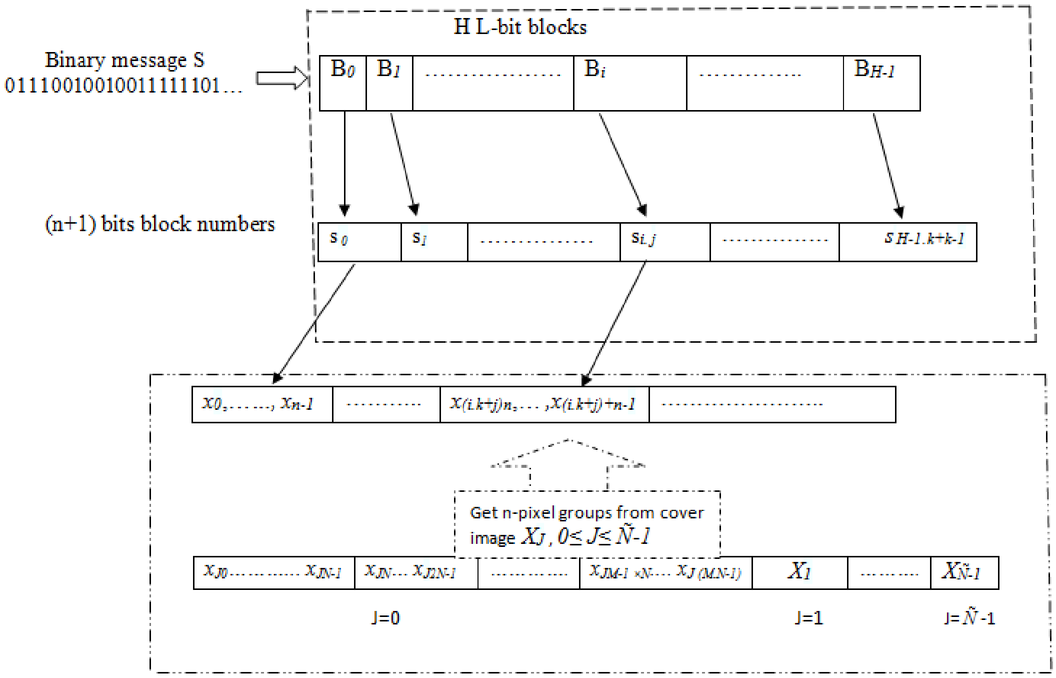


Figure 3.4: GEMD data structure for embedding procedure

3.2.2 Data Structure for GEMD Extraction

Input structures:

1. Integer, *n, n>0,* defining pixel group and bit block size.

2. Grayscale stego image SI_j $[M,N]$, where M is the number of rows and N is the number of columns, $SI_j(r,c) \in [0,1... 255]$, $0\leq r\leq M\text{-}1,\ 0\leq c\leq N\text{-}1,\ 0\leq j\leq \tilde{N}\text{-}1.$

Output: Binary secret message, S. Figure 3.5 shows data structure for GEMD extraction.

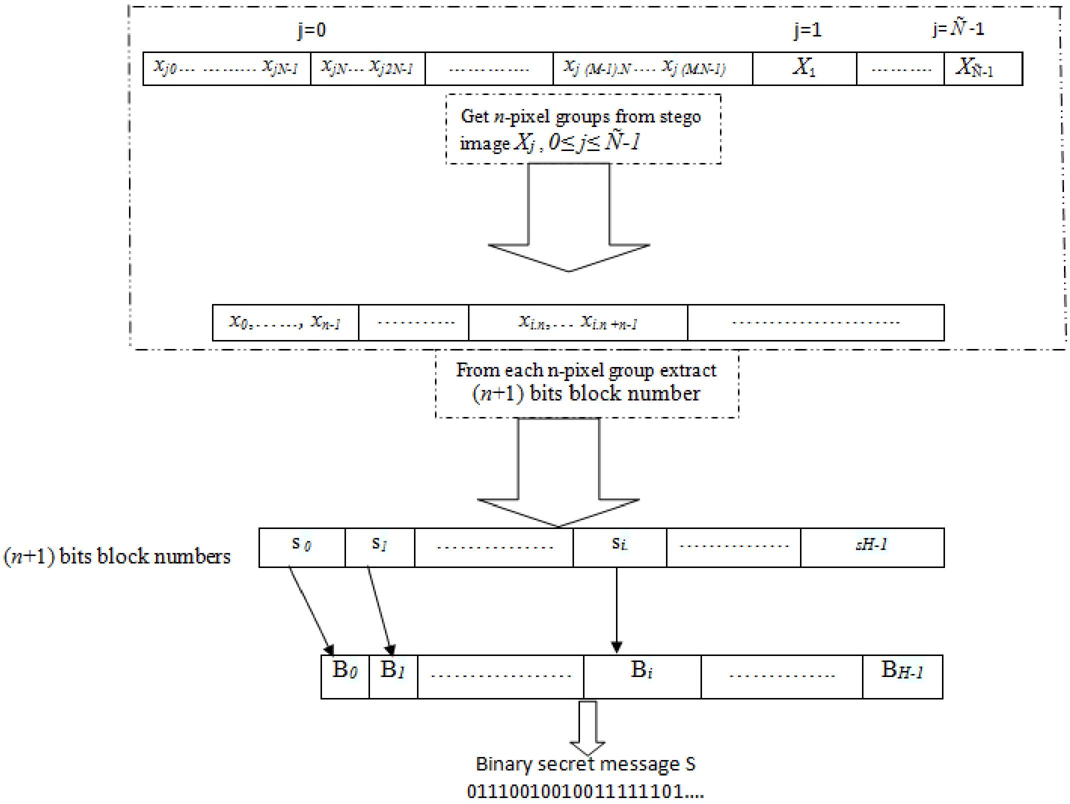


Figure 3.5: GEMD data structure for extraction procedure

3.2.3 Justification of GEMD Correctness

Statement 2.

Let's apply GEMD embedding algorithm with parameter n from Section 2.3.2, resulting in Ñ modified stego images with embedded secret message S. Then, application of GEMD extraction algorithm to these stego images results in original secret message S.

Statement 2 Justification.

Let's consider next n pixel block from the stego image $X'=(x'_1,\ldots x'_n)$ which was obtained by embedding of (n+1)-bit block number s from the input secret binary stream S, by Step3-Step6 transformation of the original stego image block $X=(x_1,\ldots x_n)$.

From (2.7)

$$ef(x') = \sum_{i=1}^{n} x'_i\,.\left(2^i - 1\right) mod\ 2^{n+1} \tag{3.13}$$

Consider in (2.6) d=2^n, then from Step 6, Case1, we have

$$x'_n = x_n + 1,\ x'_1 = x_1 + 1,\ i=1,\ldots,n \tag{3.14}$$

Then, from (3.13), (3.14)

$$ef(x') = ((x_1+1)(2^1-1) + \sum_{i=2}^{n-1} x_i\,.\left(2^i-1\right) + (x_n+1)(2^n-1))mod\ 2^{n+1}$$

$$= (\sum_{i=1}^{n} x_i\,.\left(2^i-1\right) + 1.(2^1-1) + 1.(2^n-1))mod\ 2^{n+1}$$

$$= (\sum_{i=1}^{n} x_i\,.\left(2^i-1\right) + 2^n\,)\ mod\ 2^{n+1} \tag{3.15}$$

From (2.5), (2.6), (3.15), we get

s= (t+d) mod 2^{n+1} = $ef(x')$, q.e.d.

Consider in (2.6), d<2^n.

Let binary representation of d is as follows:

$$d=\sum_{i=0}^{n} di\,.\,2^i,\ \text{and}\ d_n=0 \tag{3.16}$$

If, for simplicity,

$$d=2^k,\ k<n \tag{3.17}$$

Then, according to Step6, Case2,

$$x'_{k+1}=x_{k+1}+1 \text{ (increase left neighbor)} \tag{3.18}$$

$$x'_k=x_k -1 \text{ (decrease current position)} \tag{3.19}$$

Then from (2.7), (3.18), (3.19)

$$ef(x') = \sum_{i=1}^{n} x'_i \,.\, (2^i - 1) mod\ 2^{n+1}$$

$$= (\sum_{\mathrm{i}=1}^{\mathrm{k}-1} x_i\,(2^i - 1) + (x_k - 1)(2^k - 1) + (x_{k+1} + 1)(2^{k+1} - 1)$$

$$+ \sum_{i=k+2}^{n} x_i\,(2^i - 1)\,)\ mod(2^{n+1})$$

$$=(\sum_{i=1}^{n} x_i \,.\, (2^i - 1) - 2^k + 1 + 2^{k+1}\ - 1\,) mod\ 2^{n+1}$$

$$=(\ ef(x) + 2^k) mod 2^{n+1} \qquad (3.20)$$

From (2.5), (2.6), (3.17), (3.20),

$ef(x') = (ef(x) + d) mod 2^{n+1} = (t + d) mod 2^{n+1} = s$.

Thus, GEMD works correctly in the case when binary representation of *d* has just one 1. In the case of several consecutive ones in the binary representation of *d*, let,

d_{k+1}=0, d_k=d_{k-1}=.....=d_j=1, d_{j-1}=0, $j<k$. (3.21)

Applying for each d_i =1,*i=j,...,k*, considered above embedding, i.e. $x'_{i+1}=x_{i+1}+1$, $x'_i=x_i$ -1, *i=j,..,k*, we see that as far as each x'_i; *i=j+1,..., k*, is modified twice, once increased as a left neighbor, and once decreased as being in the current position, hence ultimately, only x'_{k+1} ,and x'_j are modified:

$x'_{k+1}=x_{i+1}+1$, $x'_j=x_j$ -1, that just corresponds to Case 2 modifications of Step6 in the GEMD Embedding algorithm. q.e.d.

For example, we have n=5 pixels in a group, and d is as follows in binary:

	x_5	x_4	x_3	x_2	x_1	*pixels*
d_5	d_4	d_3	d_2	d_1	d_0	
0	*1*	*1*	*1*	*0*	*0*	*Binary values*
	k=4		*J=2*			

Then we have 3 consecutive ones in positions 4,..,2,k=4, j=2 in (3.21). According to (3.18), (3.19), the first change from 0 to 1 is at position $k=4$, so

$$x'_5=x_5+1,\ x'_4=x_4-1 \tag{3.22}$$

Then $d_3=1$, and by (3.18), (3.19),

$$x'_4=x_4+1,\ x'_3=x_3-1 \tag{3.23}$$

Then $d_2=1$, and by (3.18), (3.19),

$$x'_3=x_3+1,\ x'_2=x_2-1 \tag{3.24}$$

Then we get from (3.22), (3.23), (3.24)

$x'_5=x_5+1$

$x'_4=x_4-1$ *and* $x'_4=x_4+1$, *so* $x'_4=x_4$ (3.25)

$x'_3=x_3-1$ and $x'_3=x_3+1$, so $x'_3=x_3$

$x'_2=x_2 - 1$

Hence for this case of k=4, j=2, we have ultimately for (3.25) $x'_5=x_5+1$, $x'_2=x_2 - 1$, $x'_i=x_i$, $i=j+1,....,k$, that complies with Step6, Case2 of the GEMD Embedding algorithm. q.e.d.

Consider in (2.6), $d>2^n$,

$d' = 2^{n+1} - d < 2^n$, so, $d = -d' \ mod\ 2^{n+1}$, and in binary

$d'= (d_n\ d_{n-1}\ d_{n-2}...\ d_1\ d_0\ )_2$, i.e.

$$d'= \sum_{i=0}^{n} d'i\ .\ 2^i < 2^n \tag{3.26}$$

Let

$d'=2^k$, k<n (3.27)

Then similar to the Case2, but now considering according to Step6, Case3 of the GEMD Embedding algorithm, we used decreasing, and $x'_{k+1}=x_{k+1}$ -1, $x'_k=x_k$+1. Then we have from (2.7), (3.27)

$$ef(x') = \sum_{i=1}^{n} x'_i \,.\left(2^i - 1\right) mod\ 2^{n+1}$$

$$= \left(\sum\nolimits_{\mathrm{i=1}}^{\mathrm{k-1}} x_i\left(2^i - 1\right) + (x_k + 1)(2^k - 1) + (x_{k+1} - 1)(2^{k+1} - 1) + \sum_{i=k+2}^{n} x_i\left(2^i - 1\right)\right) mod 2^{n+1}$$

$$=(\sum\nolimits_{i=1}^{n} x_i\,.\left(2^i - 1\right) + 2^k - 1 - 2^{k+1} + 1\,) mod\ 2^{n+1}$$

$$= (ef(x) - 2^k) mod 2^{n+1}$$

$$= (ef(x) - d') mod 2^{n+1} \qquad (3.28)$$

As we have $d = -d'\ mod\ 2^{n+1}$, then from (2.7), (2.8), (3.28),

$ef(x') = (ef(x) + d) mod 2^{n+1} = s$

In the case of several consecutive ones appearance in d' is considered just as in the Case2. q.e.d.

3.2 Summary of Chapter 3

Thus, in this chapter we have considered details of data structures for input, output of EMD and GEMD that is necessary for their implementation and proved correctness of them. In the next Chapter 4 the implementation of the methods will be discussed.

Chapter 4

IMPLEMENTATION OF THE EMD AND GEMD

In this chapter, we will show the implementation of EMD and GEMD schemes. In testing the algorithms, a personal computer with the following characteristics was used; CPU: Intel ®Core (MT) i3 3210M 2.10 GHz, with a memory of 2GB, Windows 7 operation system, and MATLAB 2013 was used for simulation.

4.1 EMD Implementation

In EMD implementation, four gray scale secret images with same size 512×512 pixels are used; also the cover images are in the size 512×512. First, we need to specify the number of pixels *n* in each group of cover image, and then the number of bits *L* in each block of secret message which will be represented by *k*-digit (2*n*+1)-ary number, where *k* is calculated in the main program as follows (full code is in Appendix A.1):

```
addpath('cover_set/');addpath('secret_set/');
img_name = 'P';sec_name='S';
M=512; N=M;
L=input('Input L: the number of bits in a block ');
n=input(' Input n: the number of pixels in a group ');
k=ceil(L/(log2(2*n+1)))
Bpp=(log2(2*n+1))/n
Sec = imread([sec_name, '',num2str(1) '.jpg']);
S =reshap_im(sec,M,N);
[Bin]= conv2binary(S);
s_size=numel(Bin)
SS=[Bin zeros(1,(L-(mod(s_size,L))))]
H=ceil(s_size /L);
Cover_im = ceil((H*k)/C);
```

In line 3 we specify the size of cover image and secret image, rows *M,* and columns *N* as 512×512. In line 6 we calculate the number of digits *k* as (2.1), and in line 7 we calculate the embedding capacity BPP as in (2.10). In line 8 we read the first secret image as we have four gray scale secret images and in line 9 we reshape it into one dimensional array by reshap_im function (A.2). In line 10, we convert each pixel of the secret image into binary by calling conv2binary function (A.3). In line 12 last block of binary secret message may padded by zeros, and in lines 13, 14, the number of blocks *H* and the number of cover image are calculated as (3.1), (3.2) respectively, then the result is as follows.

```
Input L: the number of bits in a block    16
Input n: the number of pixels in a group    2
k = 7
H= 131072
Cover_im = 7
```

Necessary for *CI* number formulas were derived in ch3. In the main program (A.1) the functions are called as follows

```
1. Covers = uint8( zeros(M,N,Cover_im) );
2. Stegos = uint8( zeros(M,N,Cover_im) );
3.  [Dig] = BTO2NP1(SS,L,k,n,H );
4. h=1;
5. for i=1:Cover_im
6. tic
7. CI = imread([img_name, '',num2str(i) '.tif']);
8. Covers(:,:,i)=CI;
9. ci1 =reshap_im(CI,M,N);
10. x=1;
11. for r=1:C
12. group= ci1((x-1)*n+1:x*n);
13. [em_group]=embed(group,Dig(h),n);
14. ci1((x-1)*n+1:x*n)=[em_group];
15. h=h+1;
16. x=x+1;
17. end
18. ci2 =reshap_im2(ci1,M,N);
```

```
19. Stegos(:,:,i)=ci2;
20. tim(i)=toc
21. mem= memory
22. end
```

In line 3 we convert each L-bit block of secret message *SS* into *k*-digit (2*n*+1)-ary number by BTO2NP1 function (A.4). In lines 6 and 20 we calculate the time consumption in seconds with a function that starts timer, tic, in line 6, and stop it by toc in line 20 as one cover image is embedded completely, while in line 21 memory consumption is calculated in MB. As we get *n*-pixel group from cover image in line 12, one digit is embedded each time in *n*-pixel group by embed function given in Appendix A.5, according to description in Section 2.3.1 as follows

```
function [em_group] = embed(group, Dig,n)
sum=0;
  for i=1:n
  sum =sum +double(group(i))*i   ;
  end
t=mod(sum,(2*n+1));
d=mod((Dig -t),(2*n+1));
if (d<=n && d>0)
   group(d)=(group(d))+1;
elseif (d>n)
 group(((2*n+1))-d)=(group(((2*n+1))-d))-1;
end
em_group= group;
end
```

From main program code, in line 18, we reshape stego image into two dimensional array as in Appendix A.11, and then in line 19 it is saved in array of set stego images that presented by the code in Appendix A.6. The results are available in Appendix A.8.1.

```
disp('=================================================')
disp('stego image  PSNR  MSE   Time  Memory  Capacity')
disp('              dB          sec    MB       bpp   ')
disp('================================================')
set(gcf, 'name', ' Secret Image in case n=2');
 for i=1:Cover_im
subplot(3,3,i) ;  imshow((Stegos(:,:,i)));
[PSNR(i), MSE(i)]=My_PSNR(Covers(:,:,i),Stegos(:,:,i));
title(['PSNR = ',num2str(PSNR) ]);
sprintf('%s%f%f%f%f%f',images{i},PSNR(i),MSE(i),tim(i),me
m,Bpp)
disp('=================================================')
sum_time=sum_time+tim(i);
sum_psnr=sum_psnr+PSNR(i);
sum_mse=sum_mse+MSE(i);
 end
psnr_avg=sum_psnr/Cover_im
mse_avg=sum_mse/Cover_im
tim_avg=sum_time/Cover_im
sprintf('Average ')
disp('=================================================')
sprintf('%f%f %f%f %f',psnr_avg,mse_avg,tim_avg,mem,Bpp)
```

Image quality PSNR (2.9) and MSE (2.8) are calculated in the following function given in Appendix A.7:

```
function [ My_psnr MSE ] = My_PSNR(I,J)
    X = double(I);
    Y = double(J);
    MSE = sum((X(:)-Y(:)).^2) / prod(size(X)) ;
    My_psnr = 10*log10(255 * 255/MSE);
End
```

As a sample output of our implementation, the results for first gray scale secret image (Balloon) that embedded in 7 cover images when *n*=2, *L*= 16 bits are shown Figures 4.1 and 4.2.

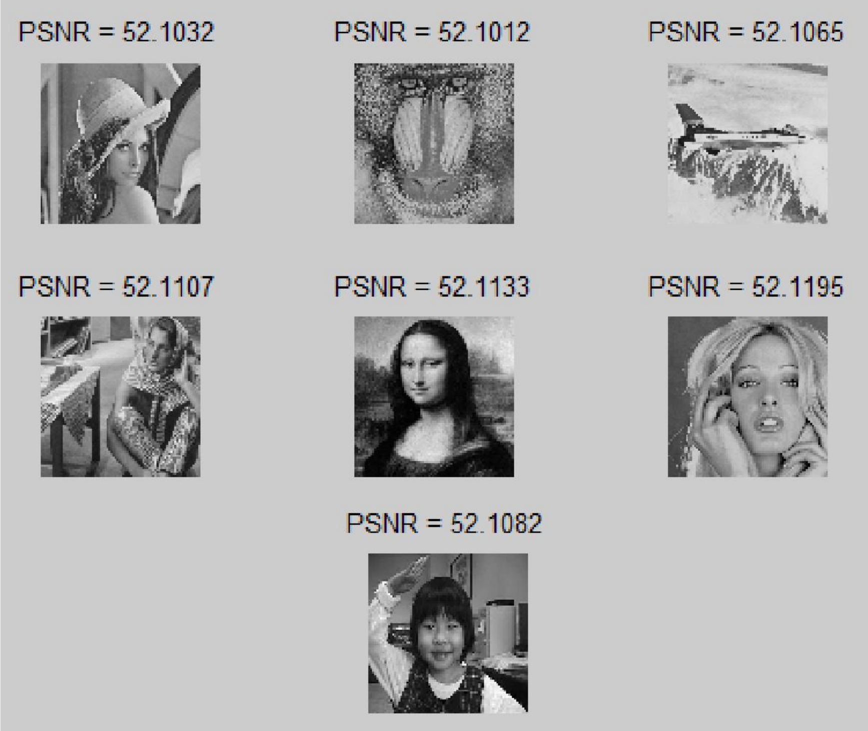
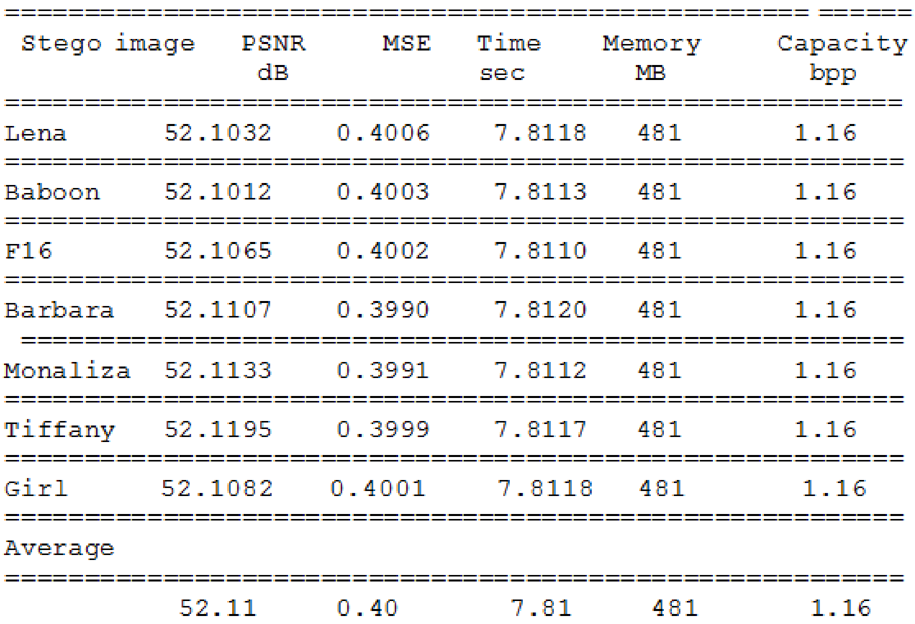


Figure 4.1: Seven stego images in EMD implementation Appendix A.8.1 (1) Lena; (2) Baboon.; (3) F16; (4) Barbara**.** (5) Monaliza; (6) Tiffany; (7) Girl.



```
=================================================== ======
 Stego image   PSNR      MSE    Time     Memory      Capacity
                dB              sec        MB          bpp
==========================================================
Lena         52.1032    0.4006    7.8118    481         1.16
==========================================================
Baboon       52.1012    0.4003    7.8113    481         1.16
==========================================================
F16          52.1065    0.4002    7.8110    481         1.16
==========================================================
Barbara      52.1107    0.3990    7.8120    481         1.16
 =========================================================
Monaliza     52.1133    0.3991    7.8112    481         1.16
==========================================================
Tiffany      52.1195    0.3999    7.8117    481         1.16
==========================================================
Girl        52.1082    0.4001     7.8118    481          1.16
==========================================================
Average
==========================================================
              52.11      0.40          7.81      481          1.16
```

