Coursework for Introducing Matlab to Communications Engineers

Robert G. Maunder, *Senior Member, IEEE*

Abstract

This coursework is intended to accompany a lecture and laboratory series that introduces the syntax and operation of Matlab, possibly at the beginning of a master-level degree course on communications engineering. Owing to this, the coursework is specifically designed to have relevance to communication engineers, granting them the ability to use Matlab to design, simulate and characterize communication systems. The coursework assumes no prior knowledge or experience in communication engineering or Matlab, but is designed to be interesting and challenging to students that do have this knowledge or experience. The coursework considers the most relevant aspects of Matlab, namely matrix manipulation, programming, Monte Carlo simulation and error ratio plotting. It also requires students to design and optimize innovative solutions to problems with no single correct answer. The students may be expected to spend around 50 hours completing this coursework and they are assessed on the functionality, efficiency and human-readability of their solutions. Automated marking may be used to assess the functionality and efficiency of the solutions, leaving only the human-readability requiring manual marking. The coursework has been successfully used and refined over several years at the University of Southampton, UK.

INTRODUCTION

In the exercises detailed below, you are asked to write some Matlab code and produce some results. Although you may verbally discuss your ideas with your classmates, you should not show them your Matlab code or results. When you are finished, you should copy and paste your Matlab code and results into a Word document, which you should then save as a .pdf file. This Word document should only include the Matlab code and results that are directly requested in the **bold** paragraphs below - it should not include anything else. In particular, there is no need to write a report for this coursework. When you are finished, you need to make an electronic submission of the .pdf version of your Word document, your Matlab code and your result files before the deadline. You also need to print out your Word document and submit it before the same deadline.

As shown in the marking scheme of Table I, one third of your marks in this coursework depend on each of the functionality, efficiency and human-readability of your Matlab code. You can see how to interpret your marks in Table II. The human-readability of your Matlab code will be marked by looking at your Word document. The results in your Word document will also be used to assess some aspects of your Matlab code's functionality and efficiency. However, an automated marking system will be used to run your Matlab code, in order to assess its functionality and efficiency more thoroughly. This will be achieved using test Matlab scripts that are similar to the ones that accompany these coursework instructions. However, the versions of these scripts that will be used for marking include some additional tests, so you need to ensure that your Matlab code works in the general case, not just for specific tests. Some of the tests in these scripts produce measurements, which will be used to mark the functionality and efficiency of your Matlab code. Some of the tests produce pass or fail messages, which will affect the marks awarded for your Matlab code's functionality. Note that some of these tests deliberately give erroneous inputs to your Matlab code. In order to pass these tests, your Matlab code will need to generate an error message using a command like the following one.

```
error('IEEEComSoc:argChk','Your error message goes here');
```

The author is with Electronics and Computer Science, University of Southampton, SO17 1BJ, United Kingdom, e-mail: rm@ecs.soton.ac.uk

[1]You can see discussions of good Matlab programming practice at http://www.mit.edu/∼pwb/cssm/GMPP.pdf and http://www.datatool.com/downloads/matlab_style_guidelines.pdf.

TABLE I
MARKING SCHEME

33.33%	Functionality. Does the Matlab code work? Does it meet the specifications provided in the exercises below? Are the results correct? Does the Matlab code generate appropriate error messages?
33.33%	Efficiency. How quickly does the Matlab code run? Has the Matlab code made good use of built in functions, vectors and matrices? Or does it use proprietary code and an excessive number of for loops?
33.33%	Human-readability. Is the Matlab code reusable, elegant and well structured? Does it follow good programming practice[1]? Does it include useful, concise and relevant comments, which help explain its non-trivial features? Are the chosen variable names descriptive, concise and relevant? Or does the Matlab code use an excessive number of lines? Does the Matlab code use `switch`, `break`, `continue` or `return` without having a really really good reason to?

TABLE II
HOW TO INTERPRET YOUR MARKS

40 out of 40	Perfect solutions. This mark is only given if there is no way to further improve the functionality, efficiency or human-readability of the Matlab code and results.
29–39 out of 40	Excellent solutions. These marks are awarded to students that have really thought about their solutions and have come up with clever ways to optimize the functionality, efficiency and human-readability of the Matlab code and results.
28 out of 40	Solid solutions. All of the results are correct and the Matlab code is reasonably functional, efficient and human-readable, but all the tricks for making them particularly so have been missed.
20–27 out of 40	Flawed solutions. Typically, these marks are given if all parts of all exercises have been completed, but there are mistakes in the results or Matlab code. For example, the Matlab code may work for the specific tests provided in the scripts that accompany these coursework instructions, but it may not always work in the general case.
1–19 out of 40	Missing solutions. Typically, marks this low are only given if some of the results or Matlab code are missing from the submission.
0 out of 40	No solutions. You would probably have to submit nothing at all to get a mark this low!

EXERCISE 1 - TWO-DIMENSIONAL PARITY CHECK ENCODER

Consider a wireless communication scheme which is designed to convey messages between a transmitter and a receiver over a wireless channel. Suppose that each message is a row vector $\mathbf{x}$ comprising k number of symbols. Each symbol has an integer value in the range 0 to $(M-1)$, where M is known as the radix of the symbols. For example, the message vector $\mathbf{x} = [2, 0, 1, 1, 3, 2]$ comprises $k = 6$ symbols, having a radix of $M = 4$.

A Two-Dimensional Parity Check (TDPC) encoder may be used in the transmitter to protect the message vector $\mathbf{x}$ from the corruption that may be imposed by the wireless channel, owing to noise, fading and interference, for example. A TDPC encoder is controlled using three parameters, namely the radix M, the number of rows i in the TDPC matrix and the number of columns j in the TDPC matrix. The TDPC encoder can be used to encode a message vector $\mathbf{x}$ comprising k symbols, provided that the radix M of the TDPC encoder and of the message are equal, and provided that the number of rows i and columns j are set such that $(i-1) \cdot (j-1) = k$.

The first step in TDPC encoding is to reshape the vector $\mathbf{x}$, so that it becomes a matrix $\mathbf{X}$, having $(i-1)$ rows and $(j-1)$ columns. Taking each of the symbols in $\mathbf{x}$ from left to right in turn, the matrix $\mathbf{X}$ is populated one column at a time, starting by filling the left-most column from top to bottom, before similarly filling each of the other columns from left to right in turn. In the case where the example message vector $\mathbf{x} = [2, 0, 1, 1, 3, 2]$ is TDPC encoded using the parameters $i = 3$ and $j = 4$, we obtain the matrix

$\mathbf{X} = [\,2\,, 1, 30, 1, 2]$, which has $(i-1) = 2$ rows and $(j-1) = 3$ columns.

In a second step, the TDPC matrix $\mathbf{Y}$ is formed by copying the matrix $\mathbf{X}$, but adding an extra row at the bottom and an extra column at the right. Therefore, the TDPC matrix $\mathbf{Y}$ has i rows and j columns. The values in the extra row are set such that each column has a sum that is a multiple of M. Likewise, the values in the extra column are set such that each row has a sum that is a multiple of M. In the case where the example message vector $\mathbf{x} = [2, 0, 1, 1, 3, 2]$ is TDPC encoded using the parameters $i = 3$ and $j = 4$, we obtain the TDPC matrix $\mathbf{Y} = [\,2\,, 1, 3, 20, 1, 2, 12, 2, 3, 1]$, which has $i = 3$ rows and $j = 4$ columns. Note that the first, second and fourth columns of this example $\mathbf{Y}$ have sums of 4, while the third column has a sum of 8. Likewise, the second row has a sum of 4, while the first and second rows have sums of 8. Note that these values of 4 and 8 are both multiples of $M = 4$. In other words, a remainder of zero results when dividing the sum of each row and column by $M = 4$, which is to say that the modulo-$M = 4$ sum of each row and each column is zero.

In a final step, the TDPC matrix $\mathbf{Y}$ is reshaped, so that it becomes a row vector $\mathbf{y}$, having a length of $(i \cdot j)$. Taking each of the columns in $\mathbf{Y}$ from left to right in turn and reading the columns from top to bottom, the row vector $\mathbf{y}$ is populated one symbol at a time, from left to right. In the case where the example message vector $\mathbf{x} = [2, 0, 1, 1, 3, 2]$ is TDPC encoded using the parameters $i = 3$ and $j = 4$, we obtain the encoded vector $\mathbf{y} = [2, 0, 2, 1, 1, 2, 3, 2, 3, 2, 1, 1]$, which comprises $n = i \cdot j = 12$ symbols.