Figure 4.2: EMD implementation results in case *n*=2. Appendix A.8.1

For extraction stage in the following code, as we extract binary message from EXTRACTION function, Appendix A.10, we convert each 8 bits to decimal to represent a pixel, and then we reshape the steam of numbers into two dimensions, Appendix A.11, to get our secret message as shown in Figure 4.3, and in the next code, Appendix A.9.

```
for j=1:Cover_im
    Stegos1(1,:,j)=reshap_im(Stegos(:,:,j),M,N);
end
    [secret_message]= EXTRACTION( Stegos1,k,n,H,L );
     v=1;
for i=1:M*N
     bmess=secret_message((v-1)*8+1:v*8);
     a=bin2dec(num2str(bmess));
     d_msg=[d_msg a];
     v=v+1;
 end
   secret_im=reshap_im2(d_msg,M,N);
   set(gcf,'name',' Extracted Secret image');
   imshow(uint8(secret_im));
```

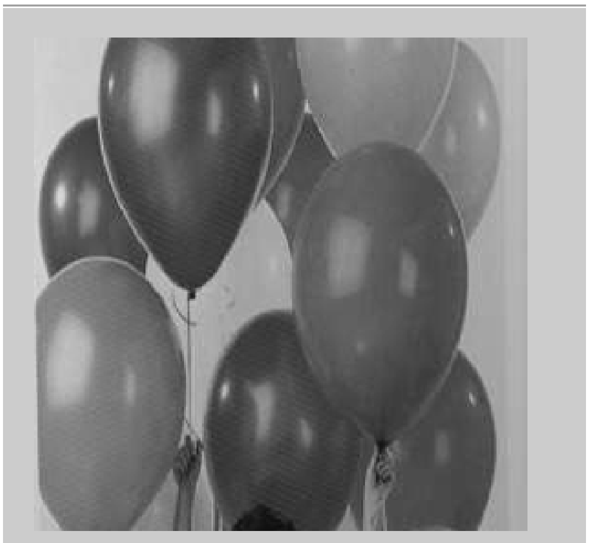


Figure 4.3: Extracted Secret image

4.2 GEMD Implementation

In this method no need for transformation, so L= (*n*+*1*) bits are embedded in *n* pixel group. Use already defined formulas. After specifying *n* and *L* in the main program, Appendix B.1, we read the secret image and reshape it into one dimensional array, Appendix B.2, then convert it to binary by conv2binary function, Appendix B.3. Binary secret message stream is divided then into (*n*+1)-bit blocks and then to decimal numbers by next function, Appendix B.4:

```
function [Num] = GET_B(S,L,H)
    Num=[];
    for i=1:H
    B= S((i-1)*L+1:i*L);
    d=bin2dec(num2str(B));
    Num=[Num d];
    end
end
```

Embedding function, Appendix B.5, is as follows.

```
function [em_group ] = GEMDembed( group,num,n )
sum=0;
for i=1:n
sum = sum + double( group(i)) *((2^i)-1);
end
t=mod(sum,(2^(n+1)));
d=mod(num -t,(2^(n+1)));
if (d==2^n) R=1;
elseif(d<(2^n)) R=2;
else R=3;
end
switch R
case 1
group(n)= group(n)+1;
group(1)= group(1)+1;
case 2
d=dec2bin(d,(n+1));
for i=0:n-1
if ((d(i+1)=='1')&&(d(i+2)=='0'))
group(n-i)=group(n-i)-1;
elseif ((d(i+1)=='0')&&(d(i+2)=='1'))
group(n-i)=group(n-i)+1;
end
end
case 3
d=(2^(n+1))-d;
```

```
26 b=dec2bin(d,(n+1));
27 for j=0:n-1
28 if((b(j+1)=='1')&&(b(j+2)=='0'))
29 group(n-j)= group(n-j)+1;
30 elseif((b(j+1)=='0')&&(b(j+2)=='1'))
31 group(n-j)= group(n-j)-1;
32 end
33 end
34 end
35 em_group= group;
end
```

In lines 1-5, we calculate the extraction function (2.5) described in Section 2.3.2, and (2.6) in line 6. Then lines 7- 10 are the Step 5 in GEMD embedding algorithm. Step 6, Case 1 is implemented in the lines 12-14, and in lines 15-23, we apply Step 6, Case 2; lines 24-35 are the Step 6, Case3 in GEMD embedding algorithm. Here in lines 16-17 and lines 26-27 we did not change the positions of *d* bits, since Matlab starts from left to right, it means from most to least significant bit as we have in the GEMD embedding algorithm.   GEMD results are presented by the following code, Appendix B.6, PSNR (2.9), MSE (2.8), GEMD embedding capacity BPP (2.11), time and memory consumption, Appendix B.1, results are available in Appendix B.8.1.

```
disp('=================================================')
disp('stego image  PSNR  MSE   Time  Memory  Capacity')
disp('              dB          sec    MB       bpp   ')
disp('================================================')
set(gcf, 'name', ' Secret Image in case n=2');
 for i=1:Cover_im
subplot(2,3,i) ;  imshow((Stegos(:,:,i)));
[PSNR(i), MSE(i)]=My_PSNR(Covers(:,:,i),Stegos(:,:,i));
title(['PSNR = ',num2str(PSNR(i))]);
sprintf('%s%f%f%f%f',images{i},PSNR(i),MSE(i),tim(i),me
m,Bpp)
disp('=================================================')
sum_time=sum_time+tim(i);
sum_psnr=sum_psnr+PSNR(i);
```

```
sum_mse=sum_mse+MSE(i);
 end
psnr_used=(sum_psnr- PSNR(Cover_im))/(Cover_im-1);
mse_ used =(sum_mse - MSE(Cover_im))/(Cover_im-1);
tim_ used =(sum_time- tim(Cover_im))/(Cover_im-1);
psnr_set=sum_psnr/Cover_im;
mse_set=sum_mse/Cover_im;
tim_set=sum_time/Cover_im;
sprintf(' Average on fully used  ')
disp('=================================================')
sprintf('%.2f%.2f%.2f%.2f%.2f',psnr_used,mse_used,tim_use
d, mem,Bpp)
sprintf(' Average on fully set  ')
disp('=================================================')
sprintf('%.2f%.2f%.2f%.2f%.2f',psnr_set,mse_set,tim_set,m
em,Bpp)
```

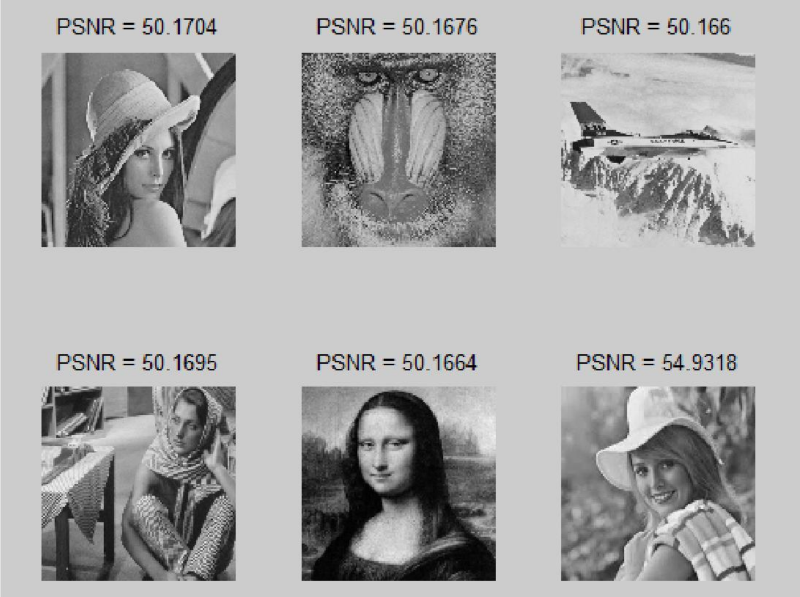
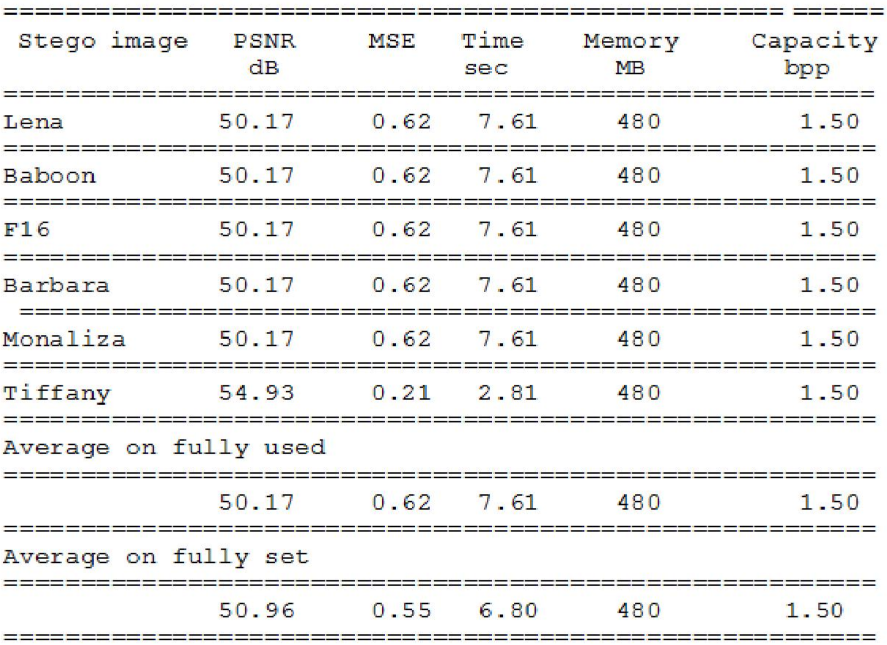


Figure 4.4: Six stego images in GEMD implementation. Appendix B.8.1



```
========================================================= ======
 Stego image    PSNR      MSE    Time      Memory      Capacity
                 dB               sec        MB           bpp
================================================================
Lena            50.17     0.62   7.61      480             1.50
================================================================
Baboon          50.17     0.62   7.61      480             1.50
================================================================
F16             50.17     0.62   7.61      480             1.50
================================================================
Barbara         50.17     0.62   7.61      480             1.50
 ===============================================================
Monaliza        50.17     0.62   7.61      480             1.50
================================================================
Tiffany         54.93     0.21   2.81      480             1.50
================================================================
Average on fully used
================================================================
                50.17     0.62   7.61      480             1.50
================================================================
Average on fully set
================================================================
                50.96     0.55   6.80      480           1.50
================================================================
```

Figure 4.5: GEMD implementation results in case *n*=2. Appendix B.8.1

As a sample output of our implementation in Figures 4.4 and 4.5, we see that last image has greater PSNR and less MSE since fewer pixels are changed because it was not embedded completely, where number of cover image is calculated according to (3.2). Also from Figure 4.5 the average on fully used is taken for the first 5 images that are fully embedded, while the average on fully set is taken for all stego images, since the last image not fully embedded. In the extraction code in main program, Appendix B.9, we call the extraction function, Appendix B.10, and then convert each 8 bits to decimal to represent a pixel, then reshape into two dimensions, Appendix B.4, to get our secret message. Next is the extraction function code, Appendix B.10:

```
function [B_msg]=  EXTRACTION(Stegos1,H,n )
B_msg=[];
   for i=1:H
       group= Stegos1((i-1)*n+1:i*n);
       sum=0;
     for j=1:n
       sum = sum + double( group(j)) *((2^j)-1);
      end
  t=mod(sum,(2^(n+1)));
  bin=dec2bin(t,n+1);
  B_msg=[B_msg bin];
   end
  B_msg;
 end
```

4.2 Summary of Chapter 4

Thus, in this chapter we have implemented and explained EMD and GEMD codes as we take *n*=2 pixels as an example in both methods, and the results in this case were shown. Furthermore the results for different values of *n* and full codes for all functions in both methods are available in Appendix.

Chapter 5

SIMULATION AND RESULTS

In this chapter we discuss the results of EMD and GEMD simulations for different number of pixels *n* in a group used in the cover images.

5.1 EMD Simulation

Gray scale secret images of size 512×512 used in the experiments are shown in Figure 5.1. Also cover images used in the experiments are shown in the Figure 5.2. To investigate the effect of using different sizes, the results are taken for two cover image sizes 512×512 and 1024×1024.

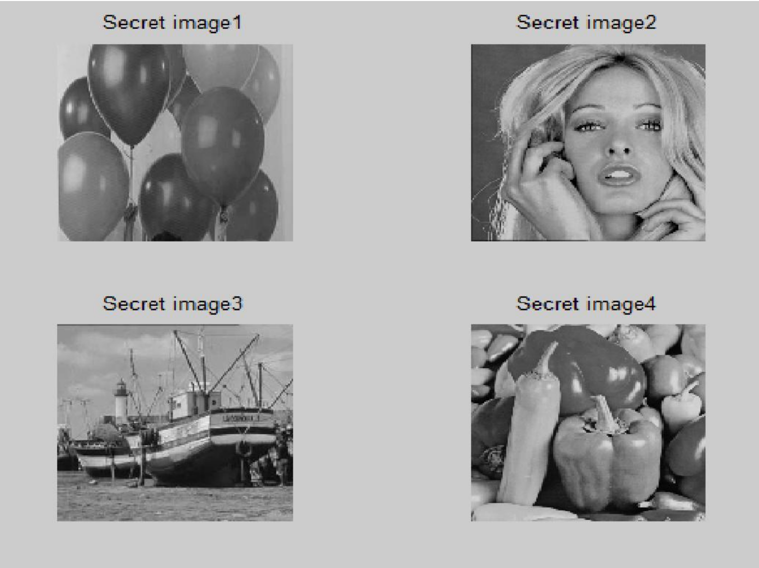


Figure 5.1: Gray scale secret images (1) Balloon; (2) Tiffany; (3) Boat; (4) Pepper

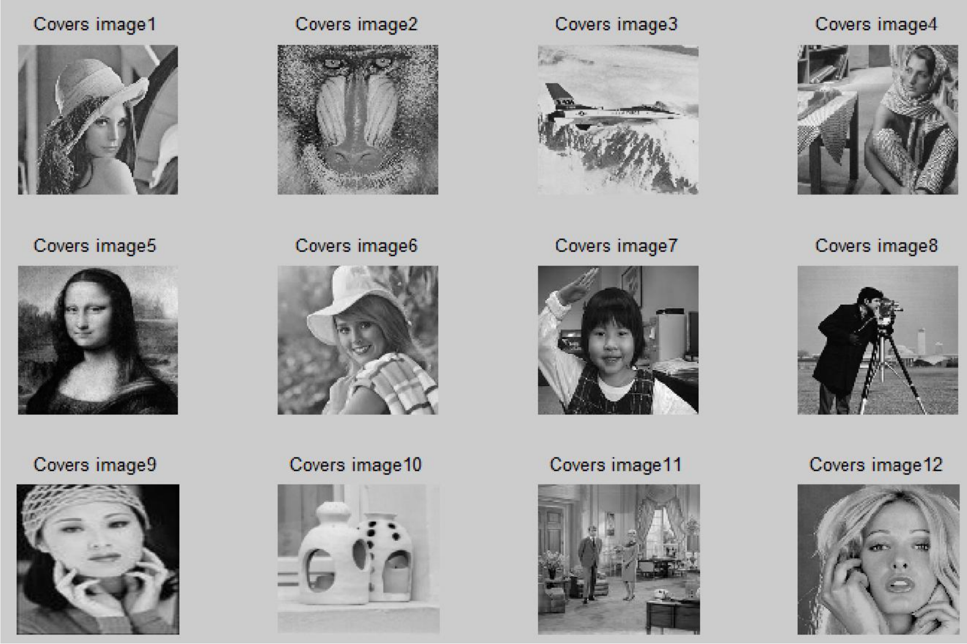


Figure 5.2: Gray scale cover images used in EMD and GEMD simulation

EMD optimal parameters used in the simulations are given in Table 5.1. Where *k* is calculated as (2.1), and number of cover images Ñ as (3.2).

Table 5.1: EMD parameters of the simulation with 512×512 cover images

parameter	Number of pixels *n* for one digit			
	2	3	4	5
L bits	16	16	32	64
k digits	7	6	11	19
cover images Ñ	7	10	11	12

From Table 5.1, and for *n*= 2, L= 16, and *k* = 7, where *k* is calculated as in (2.1) then number of cover images Ñ necessary for one secret image is calculated as follows

First, we find the number of blocks H, according to (3.1) $H = \left\lceil \frac{|S|}{L} \right\rceil$

Where |*S*| is the size of binary secret image *S,* |*S*| = 512×512×8=2097152 bits

$$H = \left\lceil \frac{2097152 \text{ bits}}{16\, bits} \right\rceil = 131072\, blocks$$

From (3.2)

$$\tilde{N} = \left\lceil \frac{H \times k}{C} \right\rceil = \left\lceil \frac{131072 \times 7}{C} \right\rceil$$

From (3.3), $C = \left\lfloor \frac{M \times N}{n} \right\rfloor = \left\lfloor \frac{512 \times 512}{2} \right\rfloor$=131072 *n*-pixel groups

Then,

$$\tilde{N} = \left\lceil \frac{131072 \times 7}{7} \right\rceil = 7 \text{ cover images.}$$

For *n*=3 pixels, L= 16 bits, and *k* = 6 digits, then

$$H = \left\lceil \frac{2097152 \text{ bits}}{16\, bits} \right\rceil = 131072\, blocks$$

$$\tilde{N} = \left\lceil \frac{131072 \times 6}{\left\lfloor \frac{512 \times 512}{3} \right\rfloor} \right\rceil = 10 \text{ cover images}$$

For *n*=4 pixels, L= 32 bits, and *k* = 11 digits, then

$$H = \left\lceil \frac{2097152 \text{ bits}}{32\, bits} \right\rceil = 65536\, blocks$$

$$\tilde{N} = \left\lceil \frac{65536 \times 11}{\left\lfloor \frac{512 \times 512}{4} \right\rfloor} \right\rceil = 11 \text{ cover images}$$

For *n*=5 pixels, L= 64 bits, and *k* = 19 digits, then

$$H = \left\lceil \frac{2097152 \text{ bits}}{64\, bits} \right\rceil = 32768\, blocks$$

$$\tilde{N} = \left\lceil \frac{32768 \times 19}{\left\lfloor \frac{512 \times 512}{5} \right\rfloor} \right\rceil = 12 \text{ cover images}$$

For 1024×1024 cover image, and with the same EMD parameters (*L* and *k*) in Table 5.1, we got different numbers of cover images required for one secret image using (3.1), (3.2), and (3.3) in each case of *n* as shown in Table 5.2.

Table 5.2: EMD parameters of the simulation for 1024×1024 cover images

parameter	Number of pixels *n* for one digit			
	2	3	4	5
L bits	16	16	32	64
k digits	7	6	11	19
cover images Ñ	2	3	3	3

From Table 5.2 we note that as we increase cover image size then we need less number of cover images. Table 5.3 shows the EMD average results using 512×512 cover image size for different values of *n* that obtained from the Appendix A.8. As in some cases we have two averages, fully used and fully set, where the average fully used indicates to the averages of PSNR, MSE, memory and time consumption for the fully images used (without the last image that not fully embedded), while the average fully set refers to the averages for the fully images set (with the last image that not fully embedded).

Table 5.3: EMD average results for 512×512 cover images

Metric	average	Number of pixels *n* for one digit			
		2	3	4	5
PSNR (dB)	Fully used	52.11	53.57	54.66	55.53
	Fully set		58.45		55.70
MSE	Fully used	0.40	0.28	0.22	0.18
	Fully set		0.25		0.17
Time (sec)	Fully used	7.81	6.38	4.66	3.97
	Fully set		5.76		3.82
Memory (MB)		481	486	490	496
Capacity (BPP)		1.16	0.93	0.79	0.69

Table 5.4 shows the EMD average results for 1024×1024 cover images, the results are obtained from Appendix A.8.

Table 5.4: EMD average results for 1024×1024 cover images

Metric	average	Number of pixels *n* for one digit			
		2	3	4	5
PSNR (dB)	Fully used	52.11	53.57	54.66	55.53
	Fully set	52.73	55.58	55.08	55.68
MSE	Fully used	0.40	0.28	0.22	0.18
	Fully set	0.37	0.21	0.20	0.18
Time (sec)	Fully used	5.72	4.31	3.02	2.18
	Fully set	5.36	4.14	2.91	2.12
Memory (MB)		493	495	497	500
Capacity (BPP)		1.16	1.16	0.93	0.69

From Table 5.3 and Table 5.4 The comparison for metrics are taken for fully used averages since the fully set average is not found in some cases in 512×512 cover images Table 5.3. For both sizes we got the same results for PSNR and MSE since in EMD embedding algorithm only one pixel in a group could be changed by ±1, it means not depends on the size, also for embedding capacity that calculated according to (2.10)For both sizes the PSNR for fully used average is better than 52 dB as the number of pixels in the cover image increases which is illustrated in Figure 5.3.

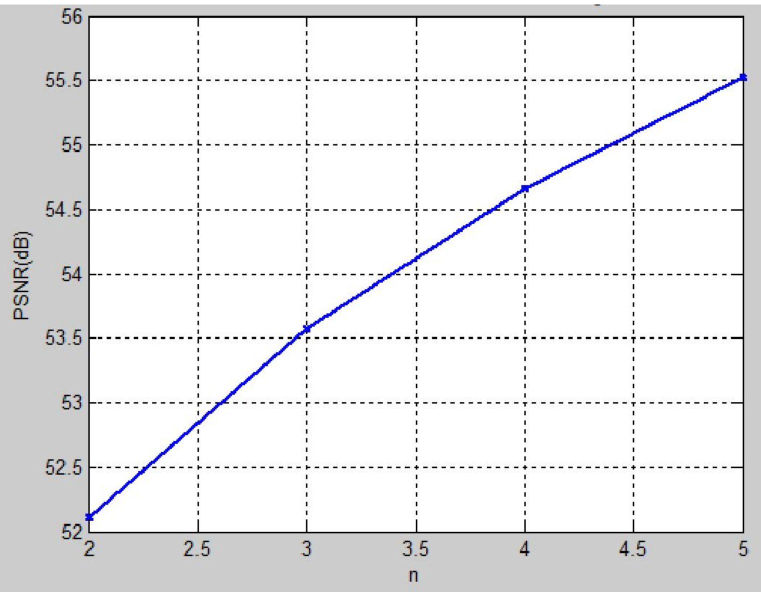


Figure 5.3: PSNR of EMD using 512×512 and 1024×1024 cover image size

Also from Table 5.3 and Table 5.4, MSE decreases from 0.40 to 0.18 as *n* increases from 2 to 5 pixels as shown in Figure 5.4.