In this exercise, your task is to write a Matlab function which performs TDPC encoding. Your Matlab function is required to use the following line, in order to declare the function's name, inputs and outputs.

```
function encoded_vector = TDPC_encoder(message_vector, radix, rows, columns)
```

Here, `message_vector` should accept the row vector $\mathbf{x}$, while `radix`, `rows` and `columns` should accept the scalars M, i and j, respectively. Your `TDPC_encoder` function should check that the values of these inputs are self-consistent and if not, it should generate an error message using the command provided on page 1. Otherwise, `encoded_vector` should return the row vector $\mathbf{y}$. You can test the functionality and efficiency of your `TDPC_encoder` function using the script `test_TDPC_encoder`, which accompanies these coursework instructions. A script similar to this one will be used to mark the functionality and efficiency of your `TDPC_encoder` function. Note that the mark that you receive for efficiency in this exercise will be capped by the mark that you receive for functionality. This is because it is easy to write a Matlab function that runs quickly, but does not have all of the required functionality.

When you are finished, include a listing of your `TDPC_encoder` function in your Word document and submit this function electronically.

EXERCISE 2 - TWO-DIMENSIONAL PARITY CHECK DECODER

In the wireless communication scheme mentioned in Exercise 1, the encoded vector $\mathbf{y}$ is conveyed between the transmitter and the receiver over a wireless channel. This channel may suffer from noise, interference and fading, causing the encoded vector to become corrupted. Owing to this, some of the symbols in the received vector $\hat{\mathbf{y}}$ may have different values to the corresponding symbols in the encoded vector $\mathbf{y}$. Note however that these erroneous symbols should still have integer values in the range 0 to $(M-1)$. For example, the wireless channel could corrupt the encoded vector $\mathbf{y} = [2, 0, 2, 1, 1, 2, 3, 2, 3, 2, 1, 1]$, resulting in the received vector $\hat{\mathbf{y}} = [2, 1, 2, 1, 1, 2, 1, 2, 3, 2, 1, 1]$, which contains two erroneous symbols.

A TDPC decoder may be employed in the receiver to mitigate the erroneous symbols in the vector $\hat{\mathbf{y}}$. A TDPC decoder is controlled using three parameters, namely the radix M, the number of rows i in the TDPC matrix and the number of columns j in the TDPC matrix. The TDPC encoder can be used to decode a received vector $\hat{\mathbf{y}}$, provided that it derives from an encoded vector $\mathbf{y}$ that has been generated using a TDPC encoder having the same parameter values.

The first step in TDPC decoding is to reshape the received vector $\hat{\mathbf{y}}$ of n symbols, so that it becomes the TDPC matrix $\hat{\mathbf{Y}}$, having i rows and j columns, where $i \cdot j = n$. Taking each of the symbols in $\hat{\mathbf{y}}$ from left to right in turn, the TDPC matrix $\hat{\mathbf{Y}}$ is populated one column at a time, starting by filling the left-most column from top to bottom, before similarly filling each of the other columns from left to right

in turn. In the case where the example received vector $\hat{\mathbf{y}} = [2, 1, 2, 1, 1, 2, 1, 2, 3, 2, 1, 1]$ is TDPC decoded using the parameters $i = 3$ and $j = 4$, we obtain the TDPC matrix $\hat{\mathbf{Y}} = [\,2\,, 1, 1, 21, 1, 2, 12, 2, 3, 1]$, which has $i = 3$ rows and $j = 4$ columns.

In a second step, the modulo-M sum of each row and each column is calculated and analyzed to identify and mitigate erroneous symbols. In the case of the example received vector $\hat{\mathbf{y}} = [2, 1, 2, 1, 1, 2, 1, 2, 3, 2, 1, 1]$, the modulo-$M = 4$ sums of the columns in $\hat{\mathbf{Y}}$ are $[1, 0, 2, 0]$, while the modulo-$M = 4$ sums of its rows are $[\,210]$. Note that the third row, the second column and the fourth column all have modulo-$M = 4$ sums of 0, just as was desired when setting the values in the extra row and column during TDPC encoding. This suggests that there are no erroneous symbols in these rows and columns. However, the modulo-$M = 4$ sums of the second row and of the first column are both equal to 1. This suggests that the symbol in the second row and the first column has an erroneous value, which has been obtained using the modulo-$M = 4$ addition of the value 1 to the correct symbol value. More specifically, this suggests that the correct value for this symbol is 0, rather than 1. Likewise, the modulo-$M = 4$ sums of the first row and of the third column are both equal to 2. This suggests that the symbol in the first row and the third column has an erroneous value, which has been obtained using the modulo-$M = 4$ addition of the value 2 to the correct symbol value. More specifically, this suggests that the correct value for this symbol is 3, rather than 1. Using this logic, it can be inferred that the correct value for the TDPC matrix is $\hat{\mathbf{Y}} = [\,2\,, 1, 3, 20, 1, 2, 12, 2, 3, 1]$.

In a fourth step, the bottom row and the right-most column of the TDPC matrix $\hat{\mathbf{Y}}$ are removed, to obtain the matrix $\hat{\mathbf{X}}$. In the case of the example received vector $\hat{\mathbf{y}} = [2, 1, 2, 1, 1, 2, 1, 2, 3, 2, 1, 1]$, removing this row and this column from the TDPC matrix $\hat{\mathbf{Y}}$ results in the matrix $\hat{\mathbf{X}} = [\,2\,, 1, 30, 1, 2]$, which has $(i - 1) = 2$ rows and $(j - 1) = 3$ columns.

In a final step, the matrix $\hat{\mathbf{X}}$ is reshaped, so that it becomes a row vector $\hat{\mathbf{x}}$, having a length of $k = (i - 1) \cdot (j - 1)$. Taking each of the columns in $\hat{\mathbf{X}}$ from left to right in turn and reading the columns from top to bottom, the row vector $\hat{\mathbf{x}}$ is populated one symbol at a time, from left to right. In the case where the example received vector $\hat{\mathbf{y}} = [2, 1, 2, 1, 1, 2, 1, 2, 3, 2, 1, 1]$ is TDPC decoded using the parameters $i = 3$ and $j = 4$, we obtain the decoded vector $\hat{\mathbf{x}} = [2, 0, 1, 1, 3, 2]$, which comprises $k = (i - 1) \cdot (j - 1) = 6$ symbols. Note this decoded vector $\hat{\mathbf{x}}$ is identical to the message vector $\mathbf{x}$ from Exercise 1, indicating that all erroneous symbols in the vector $\hat{\mathbf{y}}$ have been successfully mitigated.

Note that in the example above, the symbol errors have occurred in different rows and columns of the TDPC matrix $\hat{\mathbf{Y}}$. Also, the rows and columns have the same set of modulo-M sums, with no repetitions within this set. This has made it easy to mitigate all of the symbol errors. However, on other occasions it can be more difficult or impossible to mitigate all of the symbol errors. For example, some symbol errors may occur in the same row or column. Sometimes, these symbol errors can cancel each other out, causing the row or column to have a modulo-M sum of zero. Other times, these symbol errors can cause the rows and columns to have different sets of modulo-M sums. Furthermore, some rows or some columns may have the same modulo-M sums.

In this exercise, your task is to write a Matlab function which performs TDPC decoding and mitigates as many symbol errors as you can. Your Matlab function is required to use the following line, in order to declare the function's name, inputs and outputs.

```
function decoded_vector = TDPC_decoder(received_vector, radix, rows, columns)
```

Here, `received_vector` should accept the row vector $\hat{\mathbf{y}}$, while `radix`, `rows` and `columns` should accept the scalars M, i and j, respectively. Your `TDPC_decoder` function should check that the values of these inputs are self-consistent and if not, it should generate an error message using the command provided on page 1. Otherwise, `decoded_vector` should return the row vector $\hat{\mathbf{x}}$. You can test the functionality and efficiency of your `TDPC_decoder` function using the script `test_TDPC_decoder`, which accompanies these coursework instructions. Here, tests 1 to 7 can be used to evaluate the functionality of your `TDPC_decoder` function. Tests 8 to 17 can be used to evaluate whether your `TDPC_decoder` function can mitigate particular combinations of symbol errors. It is possible to write a `TDPC_decoder` function that can mitigate all of

the symbol errors in tests 8 to 17, although some require much more complicated programming than others. You may like to finish Exercises 3 and most of Exercise 4, before attempting to pass all of tests 8 to 17. Finally, tests 18 to 20 can be used to evaluate the overall error mitigation and efficiency of your `TDPC_decoder` function. A script similar to the one that accompanies these coursework instructions will be used to mark the functionality and efficiency of your `TDPC_decoder` function. All of the tests will be considered when marking the functionality of your `TDPC_decoder` function, but particular attention will be paid to the results of tests 18 to 20. Note that the mark that you receive for efficiency in this exercise will be capped by the mark that you receive for functionality. This is because it is easy to write a Matlab function that runs quickly, but does not have all of the required functionality.