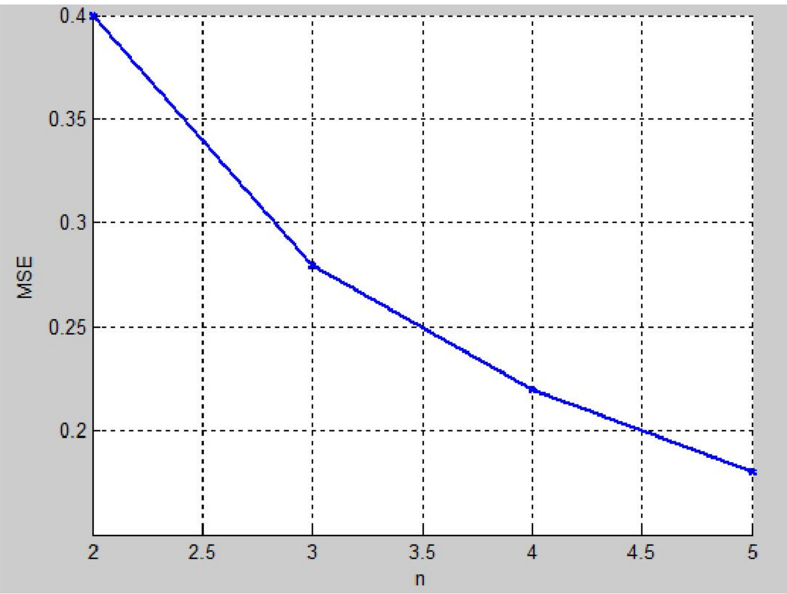


Figure 5.4: MSE of EMD using 512×512 and 1024×1024 cover image size

Figure 5.5 sows the EMD embedding capacity for both sizes in Table 5.3 and Table 5.4 that decreases from 1.16 BPP to 0.69 BPP when *n* ranges from 2 to 5 pixels.

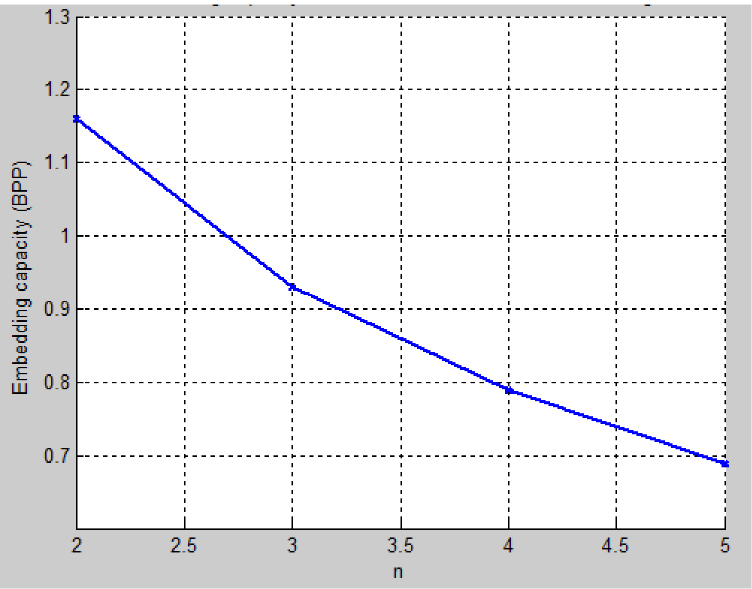


Figure 5.5: BPP of EMD using 512×512 and 1024×1024 cover image size

From Table 5.3 and Table 5.4 we note that for memory and time consumption we got different values. For time consumption, EMD using 1024×1024 cover images takes less time than using 512×512 cover images, since in using 512×512 cover images takes more cover image in each case of *n* it means more time consumption for processing data. From Table 5.3 Time consumption decreases with *n* for fully set average; it decreases from 7.81 sec to 3.97 sec, while in Table 5.4 it decreases from 5.72 sec to 2.18 sec when *n* ranges from 2 to 5 pixels as shown in Figure 5.6.

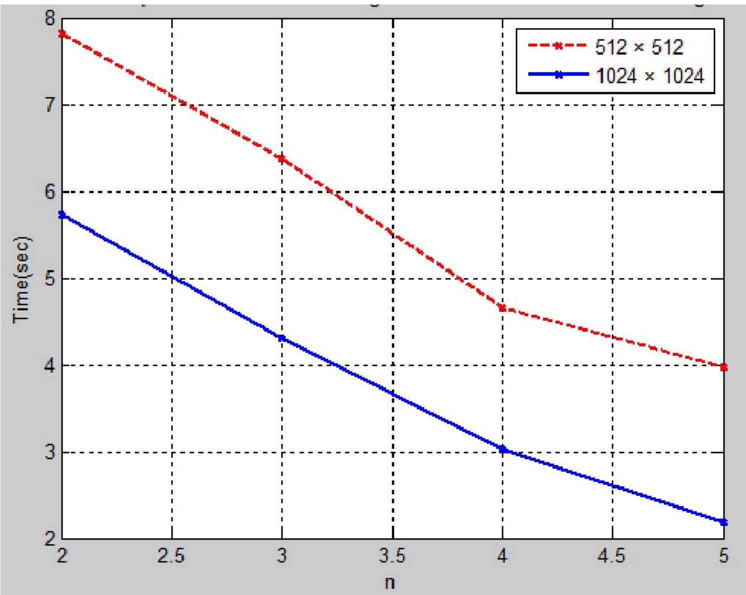


Figure 5.6: Time consumption of EMD using 512×512 and 1024×1024 cover image size

For memory consumption, EMD using 1024×1024 cover images in Table 5.4 takes more memory than using 512×512 cover images as in from Table 5.3 where memory consumption increases with *n* for fully set average; it increases 481MB to 496 MB. On the other hand using 1024×1024 cover images Table 5.4, memory consumption increases from 493 MB to 500 MB when *n* ranges from 2 to 5 pixels as shown in Figure 5.7.

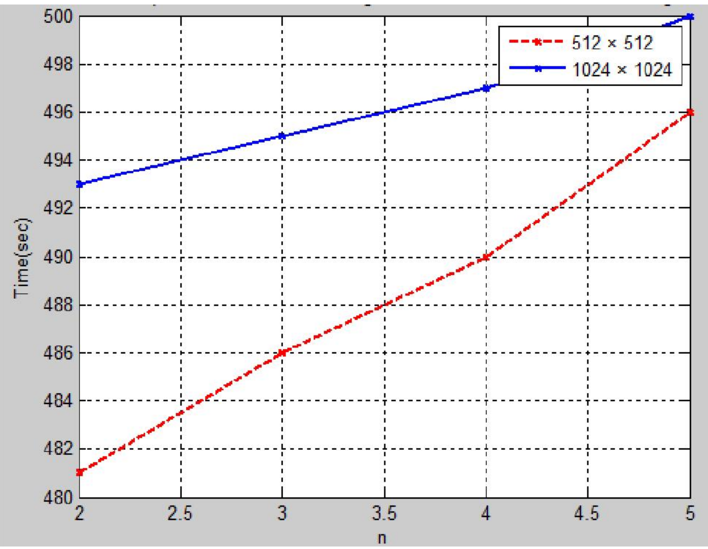


Figure 5.7: Memory consumption of EMD using 512×512 and 1024×1024 cover image size

From Table 5.3 and Table5.4 we can summarize the results in Table 5.5 as comparison results using 512×512 and 1024×1024 size of cover image for EMD method.

Table 5.5: Comparison results for EMD in two sizes of cover image

metric	EMD method	
	512×512	1024×1024
PSNR(dB)	53.97	53.97
MSE	0.27	0.27
Time (sec)	5.71	3.81
Memory (MB)	488	496
Capacity (BPP)	0.89	0.89

From Table 5.5 we find that as we use grater size of cover image then we have less time consumption by 0.33% for EMD, since in case of using 512 × 512 cover images, we need more cover images and then more time for data processing. For memory consumption, using 1024 × 1024 cover images required more memory by 0.02%. On the other hand, for both sizes we get the same results for EMD metrics PSNR, MSE, and embedding capacity BPP. The results for both cover image sizes are obtained from appendix A.

5.2 GEMD Simulation

In GEMD, we take $L=n+1$ bits to embed in n pixels. GEMD parameters used in the simulations are given in Table 5.6. Number of cover images is calculated as (3.2).

Table 5.6: GEMD parameters of the simulation for 512×512 cover images

parameter	Number of pixels *n* for one block L			
	2	3	4	5
L bits	3	4	5	6
cover image Ñ	6	7	7	7

From Table 5.3, and for *n* =2, L= 3, then

$$H = \left\lceil \frac{2097152 \text{ bits}}{3\ bits} \right\rceil = 699051\ blocks$$

$$\tilde{N} = \left\lceil \frac{699051}{\left\lceil \frac{512\times512}{2} \right\rceil} \right\rceil = 6 \text{ cover images}$$

For *n*=3 pixels, L= 4 bits, then

$$H = \left\lceil \frac{2097152 \text{ bits}}{4\ bits} \right\rceil = 524288\ blocks$$

$$\tilde{N} = \left\lceil \frac{524288}{\left\lceil \frac{512\times512}{3} \right\rceil} \right\rceil = 7 \text{ cover images}$$

For *n*=4 pixels, L= 5 bits, then

$$H = \left\lceil \frac{2097152 \text{ bits}}{5\ bits} \right\rceil = 419431\ blocks$$

$$\tilde{N} = \left\lceil \frac{419431}{\left\lceil \frac{512\times512}{4} \right\rceil} \right\rceil = 7 \text{ cover images}$$

For *n*=5 pixels, L= 6 bits, then

$$H = \left\lceil \frac{2097152 \text{ bits}}{6\ bits} \right\rceil = 349526\ blocks$$

$$\tilde{N} = \left\lceil \frac{349526}{\left\lceil \frac{512\times512}{5} \right\rceil} \right\rceil = 7 \text{ cover images}$$

For 1024×1024 cover image, and with the same GEMD parameter L in Table 5.6, we got different numbers of cover images required for one secret image using (3.1), (3.2), and (3.3) in each case of n as shown in Table 5.7.

Table 5.7: GEMD parameters of the simulation for 1024×1024 cover images

parameter	Number of pixels n for one block L			
	2	3	4	5
L bits	3	4	5	6
cover images Ñ	2	2	2	2

From Table 5.6 and Table 5.7 we note that as we increase cover image size then we need less number of cover images. Table 5.8 shows the GEMD average results for both cover image sizes that are obtained from Appendix B.

Table 5.8: GEMD average results for 512×512 cover images

Metric	average	Number of pixels n for one block L			
		2	3	4	5
PSNR (dB)	Fully used	50.17	50.79	51.01	51.09
	Fully set	50.96	57.72	51.58	51.33
MSE	Fully used	0.62	0.54	0.51	0.50
	Fully set	0.55	0.46	0.47	0.48
Time (Sec)	Fully used	7.61	5.84	4.40	3.60
	Fully set	6.80	5.03	4.18	3.42
Memory (MB)		480	484	487	491
Capacity (BPP)		1.50	1.33	1.25	1.20

Table 5.9: GEMD average results for 1024×1024 cover images

Metric	average	Number of pixels *n* for one digit			
		2	3	4	5
PSNR (dB)	Fully used	50.16	50.79	51.01	51.09
	Fully set	52.55	52.30	52.11	51.97
MSE	Fully used	0.62	0.54	0.51	0.50
	Fully set	0.42	0.41	0.41	0.42
Time (sec)	Fully used	4.61	3.77	2.92	2.02
	Fully set	4.26	3.31	2.47	1.97
Memory (MB)		491	493	496	498
Capacity (BPP)		1.50	1.33	1.25	1.20

From Table 5.8 and Table 5.9, we have in GEMD results two averages, the first one for the averages without the last image, while the second one for the averages with the last image that not fully embedded as we calculated above for all cases of *n*. For both sizes we got the same results for PSNR and MSE since in GEMD embedding algorithm more than one pixel in a group could be changed by ±1, it means not depends on the size, also for embedding capacity that calculated according to (2.11). Also for both tables, image quality PSNR for GEMD is nearly 51 dB as illustrated in Figure 5.8.

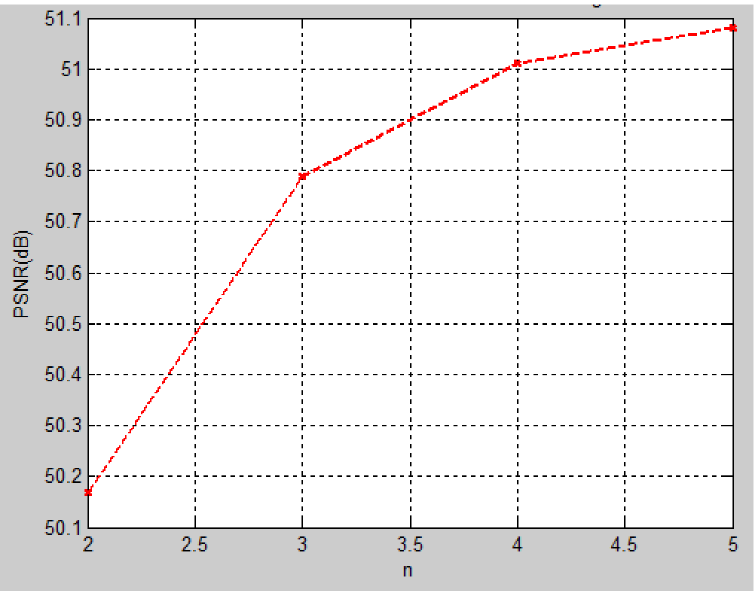


Figure 5.8: PSNR of GEMD using 512×512 and 1024×1024 cover image size

On the other hand MSE decreases with *n*; for both sizes it decreases from 0.62 to 0.50 as shown in Figure 5.9.

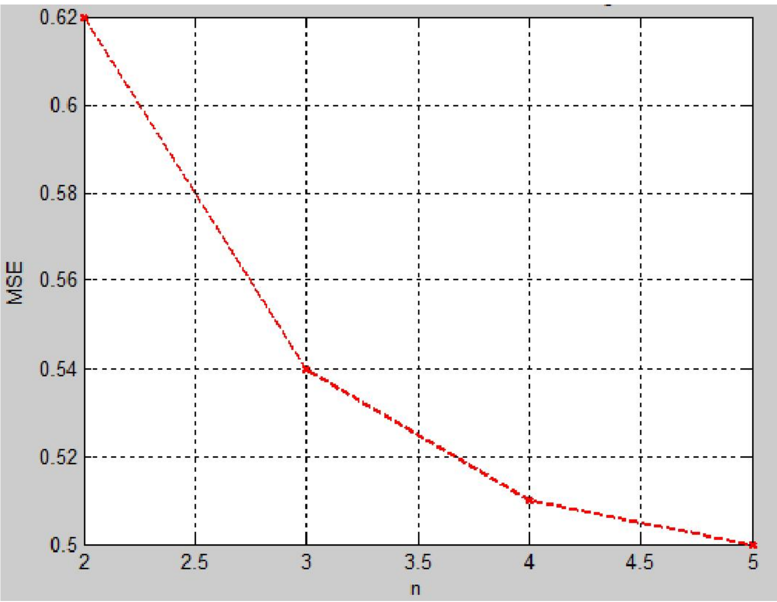


Figure 5.9: MSE of GEMD using 512×512 and 1024×1024 cover image size

For both sizes, embedding capacity BPP, it decreases from 1.50 BPP to 1.20 BPP as *n* increases from 2 to 5 pixels as shown in Figure 5.10.

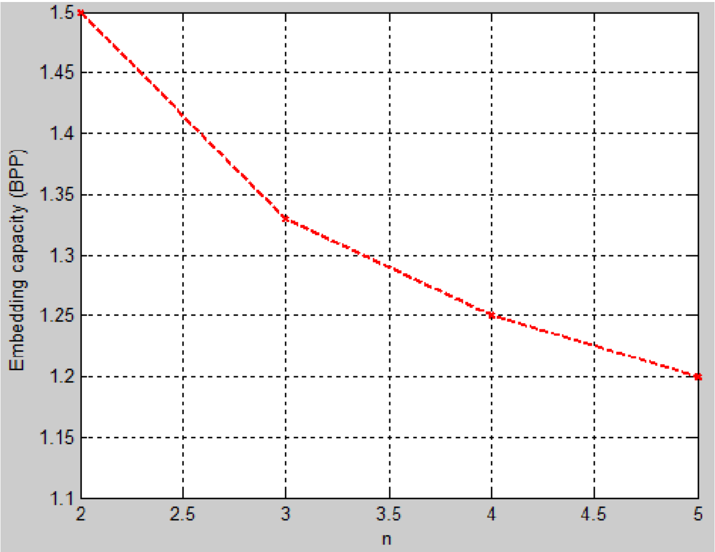


Figure 5.10: BPP of GEMD using 512×512 and 1024×1024 cover image size

From Table 5.8 and Table 5.9 we note that for memory and time consumption we got different values. For time consumption, GEMD using 1024×1024 cover images takes less time than using 512×512 cover images, where time consumption decreases with *n* for fully set average; from Table 5.8 it decreases from 7.61 sec to 3.60 sec, while in Table 5.9 it decreases from 4.61 sec to 2.02 sec when *n* ranges from 2 to 5 pixels as shown in Figure 5.11.

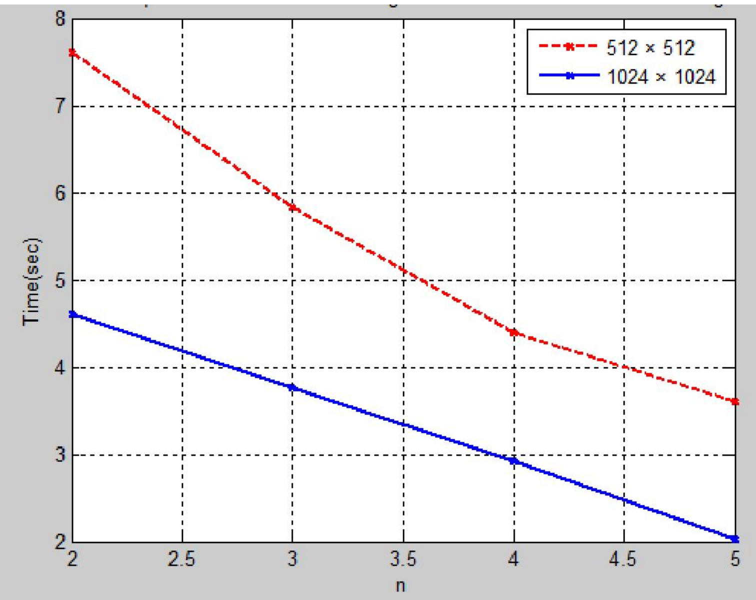


Figure 5.11: Time consumption of GEMD using 512×512 and 1024×1024 cover image size

For memory consumption, GEMD using 1024×1024 cover images in Table 5.9 takes more memory than using 512×512 cover images, since using greater size required to reserve greater locations in memory for array image size. In from Table 5.8 memory consumption increases with *n* for fully set average; it increases from 480MB to 491 MB. On the other hand using 1024×1024 cover images Table 5.9 memory consumption increases from 491 MB to 498 MB when *n* ranges from 2 to 5 pixels as shown in Figure 5.12.

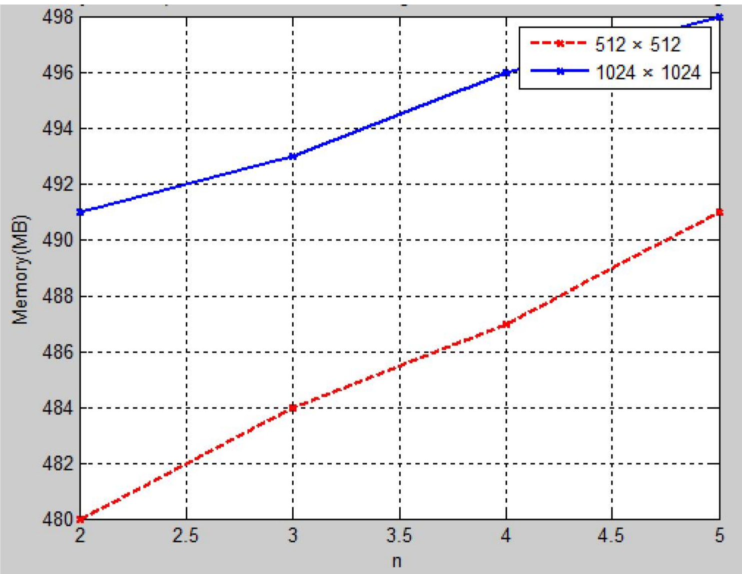


Figure 5.12: Memory consumption of GEMD using 512×512 and 1024×1024 cover image size

From Table 5.8 and Table 5.9 we can summarize the results in Table 5.10 as comparison results using 512×512 and 1024×1024 size of cover image for GEMD method.

Table 5.10: Comparison results for GEMD in two sizes of cover image

metric	GEMD method	
	512×512	1024×1024
PSNR(dB)	50.77	50.77
MSE	0.54	0.54
Time (sec)	5.36	3.33
Memory (MB)	485	494
Capacity (BPP)	1.32	1.32

From Table 5.10 we find that as we use grater size of cover image then we have less time consumption by 0.38% for GEMD. Since in case of using 512 × 512 cover images, we need more cover images and then more time for data processing. For memory consumption, using 1024 × 1024 cover images required more memory by 0.02%. On the other hand, for both sizes we get the same results for metrics PSNR, MSE, and embedding capacity BPP. The GEMD results for both cover image sizes are obtained from appendix B.

5.3 EMD and GEMD Comparison Results

The comparison results for both methods are obtained for MSE (2.8), PSNR (2.9), embedding capacity Bit Per Pixel BPP (2.10), (2.11), memory and time consumption, where the average is taken for each metric over *n* because *n* for each method has different meaning. It means for EMD the number of pixels required to embed one digit from the block, while in GEMD it is the number of pixels required to embed one block. Table 5.11 shows EMD-versus-GEMD comparison results using 512×512 cover images, where the results are obtained from Table 5.3, Table 5.8.while Table 5.12 shows EMD-versus-GEMD comparison results using 1024×1024 cover images, where the results are obtained from Table 5.4, Table 5.9

Table 5.11: The EMD-versus-GEMD results for 512×512 cover images

method	Metric				
	PSNR (dB)	MSE	Time (Sec)	Memory (MB)	Capacity (BPP)
EMD	53.97	0.27	5.71	488	0.89
GEMD	50.77	0.54	5.36	485	1.32

Table 5.12: The EMD-versus-GEMD results for 1024×1024 cover images

method	Metric				
	PSNR (dB)	MSE	Time (Sec)	Memory (MB)	Capacity (BPP)
EMD	53.97	0.27	3.81	496	0.89
GEMD	50.77	0.54	3.33	494	1.32

From Table 5.11 and Table 5.12, we find that EMD stego image quality PSNR is better than 53 dB, since in embedding procedure only one pixel among *n*-pixel group is modified, while in GEMD it is nearly 51 dB, because more than one pixel in each *n*-pixel group could be modified. So in both sizes PSNR in EMD is better than GEMD by 0.06. For MSE comparison result, in both sizes EMD has less error than GEMD by 0.5%, since fewer pixels are changed. On the other hand, GEMD is better in embedding capacity, BPP, by 0.33%. GEMD has less memory and time consumption for both sizes. For 512×512 cover images in Table 5.11, GEMD is better in memory and time consumption by 0.006% and 0.06% respectively. While for using 1024×1024 cover images in Table 5.12, GEMD is better in memory and time consumption by 0.004% and 0.13% respectively.

5.4 Comparison Results to Known Experiments

From the known experiments conducted in Section 2.4, Table 2.1 and our results using 512×512 cover images in Table 5.3 and Table 5.8, we get the same results for the image quality PSNR, MSE and the embedding capacity BPP different cases of *n* as shown in Table 5.13.

Table 5.13: EMD and GEMD comparison results versus known experiments for 512×512 cover images

Result	Metric	*n*=2 pixels		*n*=3 pixels		*n*=4 pixels		*n*=5 pixels	
		EMD	GEMD	EMD	GEMD	EMD	GEMD	EMD	GEMD
[7]	PSNR	52.11	50.17	53.57	50.79	54.66	51.00	55.53	51.09
	MSE	0.40	0.62	0.28	0.54	0.22	0.51	0.18	0.50
	BPP	1.16	1.50	0.93	1.33	0.79	1.25	0.69	1.20
Our	PSNR	52.11	50.17	53.57	50.79	54.66	51.01	55.53	51.09
	MSE	0.40	0.62	0.28	0.54	0.22	0.51	0.18	0.50
	BPP	1.16	1.50	0.93	1.33	0.79	1.25	0.69	1.20

5.5 Summary of Chapter 5

Thus, in this chapter we have discussed and compared EMD and GEMD results with same size and also with different size of cover images which are obtained from Appendixes, as we also compare these results with known experiments [7] presented in Section 2.4.

Chapter 6

CONCLUSION AND THE FUTURE WORK

This thesis analyzes two steganographic methods; EMD and GEMD. The algorithms are explained in details such as the input, output, data structure, the justification of their correctness and the best values for their parameters which required a minimum number of cover images to maintain good image quality PSNR and minimum MSE, time and memory consumption, then the experiments results compared between both methods and then with known experiments conducted on EMD and GEMD.

The results were obtained for four gray scale secret images, and for two different sizes of cover images, where the number of cover image required for a secret image is defined according to some parameters such as the number of bits in each block of secret image, and the number of pixels n in each group of the cover image. The experiments were conducted with four different values of n, as we tried to find the best value for the number of bits in each block L of the secret image and the maximum digit k in $(2n+1)$-ary in each case of n to achieve the best case of EMD and GEMD which taking less number of cover image. According to our analysis, and for both sizes cover image, EMD stego image quality PSNR is better than GEMD, since fewer pixels values are modified. On the other hand, GEMD has less memory and time consummation, since increasing of cover size takes more memory and less time for both methods. For MSE comparison result EMD has less error than GEMD, because in EMD at most only one pixel is changed by ±1 in a group, while in GEMD more than one pixel in a group could be modified. But also GEMD has greater embedding capacity for both sizes. In addition to greater cover size required less number of cover images and for both sizes GEMD required less number of cover images

As a comparison between the results using different size of cover image, we find that as we use greater size then we need less number of cover images, and less time consumption. On the other hand we need more memory consumption. For other metrics, PSNR, MSE, and embedding capacity BPP, we get the same results.

However, the both methods have the same aim for hiding data, but one of them, EMD, is better in image quality, PSNR and MSE, while GEMD is better in memory and time consumption and also better embedding capacity.

As a future work, I propose to study and implement these methods by using color images in order to improve the performance of them.

REFERENCES

[1] Cheddad, A., Condell, J., Curran, K., & Kevitt, P. M. (2010, October). Digital Image Steganography: Survey and Analysis of Current Methods. *Signal Processing*, pp. 727-752. Vol 90. No.3.

[2] Devi, M., & Sharma, N. (2014, March). Improved Detection of Least Significant Bit SteganographyAlgorithms in Color and Gray Scale Images. *IEEE*, *Recent Advances in Engineering and Computational Sciences (RAECS)* , pp. 1-5. Vol 34. No.7.

[3] Hegde, R., & S, J. (2015, July). Design and Implementation of Image Steganography by Using LSB Replacement Algorithm and Pseudo Random Encoding Technique. *International Journal on Recent and Innovation Trends in Computing and Communication*, pp.4415 - 4420. Vol 3. No.7.

[4] Jarno, M. (2006, May). LSB Matching Revisited. *IEEE*, *Signal Processing Letters*, pp. 285- 287. Vol 13 No.5.

[5] Kuo, W.-C., & Wang, C.-C. (2013, October). Data Hiding Based on Generalised Exploiting Modification Direction Method. *The Imaging Science Journal*, pp.484-490. Vol 61. No.10.

[6] Kuo, W. C., Chen, Y. H., & Chuang, C.-T. (2014, April). High-Capacity Steganographic Method Based on Division Arithmetic and Generalized

Exploiting Modification Direction. *Journal of Information Hiding and Multimedia Signal Processing*, pp. 213-222. Vol 5. No.2.

[7] Kuo, W. C., Wang, C. C., & Hou, H. C. (2015, August). Signed Digit Data Hiding Scheme. *Information Processing Letters*, pp. 15-26. Vol 5. No.2.

[8] Kieu, T. D. & Chang, C. C. (2011, April) A Steganographic Scheme by Fully Exploiting Modification Directions, *Expert Systems with Applications*, pp.10648-10657. Vol 38. No.8.

[9] Lee .C.F; Wang. Y & Chang. C (2007, August). A Steganographic Method with High Embedding Capacity by Improving Exploiting Modification Direction. *Proceedings of the Third International Conference on Intelligent Information Hiding and Multimedia Signal Processing(IIHMSP07)*, pp.497-500. Vol 5. No.2.

[10] Pan, H. K., Tseng, Y. C & Chen Y. Y. (2002, August). A Secure Data Hiding Scheme for Binary Images. *IEEE Trans. Commun.*, pp. 1227-1231.Vol. 50.No.8.

[11] Rita, C. & Deepika, B. (2014, September), An Improved DCT based Steganography Technique, International Journal of Computer Applications, pp. 46-49. Vol 102. No.14.

[12] Wang, R. Z; Lin, C. F.; & Lin, J. C (2001, May). Image Hiding by Optimal LSB Substitution and Genetic Algorithm. *Pattern Recognition*, pp.671-683 .Vol 34.No 3.

[13] Wu, D. C., & Tsai, W. H. (2003, April). A Steganographic Method for Images by Pixel-Value Differencing. *Pattern Recognition Letters*, pp.1613-1626. Vol 24. No.9.

[14] Wu, H. C., Wu, N. I., Tsai, C. S., & Hwang, M. S. (2005, March). Image Steganographic Scheme Based on Pixel-Value Differencing and LSB Replacement Methods. *IEEE, image signal process*, pp. 611–615.Vol 152. No. 12.

[15] Zhang, X., & Wang, S. (2006, November). Efficient Steganographic Embedding by Exploiting Modification Direction. *IEEE Communication Letters,* pp. 781-783. Vol 12. No.7.

[16] Zhi, H. W., Kieu,T.D.,& Chin, C.C. (2010, January), A Novel Information Concealing Method Based on Exploiting Modification Direction, *Journal of Information Hiding and Multimedia Signal Processing*, pp.130-138, Vol 1. No.1.

[17] Vijay, K. & Dinesh, K. (2010, June), Performance Evaluation of DWT Based Steganography, *IEEE 2nd International Advance Computing Conference,* pp. 223-228. Vol 6. No.10.

APPENDICES

Appendix A: EMD Algorithm

A.1 The main program

```
% this program was written by Om Essad M.Lamiles in 2015-2016
for EMD algorithm [8] and its functions
clc;
clear all ;
images={'Lena','Baboon','F16','Barbara','Monaliza',
'Tiffany','Girl','Cameraman','Liza','Jug','House','Roza'}%cove
r images used in the program
sum_time=0;sum_psnr=0;sum_mse=0;
BDig=[];d_msg=[];
addpath('cover_set/');addpath('secret_set/');
img_name = 'P';sec_name='S';
M=512; N=M;
E_dig=[];
L=input('Input L: the number of bits in a block ');
n=input(' Input n: the number of pixels in a group ');
k=ceil(L/(log2(2*n+1)))%calculate k as in (2.3)
C=floor((M*N)/n); %calculate C as in (3.3)
Bpp=(log2(2*n+1))/n%calculate bppEMD as in (2.12)
sec = imread([sec_name, '',num2str(1) '.jpg']);%read the first
secret image , '' used to read image sec_name =S1
corresponding to the number in (),as we have 4 secret images
S =reshap_im(sec,M,N); %reshape secret image as one
dimensional array
[Bin]= conv2binary(S); %convert each pixel of secret image to
binary
s_size=numel(Bin)
H=ceil(s_size /L) %calculate H as in (3.1)
SS=[Bin zeros(1,(L-(mod(s_size,L))))]; %last block padded by
zeros
Cover_im = ceil((H*k)/C)%calculate Ñ as in (3.2)
Covers = uint8( zeros(M,N,Cover_im) );
Stegos = uint8( zeros(M,N,Cover_im) );
[Dig] = BTO2NP1(SS,L,k,n,H ); %get stream of k digits in
(2n+1)-ary numbers
h=1;
  for i=1:Cover_im
tic% starting of timer to calculate embedding time
CI = imread([img_name, '',num2str(i) '.tif']);
Covers(:,:,i)=CI;
ci1 =reshap_im(CI,M,N);
x=1;
if (i==Cover_im)
C=mod(numel(Dig),h)%if the last image will be not fully
embedded
end
```

```
   for r=1:C
group= ci1((x-1)*n+1:x*n); %get n pixel group from cover image
[em_group]=embed(group,Dig(h),n); %send to embedding function
ci1((x-1)*n+1:x*n)=[em_group]; %resave embedded group to cover
image to get stego image
    h=h+1;
    x=x+1;
   end
ci2 =reshap_im2(ci1,M,N);
Stegos(:,:,i)=ci2;
tim(i)=toc%get the embedding time for each image
mem=memory %calculate memory for each image
end
```

A.2 Reshaping image as one dimensional array

```
function [S] = reshap_im(Sec,m,n)
for i=1:m
    for j=1:n
      im((i-1)*n+j)= Sec(i,j);
    end
end
end
```

A.3 Converting secret image into binary stream

```
function [Bin] = conv2binary(S)
Bin=[];
for j=1:numel(S)
    b = bitget(uint8( S(j)),8:-1:1); %get pixel as binary
   [Bin]=[Bin b];
end
end
```

A.4 Converting binary message to (2n+1)_ary

```
function [Dig ] = BTO2NP1(SS,L,k,n,H )
Dig=[];
  for i=1:H
  B= SS((i-1)*L+1:i*L)%get B block from binary message SS
    sum=0;
    sum1=bin2dec(num2str(B))% convert to decimal
D=dec2base(sum1,(2*n+1),k) % convert to k digits (2n+1)-ary
number
    Dig=[Dig D];
    end
end
```

A.5 Embedding function

```
function [em_group] = embed(group, Dig ,n)
```

```
sum=0;
  for i=1:n
  sum =sum +double(group(i))*i;%calculate extraction
function(2.4)
  end
t=mod(sum,(2*n+1));
d=mod((Dig)-t),(2*n+1)); %calculate d as in(2.5)
if (d<=n && d>0)
   group(d)=(group(d))+1; %from(2.5.2)
elseif (d>n)
 group(((2*n+1))-d)=(group(((2*n+1))-d))-1; %from(2.5.3)
end
em_group= group;
end
```

A.6 Code showing results and stego images

```
disp('=================================================')
disp('stego image  PSNR  MSE   Time  Memory  Capacity')
disp('              dB          sec    MB        bpp   ')
disp('================================================')
set(gcf, 'name', ' Secret Image in case n=2');
 for i=1:Cover_im
subplot(2,3,i) ;  imshow((Stegos(:,:,i)));
[PSNR(i), MSE(i)]=My_PSNR(Covers(:,:,i),Stegos(:,:,i));
title(['PSNR = ',num2strPSNR(i) ]);
sprintf('%s%f%f%f%f%f',images{i},PSNR(i),MSE(i),tim(i),mem,Bpp
)
disp('=================================================')
sum_time=sum_time+tim(i);
sum_psnr=sum_psnr+PSNR(i);
sum_mse=sum_mse+MSE(i);
 end%take the average for PSNR,MSE and Time
psnr_used=(sum_psnr- PSNR(Cover_im))/(Cover_im-1);
mse_ used =(sum_mse - MSE(Cover_im))/(Cover_im-1);
tim_ used =(sum_time- tim(Cover_im))/(Cover_im-1);
psnr_set=sum_psnr/Cover_im;
mse_set=sum_mse/Cover_im;
tim_set=sum_time/Cover_im;
sprintf(' Average on fully used  ')
disp('=================================================')
sprintf('%.2f%.2f%.2f%.2f%.2f',psnr_used,mse_used,tim_used,
mem,Bpp)
sprintf(' Average on fully set  ')
disp('=================================================')
sprintf('%.2f%.2f%.2f%.2f%.2f',psnr_set,mse_set,tim_set,mem,Bp
p)
```

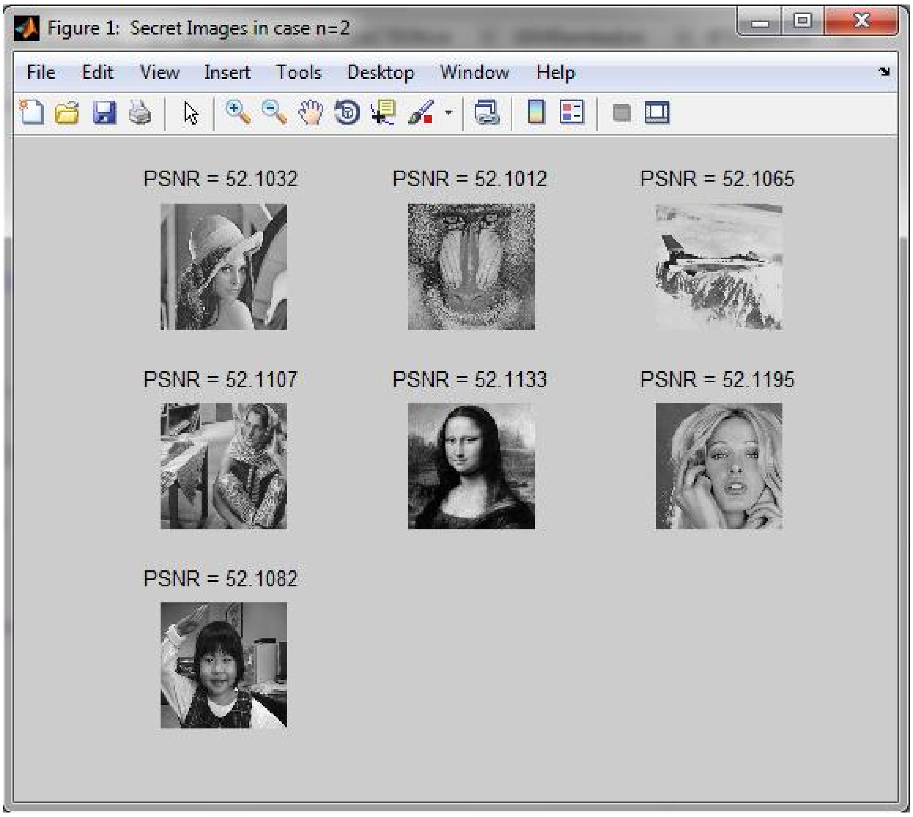
A.7 Calculation of PSNR and MSE

```
function [ My_psnr MSE ] = My_PSNR(I,J)
    X = double(I);
    Y = double(J);
    MSE = sum((X(:)-Y(:)).^2) / prod(size(X)) ;
    My_psnr = 10*log10(255 * 255/MSE);

End
```

A.8 Screenshots of EMD Result in different values of n, L, k and cover image of size 512×512 and 1024×1024.

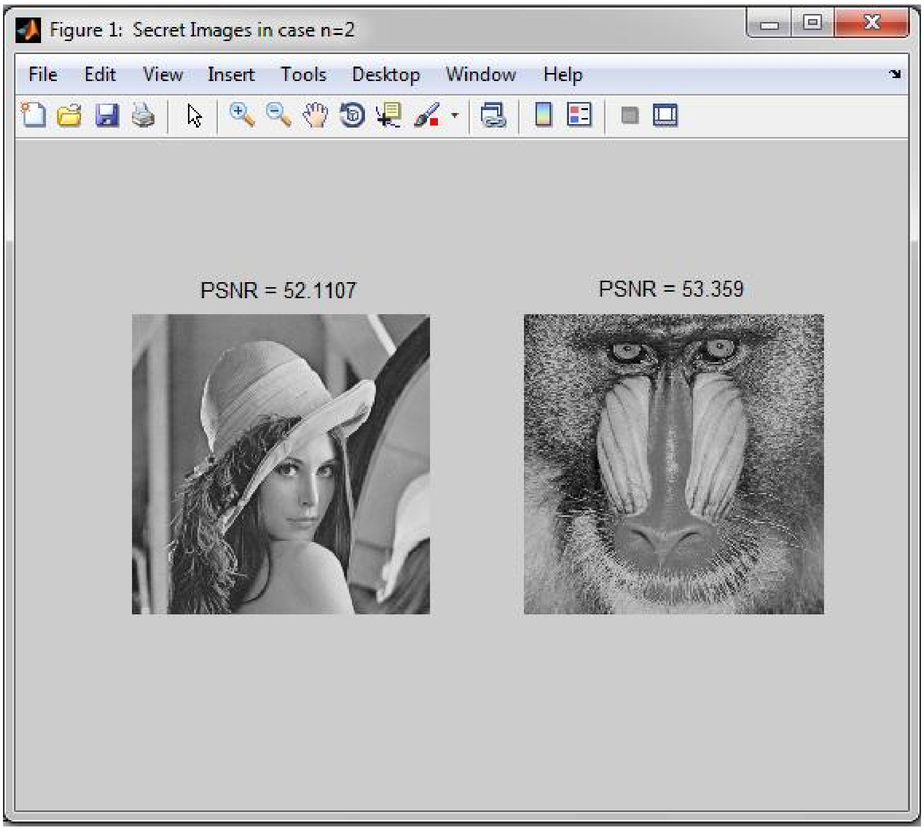
A.8.1.a. Results in *n*=2, k=7, L=16, cover images =7,512×512 cover image of size



PSNR = 52.1032
PSNR = 52.1012
PSNR = 52.1065
PSNR = 52.1107
PSNR = 52.1133
PSNR = 52.1195
PSNR = 52.1082

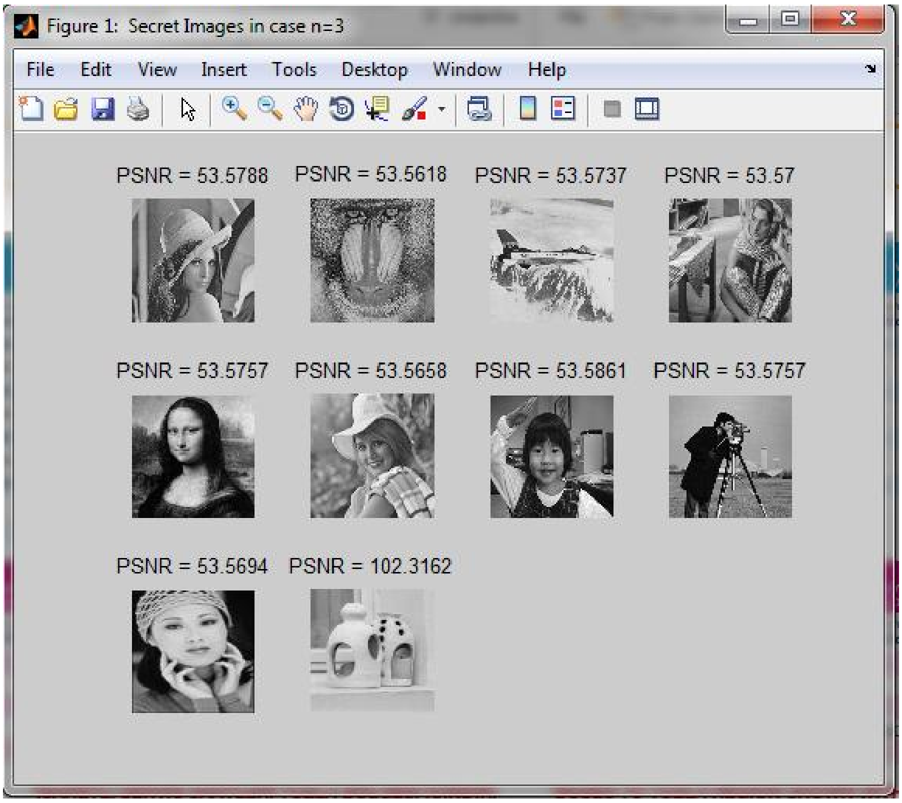
```
===================================================== ======
 Stego image    PSNR       MSE    Time      Memory      Capacity
                 dB               sec         MB          bpp
===========================================================
Lena           52.10      0.40    7.81       481         1.16
===========================================================
Baboon         52.10      0.40    7.81       481         1.16
===========================================================
F16            52.11      0.40    7.81       481         1.16
===========================================================
Barbara        52.11      0.40    7.81       481         1.16
 ==========================================================
Monaliza       52.11      0.40    7.81       481         1.16
===========================================================
Tiffany        52.12      0.40    7.81       481         1.16
===========================================================
Girl           52.11     0.40     7.81       481         1.16
===========================================================
Average on fully used
===========================================================
             52.11      0.40       7.81    481           1.16
```

A.8.1.b. Results in *n*=2, k=7, L=16, cover images =2, 1024×1024 cover image size



```
=================================================== ======
 Stego image     PSNR      MSE    Time     Memory      Capacity
                  dB               sec       MB          BPP
==========================================================
Lena               52.11    0.40   5.72       493          1.16
==========================================================
Baboon             53.36    0.35   5.01       493          1.16
==========================================================
Average on fully used
==========================================================
                  52.11     0.40    5.72      493          1.16
Average on fully set
==========================================================
                   52.73    0.37    5.36      493         1.16
==========================================================
```

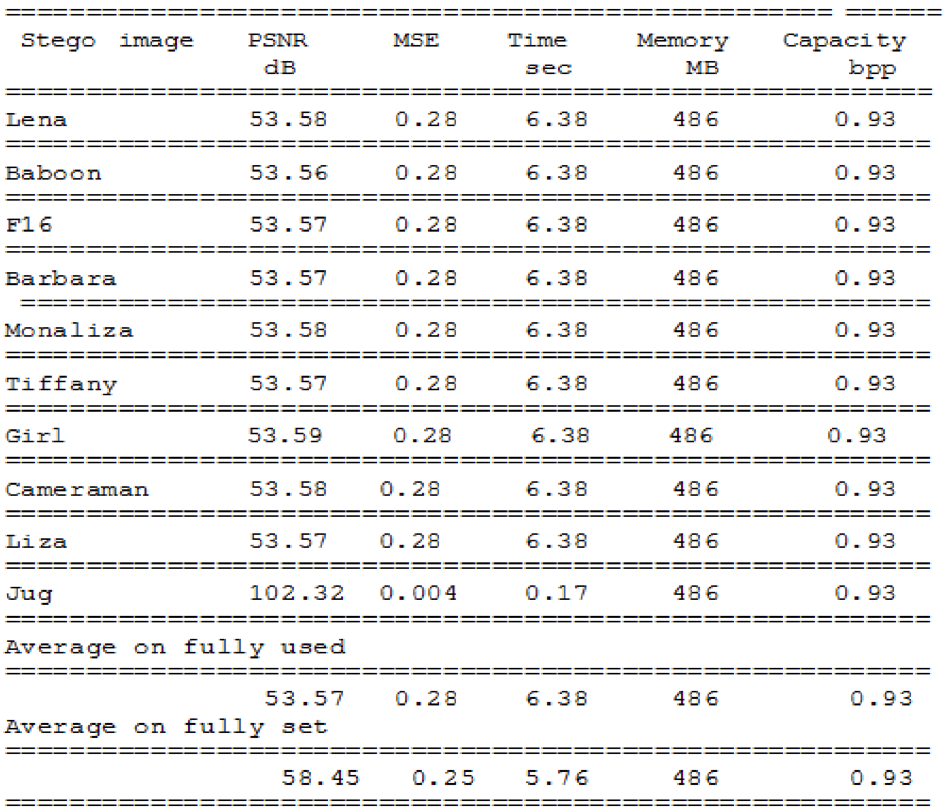
A.8.2.a. Results in n=3, k=6, L=16, cover images =9, 512×512 cover image size



PSNR = 53.5788 PSNR = 53.5618 PSNR = 53.5737 PSNR = 53.57

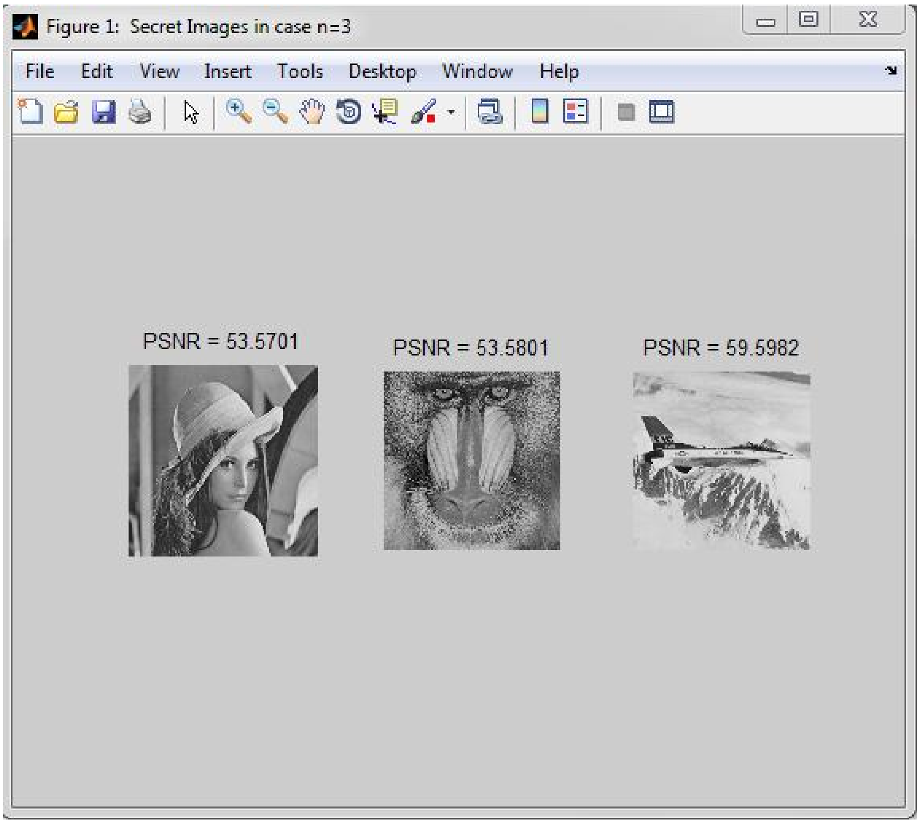
PSNR = 53.5757 PSNR = 53.5658 PSNR = 53.5861 PSNR = 53.5757

PSNR = 53.5694 PSNR = 102.3162



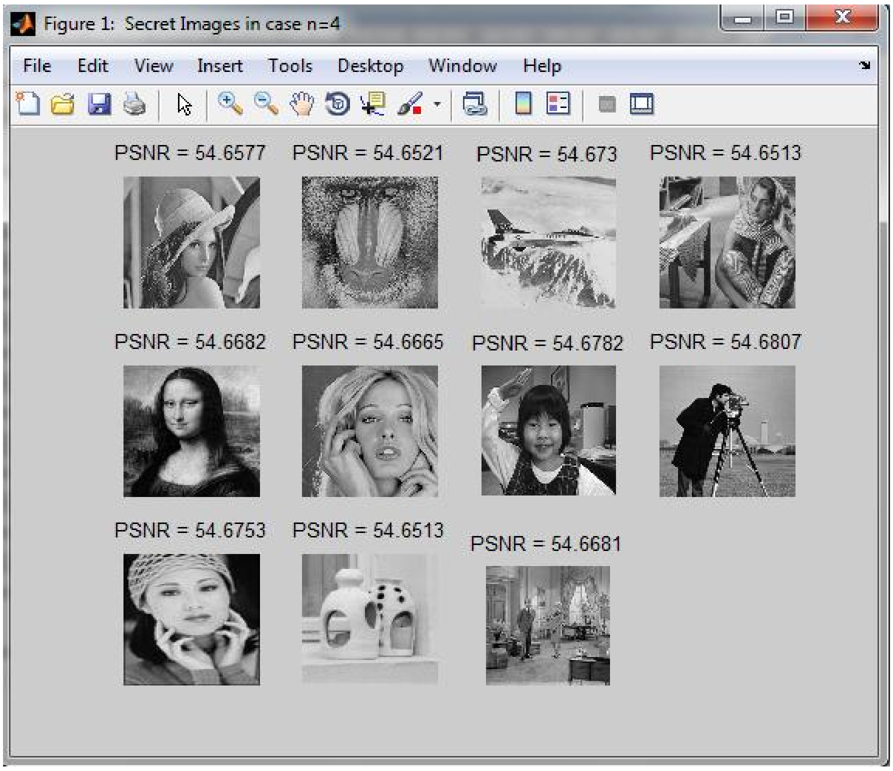
```
================================================== ======
 Stego  image     PSNR      MSE      Time     Memory    Capacity
                   dB                  sec       MB          bpp
=========================================================
Lena              53.58     0.28     6.38      486         0.93
=========================================================
Baboon            53.56     0.28     6.38      486         0.93
=========================================================
F16               53.57     0.28     6.38      486         0.93
=========================================================
Barbara           53.57     0.28     6.38      486         0.93
 ========================================================
Monaliza          53.58     0.28     6.38      486         0.93
=========================================================
Tiffany           53.57     0.28     6.38      486         0.93
=========================================================
Girl              53.59     0.28      6.38     486        0.93
=========================================================
Cameraman         53.58   0.28       6.38      486         0.93
=========================================================
Liza              53.57   0.28       6.38      486         0.93
=========================================================
Jug               102.32  0.004      0.17      486         0.93
=========================================================
Average on fully used
=========================================================
                   53.57    0.28     6.38      486          0.93
Average on fully set
=========================================================
                    58.45    0.25    5.76      486          0.93
=========================================================
```

A.8.2.b. Results in *n*=3, k=6, L=16, cover images =3, 1024×1024 cover image size



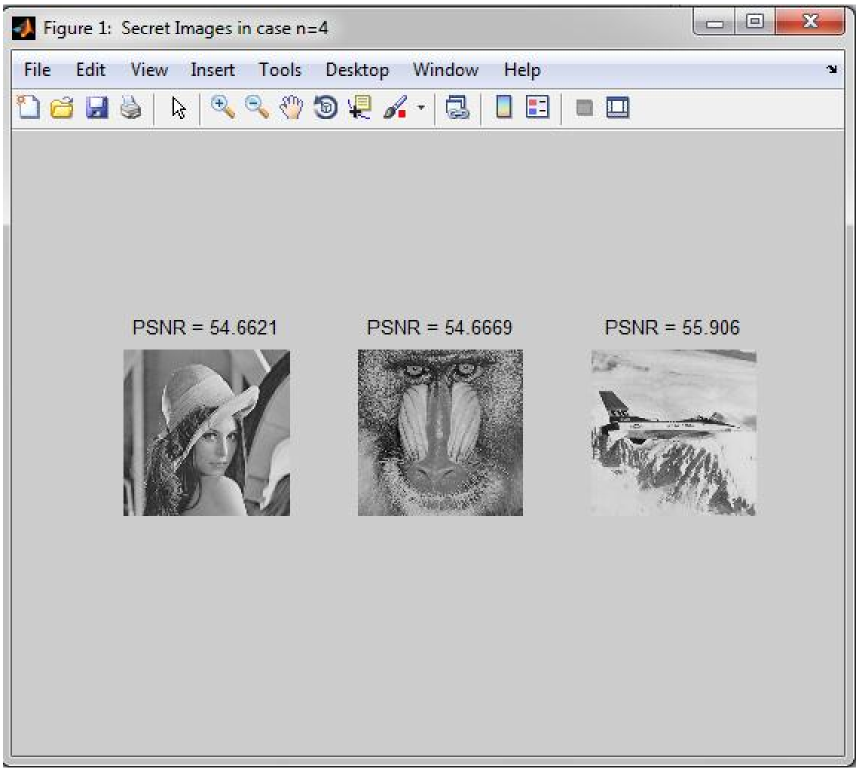
```
================================================== ======
 Stego image    PSNR     MSE     Time     Memory   Capacity
                 dB               sec       MB        BPP
==========================================================
Lena            53.57    0.28    4.31     495       0.93
==========================================================
Baboon          53.58    0.28    4.31     495        0.93
==========================================================
F16             59.60    0.07    3.80     495       0.93
==========================================================
Average on fully used
==========================================================
                 53.57   0.28    4.31     495        0.93
Average on fully set
==========================================================
                 55.58  0.21     4.14      495        0.93
==========================================================
```

A.8.3.a. Results in n=4, k=11,L=32, cover images =11, 512×512 cover image size



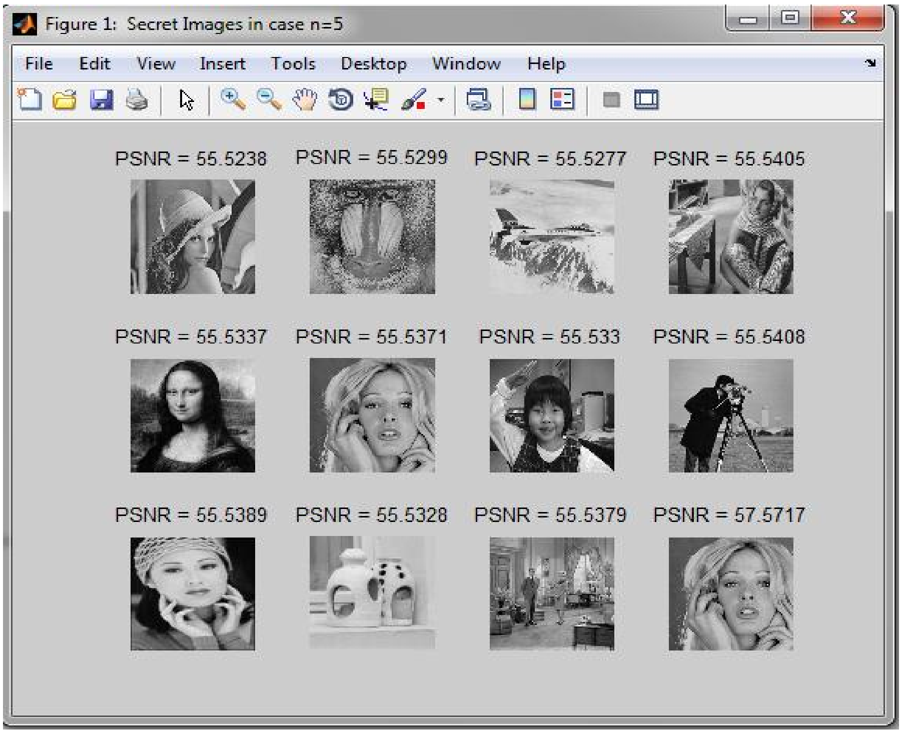
```
=================================================== ======
stego image   PSNR     MSE     Time     Memory       Capacity
               dB               sec       MB            bpp
==========================================================
Lena        54.67      0.22    4.66      490           0.79
==========================================================
Baboon      54.65      0.22    4.66      490           0.79
==========================================================
F16         54.67      0.22    4.66      490           0.79
==========================================================
Barbara     54.65      0.22    4.66      490           0.79
 =========================================================
Monaliza    54.67      0.22    4.66      490           0.79
==========================================================
Tiffany     54.67      0.22    4.66      490           0.79
==========================================================
Girl        54.68      0.22    4.66      490           0.79
==========================================================
Cameraman   54.68      0.22    4.66      490           0.79
==========================================================
Liza        54.68      0.22    4.66      490           0.79
==========================================================
Jug         54.65      0.22    4.66      490           0.79
==========================================================
House       54.67      0.22    4.66      490           0.79
==========================================================
Average on fully used
==========================================================
            54.66      0.22    4.66      490           0.79
```

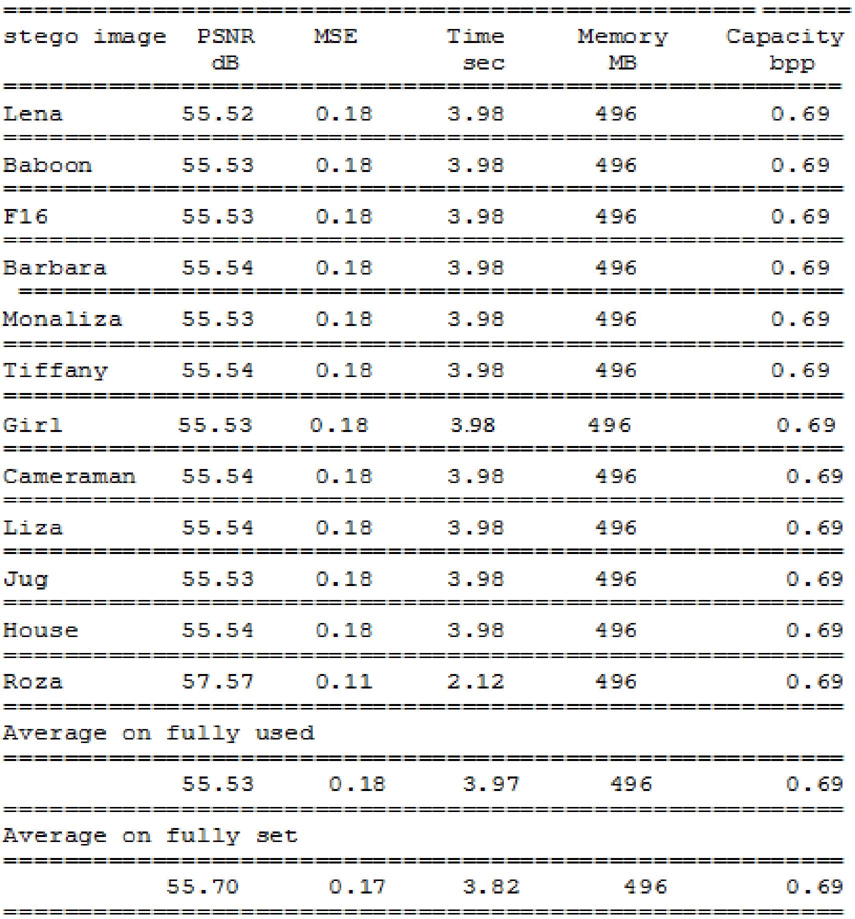
A.8.3.b. Results in *n*=4, k=11, L=32, cover images =3, 1024×1024 cover image size



```
================================================= ======
stego image    PSNR    MSE    Time    Memory       Capacity
                dB             sec      MB            BPP
========================================================
Lena          54.66    0.22    3.02     497           0.79
========================================================
Baboon        54.67    0.22    3.02     497           0.79
========================================================
F16           55.91    0.16    2.69     497           0.79
========================================================
Average on fully used
========================================================
              54.66    0.22    3.02     497            0.79
========================================================
Average on fully set
========================================================
              55.08    0.20    2.91      497           0.79
========================================================
```

A.8.4.a. Results in n=5, k=19,L=64, cover images =12, 512×512 cover image size





```
============================================================
stego image   PSNR     MSE       Time      Memory    Capacity
               dB                 sec        MB         bpp
============================================================
Lena          55.52    0.18      3.98       496        0.69
============================================================
Baboon        55.53    0.18      3.98       496        0.69
============================================================
F16           55.53    0.18      3.98       496        0.69
============================================================
Barbara       55.54    0.18      3.98       496        0.69
 ===========================================================
Monaliza      55.53    0.18      3.98       496        0.69
============================================================
Tiffany       55.54    0.18      3.98       496        0.69
============================================================
Girl          55.53    0.18      3.98       496        0.69
============================================================
Cameraman     55.54    0.18      3.98       496         0.69
============================================================
Liza          55.54    0.18      3.98       496         0.69
============================================================
Jug           55.53    0.18      3.98       496         0.69
============================================================
House         55.54    0.18      3.98       496         0.69
============================================================
Roza          57.57    0.11      2.12       496         0.69
============================================================
Average on fully used
============================================================
              55.53     0.18      3.97       496        0.69
============================================================
Average on fully set
============================================================
             55.70      0.17      3.82        496       0.69
============================================================
```

A.8.4.b. Results in *n*=5, k=19, L=64, cover images =3, 1024×1024 cover image size

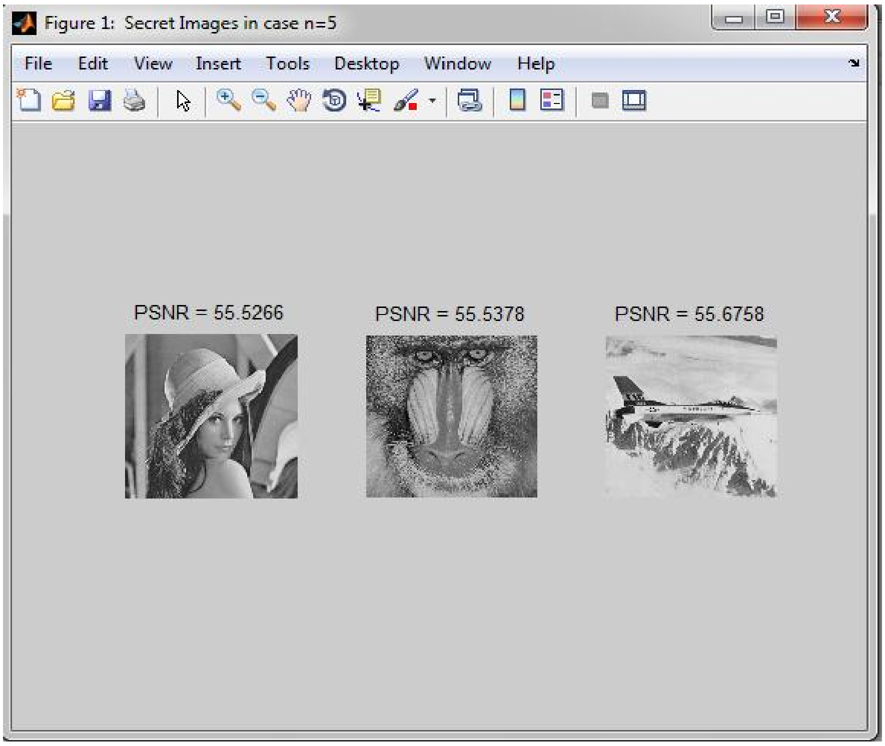
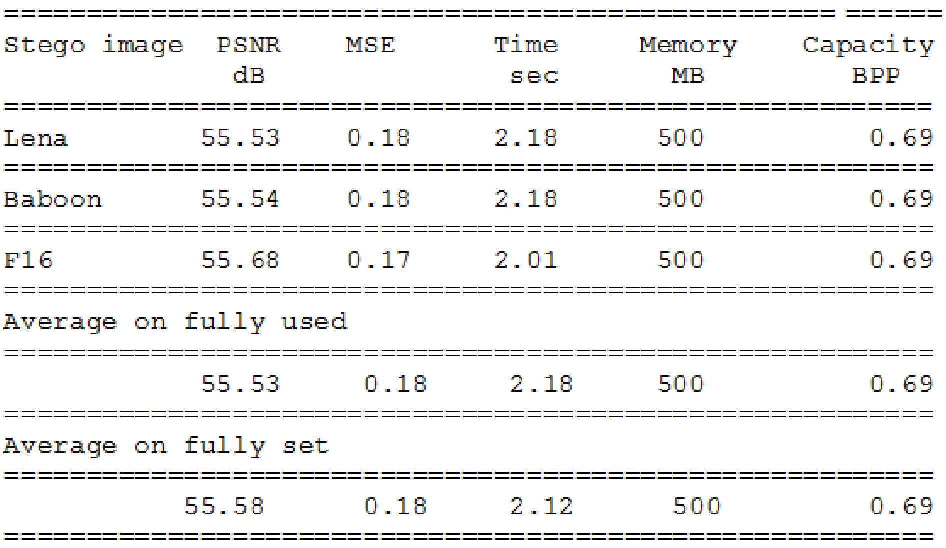


Figure 1: Secret Images in case n=5

PSNR = 55.5266

PSNR = 55.5378

PSNR = 55.6758



```
=========================================================== ======
Stego image   PSNR     MSE        Time       Memory     Capacity
               dB                  sec         MB          BPP
============================================================
Lena          55.53     0.18      2.18       500          0.69
============================================================
Baboon        55.54     0.18      2.18       500          0.69
============================================================
F16           55.68     0.17      2.01       500          0.69
============================================================
Average on fully used
============================================================
              55.53      0.18      2.18      500          0.69
============================================================
Average on fully set
============================================================
             55.58       0.18      2.12       500         0.69
============================================================
```

A.9 Extraction phase

```
options.Interpreter = 'tex';
options.Default = 'Yes';
qstring = 'Do you want to extract data?';
choice=questdlg(qstring,'EXTRACTION','Yes','No',options);
 switch choice
    case 'Yes'
for j=1:Cover_im
  Stegos1(1,:,j)=reshap_im(Stegos(:,:,j),M,N);
end
[secret_message]= EXTRACTION( Stegos1,k,n,H,L ); %get binary
stream from the EXTRACTION function
  v=1;
for i=1:M*N
  bmess=secret_message((v-1)*8+1:v*8); %get 8 bits block from
extracted binary stream
  a=bin2dec(num2str(bmess)); %convert to decimal
  d_msg=[d_msg a];
  v=v+1;
end
  secret_im=reshap_im2(d_msg,M,N); %reshape d_msg as 2
dimensional array
 set(gcf, 'name', ' Extracted Secret image');
   imshow(uint8(secret_im)); %show the Extracted Secret image
case 'No'
 break;
 end
```

A.10 Extraction function

```
function [B_msg]=  EXTRACTION( Stegos,k,n,H,L )
secret_msg=[];
B_msg=[];
```

```
R=(2*n+1);
for i=1:H
    for j=1:k
        x=(i-1)*k+j;
    group=Stegos((x-1)*n+1:x*n); %get n pixel group from stego
image
   [secret]=Extract(group,n); %get k digits
secret_msg=[secret_msg secret];
    end
    secret_msg;
    sum=0;
     E=numel(secret_msg);
for t=0:E-1
    secret_msg(E-t);
  sum= sum+(secret_msg(E-t))*R^t;%convert to decimal
end

A=dec2bin(sum,L); %convert to binary
  B_msg=[B_msg A];
secret_msg=[];
end
end
```

A.10.1 Extract function

```
function [secret_msg1]=Extract(group,n)
sum=0;
   for i=1:n
      sum = sum + double( group(i))*i; %calculate extaction
function as in (2.6)
   end
secret_msg1=mod(sum,(2*n+1));
end
```

A.11 Reshaping secret image as two dimensions

```
function [ stego ] = reshap_im2(Covers,M,N)
for i=1:M
    for j=1:N
      stego(i,j)=Covers((i-1)*N+j);
    end
end
end
```

Appendix B: GEMD Algorithm

B.1 The Main program

```
% This program was written by Om Essad M.Lamiles in 2015-2016
for GEMD algorithm [5] and its functions
clc;
clear all ;
sum_psnr=0;sum_mse=0; sum_time=0;
images={'Lena','Baboon','F16','Barbara','Monaliza'
,'Tiffany','Girl'};%cover images used in the program
addpath('cover_set/');addpath('secret_set/');
img_name = 'P';sec_name='S';
M=512; N=M;
d_msg=[];
n=input(' Input n: the number of pixels in a group ');
L=n+1;
C=floor((M*N)/n); %calculate C as in (3.3)
Bpp=(n+1)/n%calculate bppGEMD as in (2.13)
sec = imread([sec_name, '',num2str(1) '.jpg']);
S =reshap_im(sec,M,N); %reshape secret image as one
dimensional array
[Bin]= conv2binary(S); %convert each pixel of secret image to
binary
s_size=numel(Bin)
H=ceil(s_size/L) %calculate H as in (3.1)
Cover_im =ceil(H/C) %calculate Ñ as in (3.2)
s_size=numel(Bin)
SS=[Bin zeros(1,(L-(mod(s_size,L))))]%last block padded by r
zeros
Covers = uint8( zeros(M,N,Cover_im) );
Stegos = uint8( zeros(M,N,Cover_im) );
[Num]=GET_B(SS,L,H);
h=1;
for i=1:Cover_im
CI = imread([img_name, '',num2str(i) '.tif']);
tic% starting of timer to calculate embedding time
  Covers(:,:,i)=CI;
 ci1 =reshap_im(CI,M,N);
  x=1;
if (i==Cover_im) %if the last image will be not fully embedded
 C=mod(numel(Num),h)%if the last image will be not fully
embedded
  End
for r=1:C
group= ci1((x-1)*n+1:x*n); %get n pixel group from cover image
```

```
[em_group]=GEMDembed(group,Num(h),n); %send to embedding
function
ci1((x-1)*n+1:x*n)=[em_group]; %resave embedded group to cover
image to get stego image
h=h+1; x=x+1;
end
ci2 =reshap_im2(ci1,M,N);
Stegos(:,:,i)=ci2;
tim(i)=toc%get the embedding time for each image
mem=memory%calculate memory for each image
end
```

B.2 Reshaping image as one dimensional array

```
function [S] = reshap_im(Sec,m,n)
for i=1:m
    for j=1:n
      im((i-1)*n+j)= Sec(i,j);
    end
end
end
```

B.3 Converting secret image into binary stream

```
function [Bin] = conv2binary(S)
Bin=[];
for j=1:numel(S)
    b = bitget(uint8( S(j)),8:-1:1); %get pixel as binary
   [Bin]=[Bin b];
end
end
```

B.4 Dividing binary message to (n+1) bit blocks

```
function [Num] = GET_B(S,L,H)
Num=[];
 for i=1:H
    B= S((i-1)*L+1:i*L);
    d=bin2dec(num2str(B));
    Num=[Num d];
 end
end
```

B.5 Embedding function

```
function [em_group ] = GEMDembed( group,Num,n )


sum=0;
    for i=1:n
    sum = sum + double( group(i)) *((2^i)-1); %calculate
extraction function(2.7)
    end
t=mod(sum,(2^(n+1)));
d=mod(Num-t,(2^(n+1))); %calculate d as in(2.8)

if (d==2^n) R=1; %step5 in GEMD embedding algorithm
     elseif(d<(2^n)) R=2;  else R=3;
  end
    switch R%step6 in GEMD embedding algorithm
case 1
       group(n)= group(n)+1;group(1)= group(1)+1;
case 2 d=dec2bin(d,(n+1)); %convert d to binary(d_n d_n-1 ….d_0)_2
sized n+1 bits
for i=0:n-1
    if ((d(i+1)=='1')&&(d(i+2)=='0'))
        group(n-i)=group(n-i)-1;
elseif ((d(i+1)=='0')&&(d(i+2)=='1'))
        group(n-i)=group(n-i)+1;
        end
end
case 3 d=(2^(n+1))-d; b=dec2bin(d,(n+1)); %convert d to
binary(d_n d_n-1 ….d_0)_2 sized n+1 bits
for j=0:n-1
     if((b(j+1)=='1')&&(b(j+2)=='0'))
           group(n-j)= group(n-j)+1;
     elseif((b(j+1)=='0')&&(b(j+2)=='1'))
           group(n-j)= group(n-j)-1;
     end
end
    end
    em_group= group;
end
```

B.6 Code showing results and stego images

```
disp('=================================================')
disp('stego image  PSNR  MSE   Time  Memory  Capacity')
disp('              dB          sec     MB        bpp   ')
disp('================================================')
set(gcf, 'name', ' Secret Image in case n=2');
 for i=1:Cover_im
subplot(2,3,i) ;  imshow((Stegos(:,:,i)));
[PSNR(i), MSE(i)]=My_PSNR(Covers(:,:,i),Stegos(:,:,i));
```

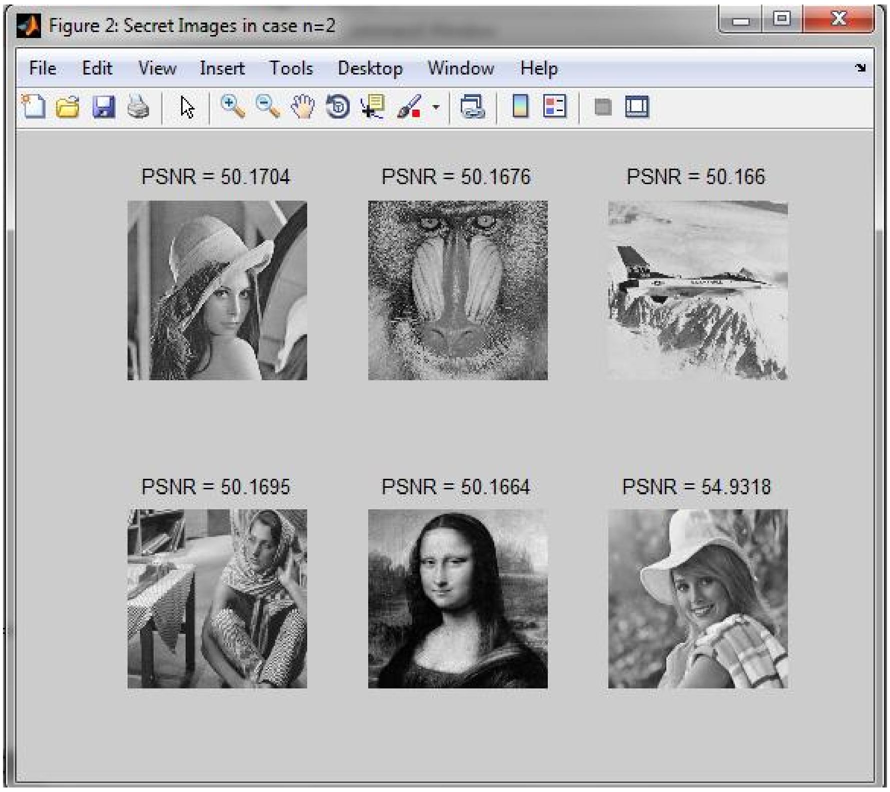
```
title(['PSNR = ',num2str(PSNR(i))]);
sprintf('%s%f%f%f%f',images{i},PSNR(i),MSE(i),tim(i),mem,Bpp
)
disp('=================================================')
sum_time=sum_time+tim(i); %take the average for PSNR,MSE
sum_psnr=sum_psnr+PSNR(i);
sum_mse=sum_mse+MSE(i);
 end
psnr_used=(sum_psnr- PSNR(Cover_im))/(Cover_im-1);
mse_ used =(sum_mse - MSE(Cover_im))/(Cover_im-1);
tim_ used =(sum_time- tim(Cover_im))/(Cover_im-1);
psnr_set=sum_psnr/Cover_im;
mse_set=sum_mse/Cover_im;
tim_set=sum_time/Cover_im;
sprintf(' Average on fully used  ')
disp('=================================================')
sprintf('%.2f%.2f%.2f%.2f%.2f',psnr_used,mse_used,tim_used,
mem,Bpp)
sprintf(' Average on fully set  ')
disp('=================================================')
sprintf('%.2f%.2f%.2f%.2f%.2f',psnr_set,mse_set,tim_set,mem,
Bpp)
```

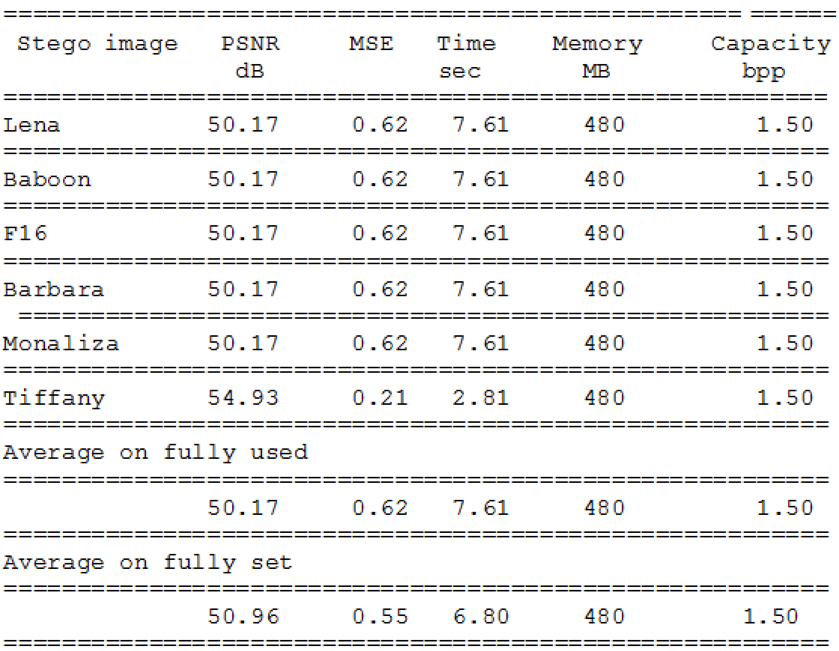
B.7 Calculation of PSNR and MSE

```
function [ My_psnr MSE ] = My_PSNR(I,J)
    X = double(I);
    Y = double(J);
    MSE = sum((X(:)-Y(:)).^2) / prod(size(X)) ;
    My_psnr = 10*log10(255 * 255/MSE);
End
```

B.8 Screenshots of GEMD Results in different values of n, L and cover images size 512×512 and 1024×1024.

B.8.1.a. Results in n=2, L=3, cover images =6, 512×512cover image size





```
================================================== ======
 Stego image     PSNR      MSE    Time     Memory     Capacity
                  dB               sec       MB          bpp
========================================================
Lena            50.17      0.62    7.61      480          1.50
========================================================
Baboon          50.17      0.62    7.61      480          1.50
========================================================
F16             50.17      0.62    7.61      480          1.50
========================================================
Barbara         50.17      0.62    7.61      480          1.50
 =======================================================
Monaliza        50.17      0.62    7.61      480          1.50
========================================================
Tiffany         54.93      0.21    2.81      480          1.50
========================================================
Average on fully used
========================================================
                50.17      0.62    7.61      480          1.50
========================================================
Average on fully set
========================================================
                50.96      0.55    6.80      480         1.50
========================================================
```

B.8.1.b. Results in n=2, L=3, cover images =2, 1024×1024 cover image size

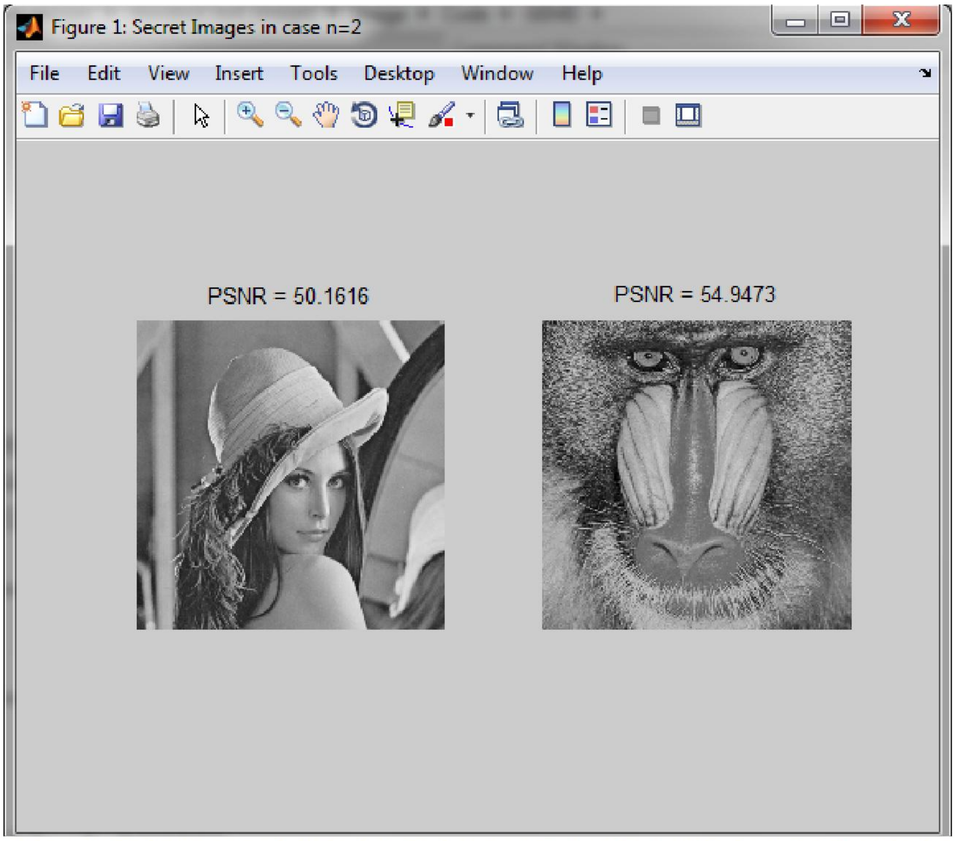
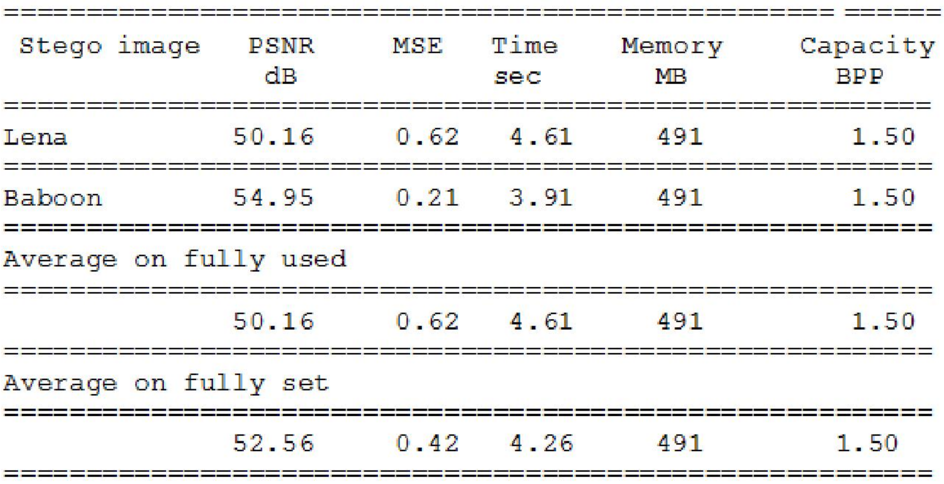
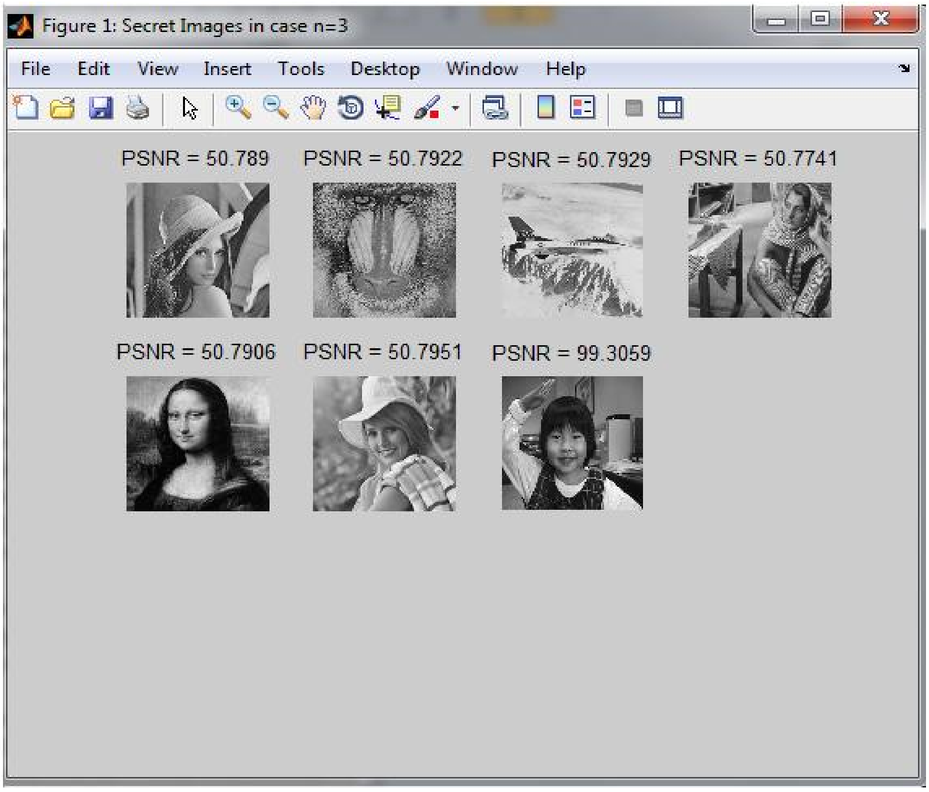


Figure 1: Secret Images in case n=2
File Edit View Insert Tools Desktop Window Help
PSNR = 50.1616
PSNR = 54.9473



```
=================================================== ======
 Stego image    PSNR     MSE   Time    Memory     Capacity
                 dB             sec      MB         BPP
==========================================================
Lena           50.16    0.62   4.61     491          1.50
==========================================================
Baboon         54.95    0.21   3.91     491          1.50
==========================================================
Average on fully used
==========================================================
               50.16    0.62   4.61     491          1.50
==========================================================
Average on fully set
==========================================================
               52.56    0.42   4.26     491         1.50
==========================================================
```

B.8.2.a Results in n=3, L=4, cover images =6, 512×512cover image size



```
================================================= ======
 stego image    PSNR     MSE    Time     Memory    Capacity
                 dB             sec       MB         bpp
=======================================================
Lena            50.79    0.54    5.84      484        1.33
=======================================================
Baboon          50.79    0.54    5.84      484        1.33
=======================================================
F16             50.79    0.54    5.84      484        1.33
=======================================================
Barbara         50.77    0.54    5.84      484        1.33
 ======================================================
Monaliza        50.79    0.54    5.84      484        1.33
=======================================================
Tiffany         50.79    0.54    5.84      484        1.33
=======================================================
Girl            99.31    0.008    0.20      484        1.33
=======================================================
Average on fully used
=======================================================
               50.79     0.54    5.84      484        1.33
=======================================================
Average on fully set
=======================================================
                57.72    0.46    5.03      484        1.33
=======================================================
```

B.8.2.b Results in n=3, L=4, cover images =2, 1024×1024cover image size

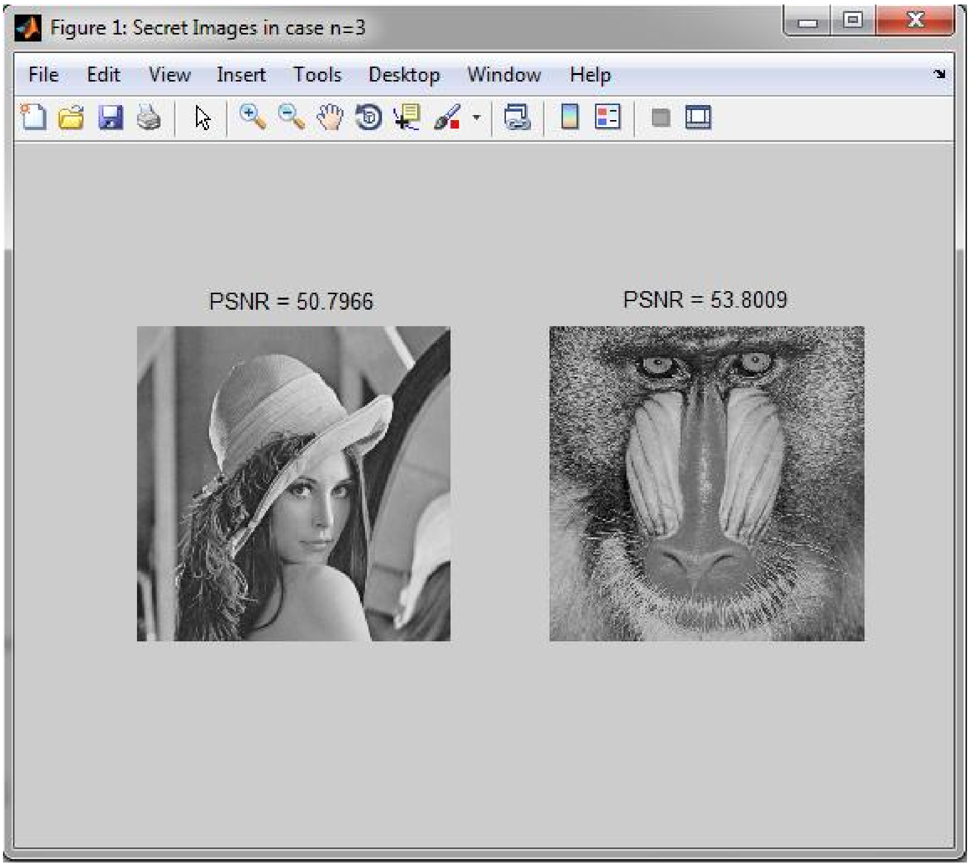
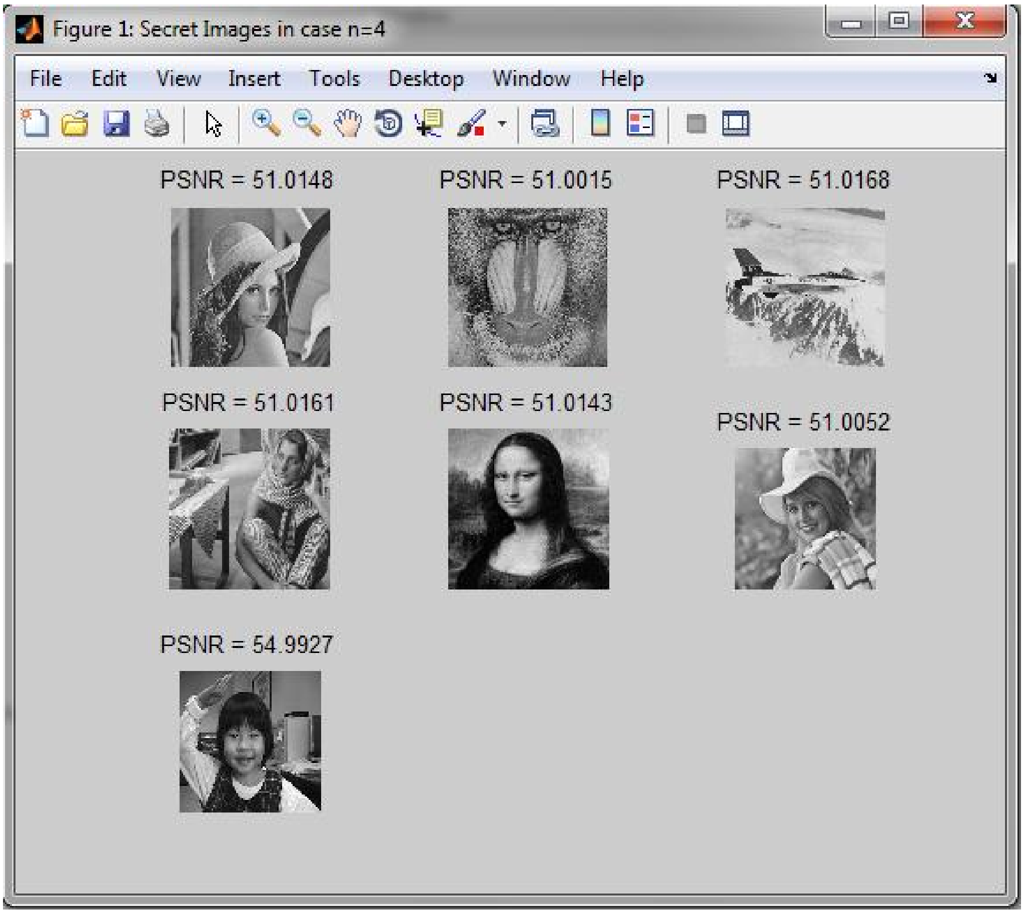
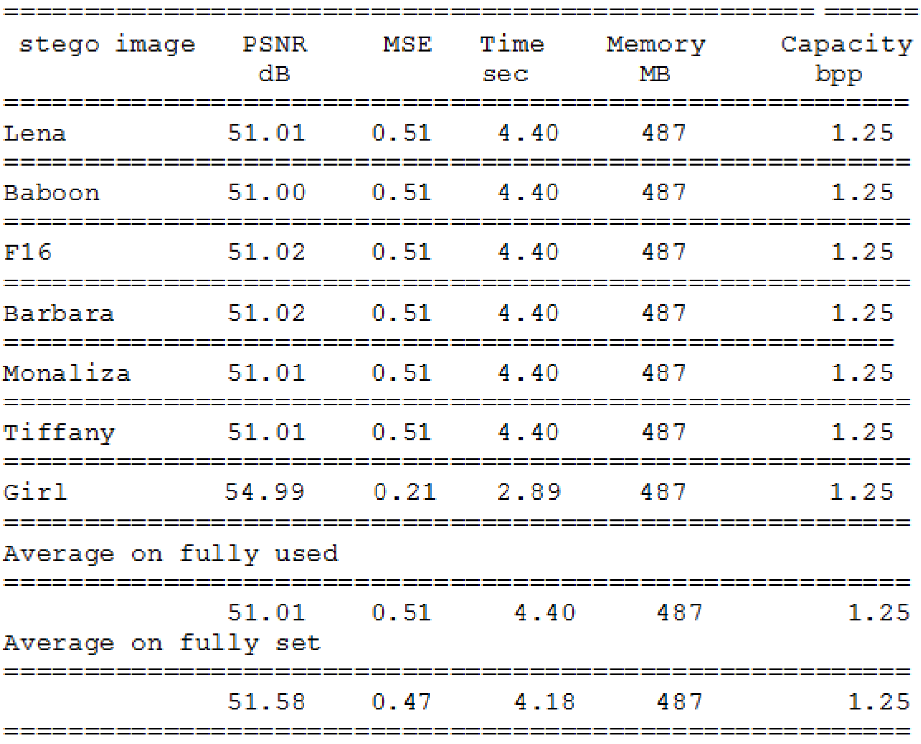


Figure 1: Secret Images in case n=3
File Edit View Insert Tools Desktop Window Help
PSNR = 50.7966
PSNR = 53.8009

```
========================================================= ======
 stego image    PSNR      MSE    Time     Memory       Capacity
                 dB               sec       MB           BPP
=========================================================
Lena              50.79     0.54    3.77      493          1.33
=========================================================
Baboon            53.80     0.27    2.84      493          1.33
=========================================================
Average on fully used
=========================================================
                 50.79      0.54    3.77       493          1.33
=========================================================
Average on fully set
=========================================================
                  52.30     0.41    3.31       493          1.33
=========================================================
```

B.8.3.a Results in n=4, L=5, cover images =7, 512×512cover image size





```
=================================================== ======
 stego image     PSNR      MSE    Time     Memory      Capacity
                  dB              sec       MB           bpp
=========================================================
Lena            51.01     0.51    4.40     487          1.25
=========================================================
Baboon          51.00     0.51    4.40     487          1.25
=========================================================
F16             51.02     0.51    4.40     487          1.25
=========================================================
Barbara         51.02     0.51    4.40     487          1.25
========================================================
Monaliza        51.01     0.51    4.40     487          1.25
=========================================================
Tiffany         51.01     0.51    4.40     487          1.25
=========================================================
Girl            54.99     0.21    2.89     487          1.25
=========================================================
Average on fully used
=========================================================
                51.01     0.51     4.40     487          1.25
Average on fully set
=========================================================
                51.58     0.47     4.18     487          1.25
=========================================================
```

B.8.3.b Results in n=4, L=5, cover images =2, 1024×1024 cover image size

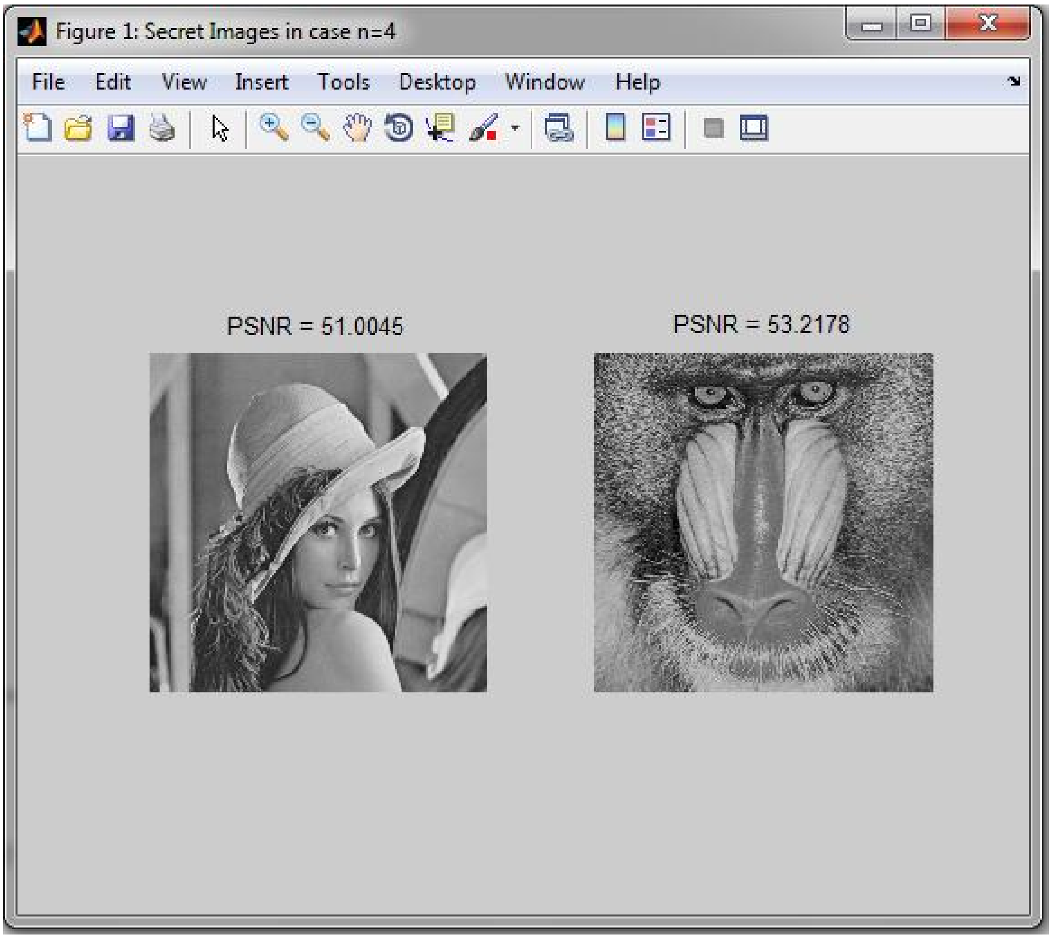
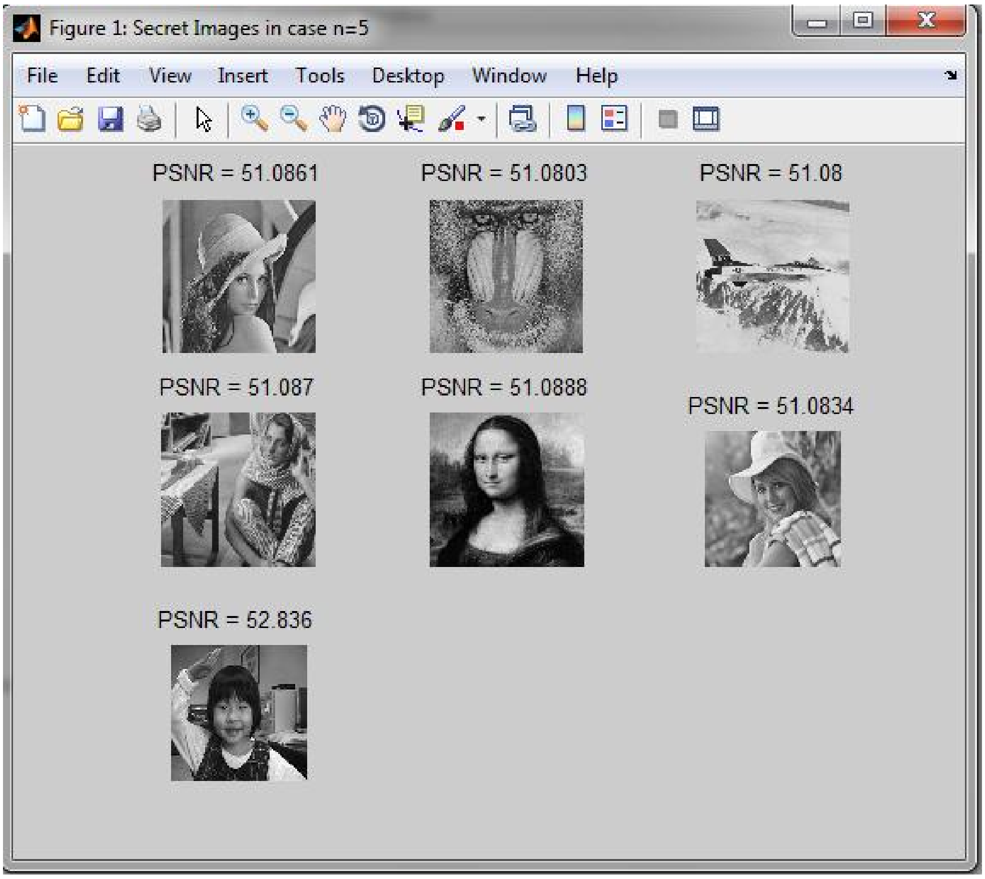


Figure 1: Secret Images in case n=4
File Edit View Insert Tools Desktop Window Help
PSNR = 51.0045
PSNR = 53.2178

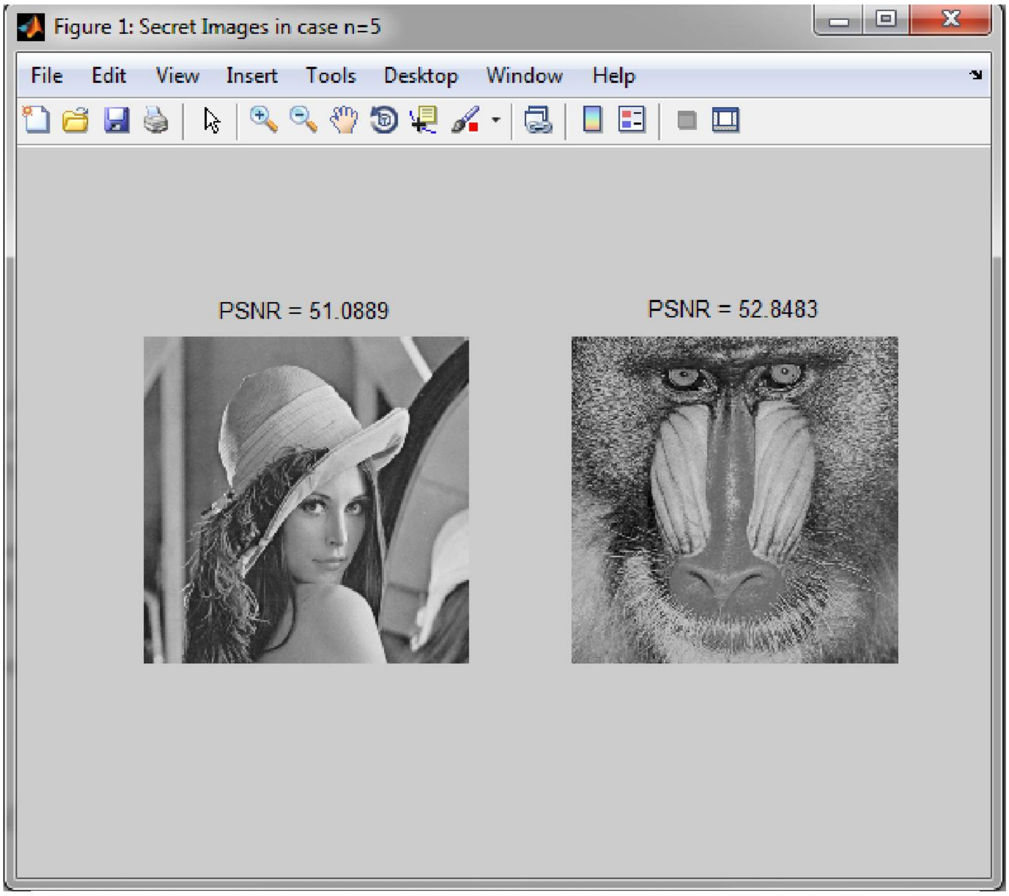
```
========================================================= ======
 Stego image    PSNR      MSE    Time     Memory      Capacity
                 dB              sec        MB           BPP
=========================================================
Lena            51.00     0.51    2.92       496          1.25
=========================================================
Baboon          53.22     0.31    2.01       496          1.25
=========================================================
Average on fully used
=========================================================
                51.00     0.51    2.92       496          1.25
Average on fully set
=========================================================
                52.11     0.41    2.47       496          1.25
=========================================================
```

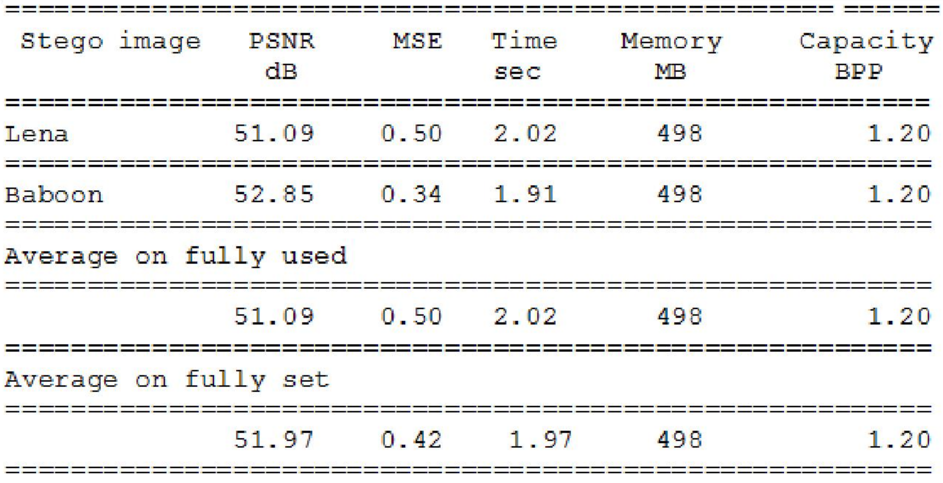
B.8.4.a Results in n=5, L=6, cover images =7, 512×512cover image size



```
=================================================== ======
 stego image    PSNR      MSE    Time     Memory     Capacity
                 dB               sec       MB          bpp
==========================================================
Lena           51.09     0.50    3.63      491          1.20
==========================================================
Baboon         51.08     0.50    3.63      491          1.20
==========================================================
F16            51.08     0.50    3.63      491          1.20
==========================================================
Barbara        51.09     0.50    3.63      491          1.20
 =========================================================
Monaliza       51.09     0.50    3.63      491          1.20
==========================================================
Tiffany        51.08     0.50    3.63      491          1.20
==========================================================
Girl           52.84     0.34    2.19      491          1.20
==========================================================
Average on fully used
==========================================================
               51.09     0.50     3.6       491         1.20
==========================================================
Average on fully set
==========================================================
               51.33     0.48      3.42      491        1.20
==========================================================
```

B.8.4.b Results in n=5, L=6, cover images =2, 1024×1024cover image size





```
======================================================= ======
 Stego image     PSNR      MSE    Time     Memory     Capacity
                  dB               sec       MB          BPP
==============================================================
Lena            51.09     0.50    2.02      498          1.20
==============================================================
Baboon          52.85     0.34    1.91      498          1.20
==============================================================
Average on fully used
==============================================================
                51.09     0.50    2.02      498          1.20
==============================================================
Average on fully set
==============================================================
                51.97     0.42     1.97     498          1.20
==============================================================
```

B.9 GEMD Extraction phase

```
options.Interpreter = 'tex';
options.Default = 'Yes';
qstring = 'Do you want to extract data?';
choice=questdlg(qstring,'EXTRACTION','Yes','No',options);
switch choice
    case 'Yes'
 for I=1:Cover_im
   Stegos1(1,:,I)=reshap_im(Stegos(:,:,I),M,N);
 End
[secret_message]= EXTRACTION( Stegos1,H,n); %get binary stream 
from the EXTRACTION function
     i=1;
    for r=1:M*N
bmess=secret_message((i-1)*8+1:i*8); %get 8 bits block from 
extracted binary stream
 a=bin2dec(num2str(bmess)); %convert to decimal
 d_msg=[d_msg a];
 i=i+1;
     end
secret_im=reshap_im2(d_msg,M,N); %reshape d_msg as 2 
dimensional array
set(gcf, 'name',' Extracted Secret image in case n=2');
imshow(uint8(secret_im)); %show the Extracted Secret image
    case 'No' break;
         end
```

B.10 Extraction function

```
function [B_msg]=  EXTRACTION(Stegos1,H,n )
B_msg=[];
   for i=1:H
group= Stegos1((i-1)*n+1:i*n); %get n pixel group from stego 
image
    sum=0;
     for j=1:n
sum = sum + double( group(j)) *((2^j)-1);
      end
t=mod(sum,(2^(n+1))); %calculate extraction function as in 
(2.9)
bin=dec2bin(t,n+1); %convert to decimal
B_msg=[B_msg bin];
   end
  B_msg;
 End
```

B.11 Reshaping secret image as two dimensions

```
function [secret_im] = reshap_im2(d_msg,M,N)
for i=1:M
    for j=1:N
      secret_im (i,j)= d_msg((i-1)*N+j);
    end
end
end
```

Appendix C: Screenshots of EMD results using 512×512 and 1024×1024 cover image size

C.1 EMD results

```
%PSNR results
n=[2 3 4 5];
PSNR_EMD=[52.11  53.57  54.66  55.53 ];
pl1=plot(n,PSNR_EMD, '-bx','LineWidth',1.5 );
title(' PSNR of EMD method for 512×512 and 1024×1024cover
images')
xlabel('n');
ylabel('PSNR(dB)');
grid on;
```

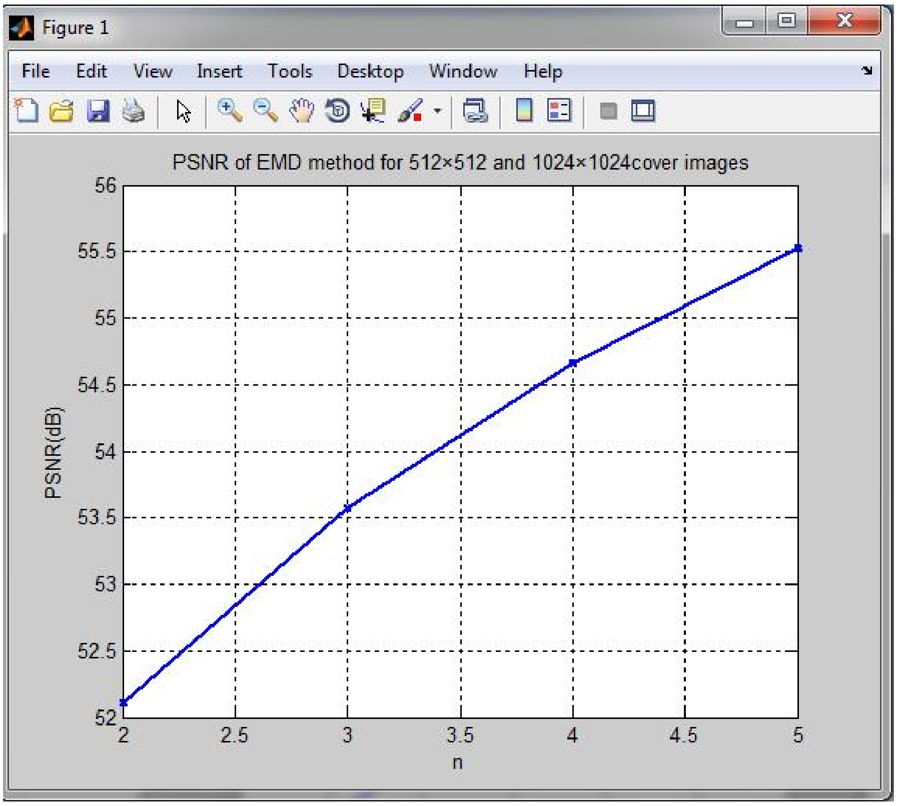


Figure 1
File Edit View Insert Tools Desktop Window Help
PSNR of EMD method for 512×512 and 1024×1024cover images
56
55.5
55
54.5
54
53.5
53
52.5
52
PSNR(dB)
2
2.5
3
3.5
4
4.5
5
n

```
%MSE results
n=[2 3 4 5];
MSE_EMD=[0.40  0.28  0.22  0.18 ]
pl1=plot(n,MSE_EMD, '-bx','LineWidth',1.5 );
title(' MSE of EMD method for 512×512 and 1024×1024
 cover images ')
xlabel('n');
ylabel('MSE');
grid on;
```

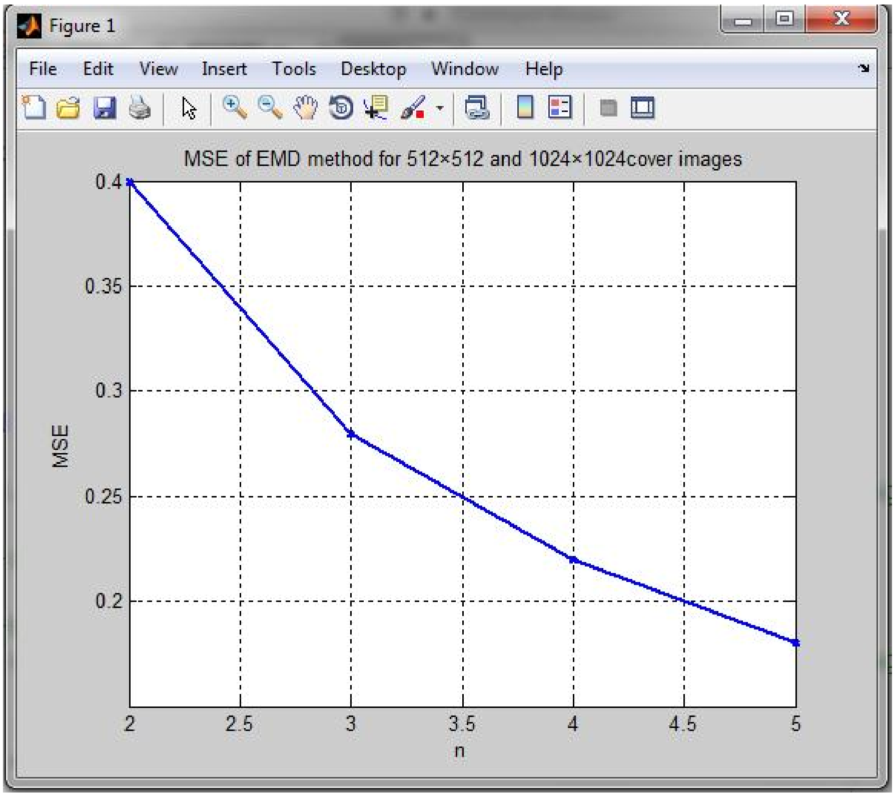
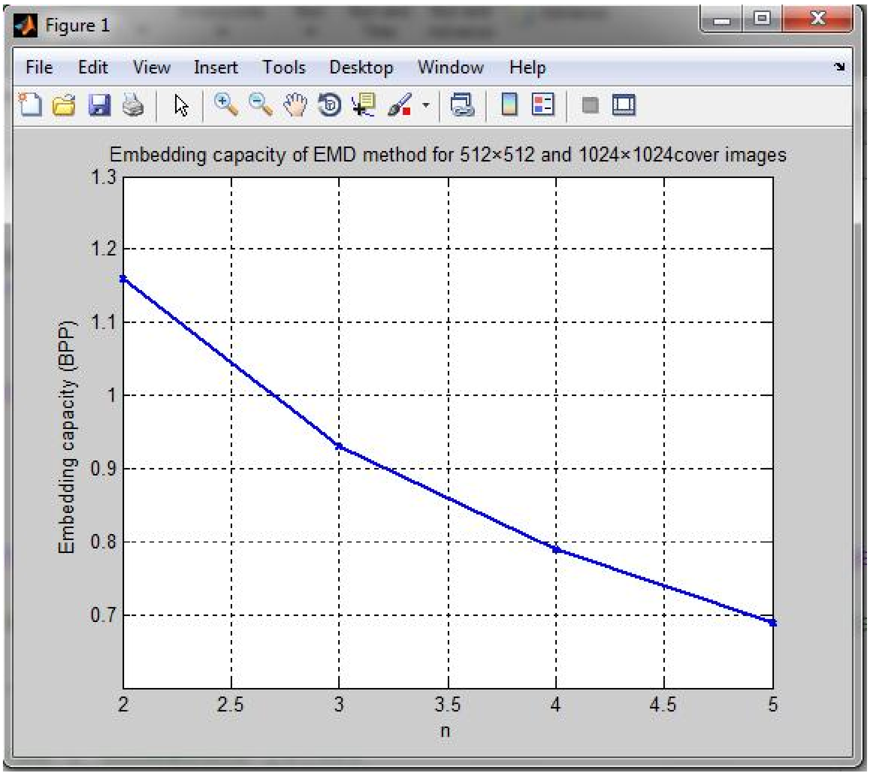
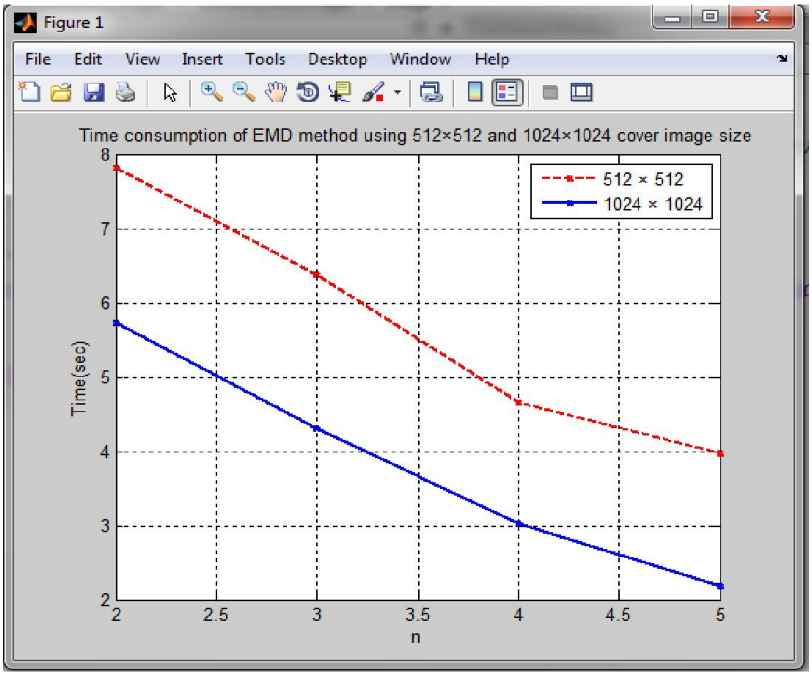


Figure 1
File Edit View Insert Tools Desktop Window Help
MSE of EMD method for 512×512 and 1024×1024cover images
MSE
0.4
0.35
0.3
0.25
0.2
2
2.5
3
3.5
4
4.5
5
n

```
%Embedding capacity BPP results
n=[2 3 4 5];
bpp_EMD=[1.16  0.93  0.79  0.69];
pl1=plot(n,bpp_EMD, '-bx','LineWidth',1.5 );
title(' Embedding capacity of EMD method for 512×512 and
1024×1024cover images')
xlabel('n');
ylabel('Embedding capacity (BPP) ');
grid on;
```



```
% Time consumption results
n=[2 3 4 5];
EMD_512=[7.81   6.38    4.66    3.97]
EMD_1024=[5.72 4.31 3.02 2.18]
pl1=plot(n,EMD_512, '--rx','LineWidth',1.5 );
hold on;
pl2=plot(n,EMD_1024, '-bx','LineWidth',1.5);
title('Time consumption of EMD method using 512×512 and
1024×1024 cover image size')
xlabel('n');
ylabel('Time(sec)');
legend([pl1, pl2], '512 × 512', '1024 × 1024');
grid on;
```



```
% Memory consumption results

n=[2 3 4 5];
EMD_512=[481    486 490 496] % values are obtained from memory
average in EMD and GEMD screenshots results in  A.8 and B.8
EMD_1024=[493   495 497 500]
pl1=plot(n,EMD_512, '--rx','LineWidth',1.5 );
hold on;
pl2=plot(n,EMD_1024, '-bx','LineWidth',1.5);
title('Memory consumption of EMD method using 512×512 and
1024×1024 cover image size')
xlabel('n');
ylabel('Memory(MB)');
legend([pl1, pl2], '512 × 512', '1024 × 1024');
grid on;
```

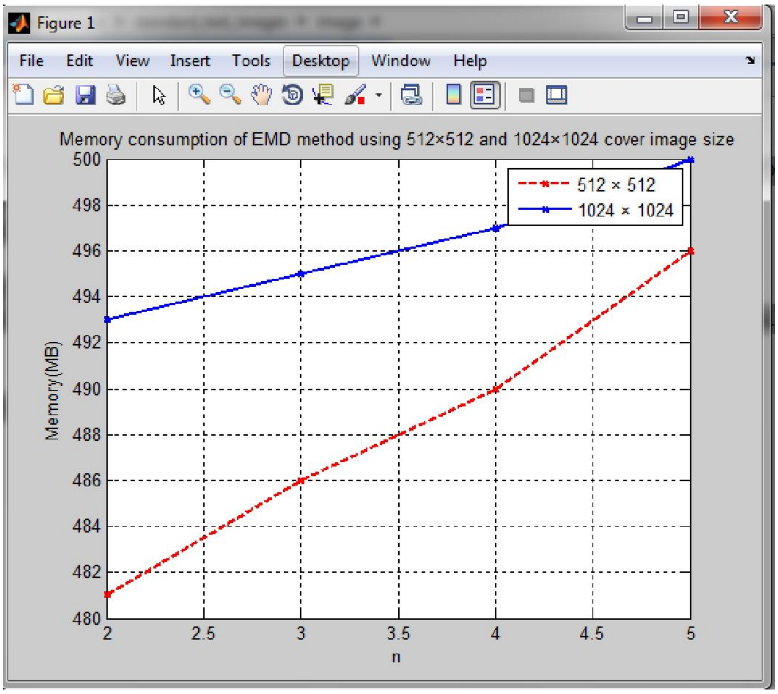
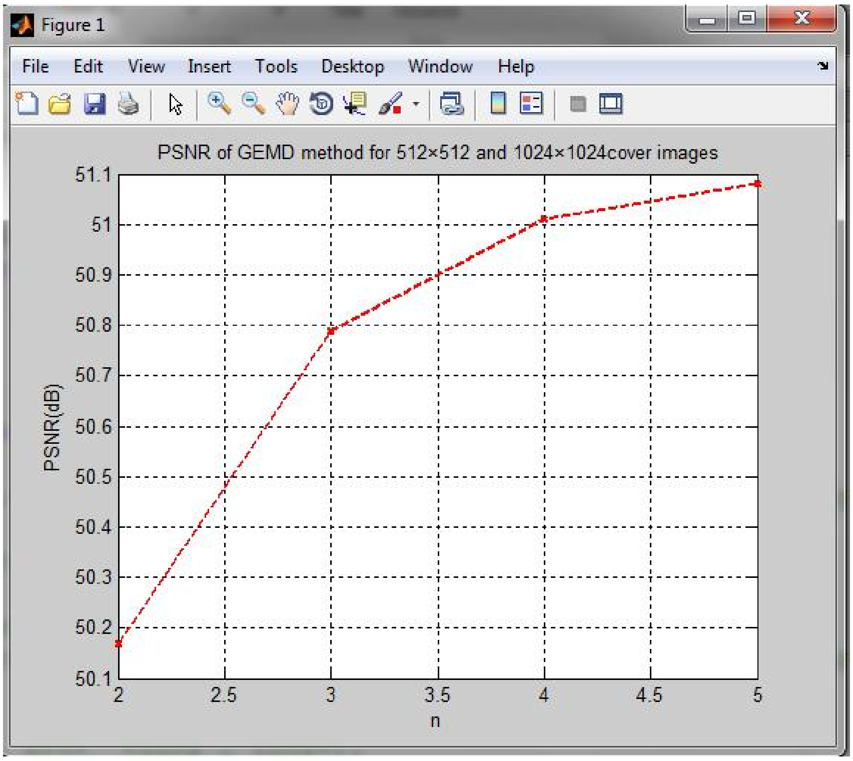


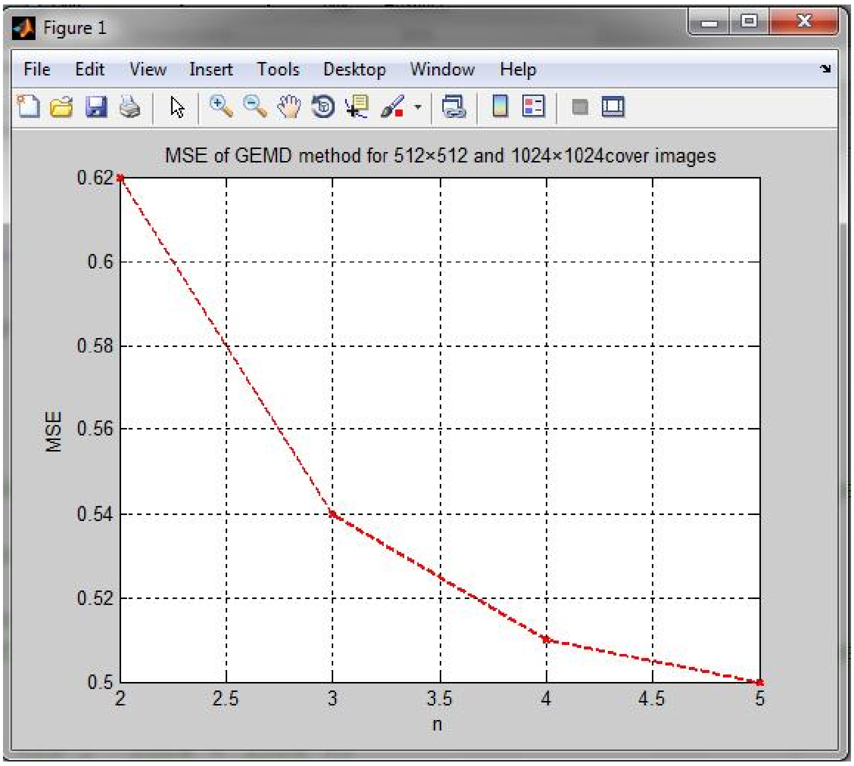
Figure 1
File Edit View Insert Tools Desktop Window Help
Memory consumption of EMD method using 512×512 and 1024×1024 cover image size
500
498
496
494
492
490
488
486
484
482
480
Memory(MB)
2
2.5
3
3.5
4
4.5
5
n
512 × 512
1024 × 1024

C.2 GEMD results

```
%PSNR results
n=[2 3 4 5];
PSNR_GEMD=[50.17  50.79  51.01  51.08];
pl1=plot(n,PSNR_GEMD, '--rx','LineWidth',1.5 );
title(' PSNR of GEMD method for 512×512 and 1024×1024cover
images')
xlabel('n');
ylabel('PSNR(dB)');
grid on;
```



```
% MSE results
n=[2 3 4 5];
MSE_GEMD=[0.62  0.54  0.51 0.50]
pl1=plot(n,MSE_GEMD, '--rx','LineWidth',1.5 );
title(' MSE of GEMD method for 512×512 and 1024×1024cover
images ')
xlabel('n'); ylabel('MSE');
grid on;
```



```
% Embedding capacity BPP results
n=[2 3 4 5];
bpp_GEMD=[1.50  1.33 1.25 1.20];
pl1=plot(n,bpp_GEMD, '--rx','LineWidth',1.5 );
title(' Embedding capacity of GEMD method for 512×512 and
1024×1024 cover images')
xlabel('n'); ylabel('Embedding capacity (bpp) ');grid on;
```

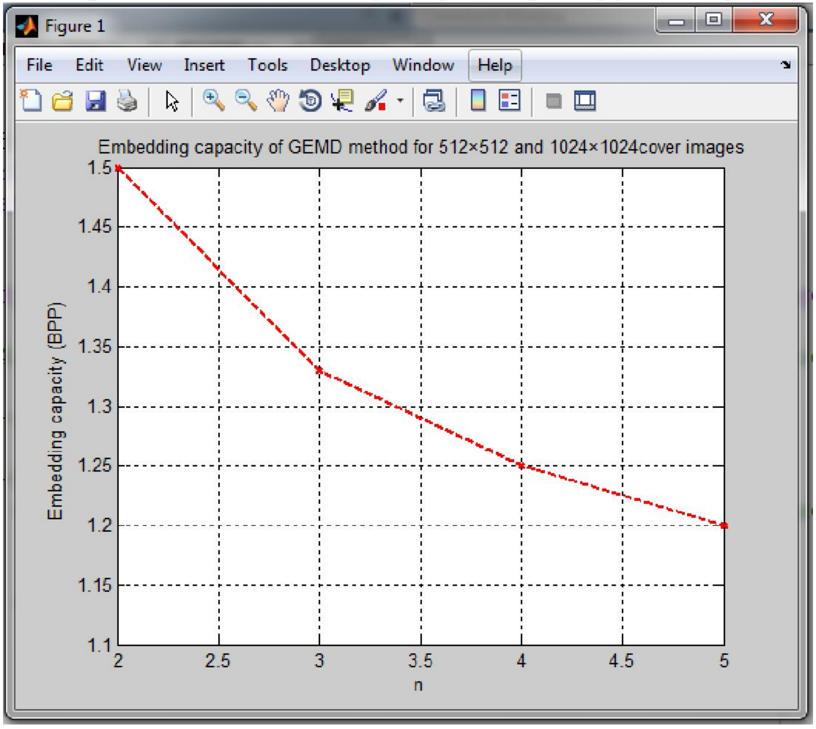
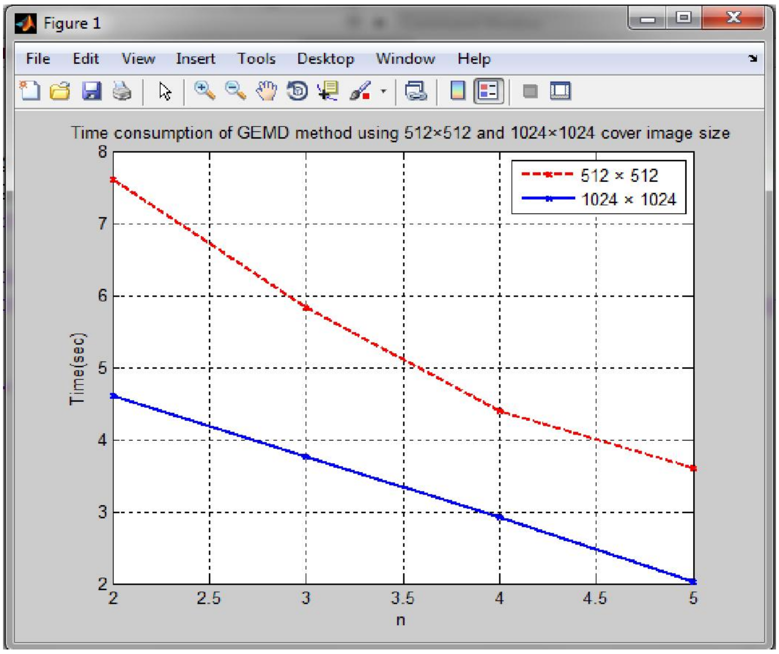


Figure 1
File Edit View Insert Tools Desktop Window Help
Embedding capacity of GEMD method for 512×512 and 1024×1024cover images
Embedding capacity (BPP)
1.5
1.45
1.4
1.35
1.3
1.25
1.2
1.15
1.1
2
2.5
3
3.5
4
4.5
5
n

```
%Time consumption results
n=[2 3 4 5];
GEMD_512=[7.61  5.84    4.40    3.60]
GEMD_1024=[4.61 3.77    2.92    2.02]
pl1=plot(n,GEMD_512, '--rx','LineWidth',1.5 );hold on;
pl2=plot(n,GEMD_1024, '-bx','LineWidth',1.5);
title('Time consumption of GEMD method using 512×512 and
1024×1024 cover image size')
xlabel('n');ylabel('Time(sec)');
legend([pl1, pl2], '512 × 512', '1024 × 1024');grid on;
```



```
% Memory consumption results
n=[2 3 4 5];
GEMD_512=[480   484 487 491]
GEMD_1024=[491  493 496 498]
pl1=plot(n,GEMD_512, '--rx','LineWidth',1.5 );
hold on;
pl2=plot(n,GEMD_1024, '-bx','LineWidth',1.5);
title('Memory consumption of GEMD method using 512×512 and
1024×1024 cover image size')
xlabel('n');ylabel('Memory(MB)');
legend([pl1, pl2], '512 × 512', '1024 × 1024');grid on;
```

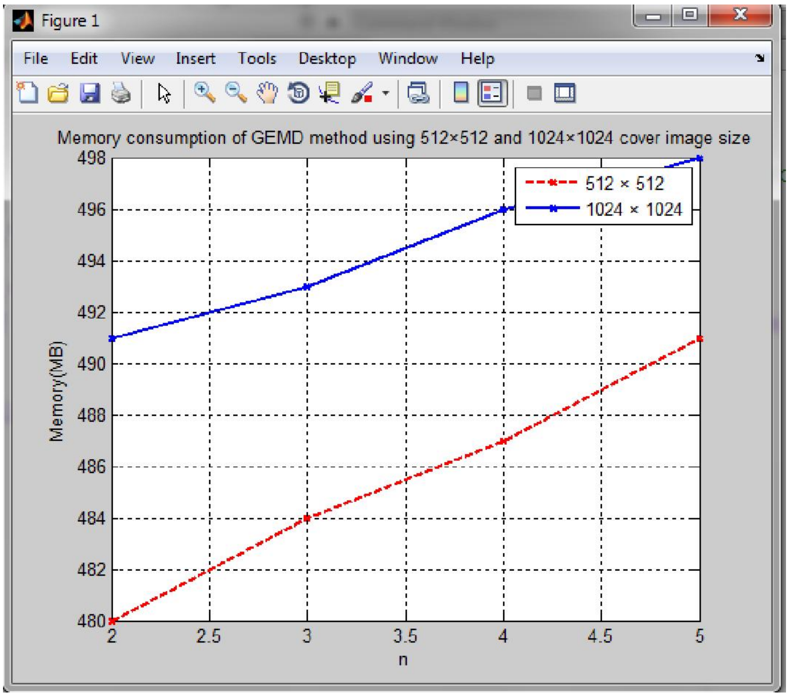


Figure 1
File Edit View Insert Tools Desktop Window Help
Memory consumption of GEMD method using 512×512 and 1024×1024 cover image size
Memory(MB)
n
512 × 512
1024 × 1024