When you are finished, include a listing of your `TDPC_decoder` function in your Word document and submit this function electronically.

EXERCISE 3 - M-ARY SYMMETRIC CHANNEL

In the wireless communication scheme mentioned in Exercises 1 and 2, an encoded vector $\mathbf{y}$ of n symbols having values in the range 0 to $(M-1)$ is conveyed over a wireless channel. The resulting received vector $\hat{\mathbf{y}}$ also comprises n symbols having integer values in the range 0 to $(M-1)$, although some of these may have different values to the corresponding symbols in the encoded vector $\mathbf{y}$. The occurrence of these symbol errors may be (crudely) modeled using an M-ary symmetric channel.

An M-ary symmetric channel is parametrized by two parameters, namely the symbol error probability P_e in the range 0 to 1 and the radix M, which is required to equal the radix M of the encoded vector $\mathbf{y}$. The transmission of each of the n symbols in the encoded vector $\mathbf{y}$ is modeled separately and independently, by generating a random number. Depending on the value of the random number, the value of symbol will be either copied to the corresponding symbol in the received vector $\hat{\mathbf{y}}$ without error, or will be replaced with a different value in the range 0 to $(M-1)$. This is performed such that a symbol error is avoided with a probability of $(1-P_e)$ and incurred with a probability of P_e. When a symbol error occurs, the value for the symbol in the received vector $\hat{\mathbf{y}}$ is selected from among the $(M-1)$ possible incorrect values, each with the same probability of $1/(M-1)$. Therefore, the overall probability of receiving a particular one of these incorrect values is given by $P_e \cdot 1/(M-1)$.

For example, consider the transmission of an $M=4$-ary symbol value of 2 over an $M=4$-ary symmetric channel having a symbol error probability of $P_e = 0.3$. With a probability of $(1-P_e) = 0.7$, the symbol will be correctly received with a value of 2. However, with a probability of $P_e = 0.3$, the symbol will be received incorrectly, with a value of either 0, 1 or 3. In this event, each of these incorrect values is associated with an equal probability of $1/(M-1) = 0.333$. Therefore, the overall probability of receiving the incorrect value of 0 is given by $P_e \cdot 1/(M-1) = 0.1$. Likewise, the probabilities of receiving the incorrect values of 1 and 3 are also both equal to 0.1. Note that the probabilities of receiving the four possible symbol values in the set $\{0,1,2,3\}$ are given by the set $\{0.1, 0.1, 0.7, 0.1\}$, which sum to 1.0, as may be expected.

In this exercise, your task is to write a Matlab function which simulates transmission over an M-ary symmetric channel. Your Matlab function is required to use the following line, in order to declare the function's name, inputs and outputs.

```
function received_vector = MSC(encoded_vector, radix, Pe)
```

Here, `encoded_vector` should accept the row vector $\mathbf{y}$, while `radix` and `Pe` should accept the scalars M and P_e, respectively. Your `MSC` function should check that the values of these inputs are self-consistent and if not, it should generate an error message using the command provided on page 1. Otherwise, `received_vector` should return the row vector $\hat{\mathbf{y}}$. You can test the functionality and efficiency of your `MSC` function using the script `test_MSC`, which accompanies these coursework instructions. A script similar to this one will be used to mark the functionality and efficiency of your `MSC` function. Note that the mark that you receive for efficiency in this exercise will be capped by the mark that you receive for functionality. This is because it is easy to write a Matlab function that runs quickly, but does not have all of the required functionality.

When you are finished, include a listing of your `msc` function in your Word document and submit this function electronically.

EXERCISE 4 - MONTE-CARLO SIMULATION

The error mitigation ability of the wireless communication scheme mentioned in Exercises 1, 2 and 3 may be characterized using a Monte-Carlo simulation. Here, we can simulate the TDPC encoding of lots of message vectors $\mathbf{x}$, as well as their transmission over an M-ary symmetric channel having a particular symbol error probability P_e, while counting the total number of symbol errors in the decoded vectors $\hat{\mathbf{x}}$ produced by the TDPC decoder. The Symbol Error Ratio (SER) is given by the total number of symbol errors in the decoded vectors $\hat{\mathbf{x}}$ divided by the total number of symbols in the message vectors $\mathbf{x}$. We can repeat this experiment for a range of different values for the symbol error probability P_e and plot the resultant SERs against P_e on logarithmic axes, as exemplified in Figure 1. If a sufficiently high number of message vectors $\mathbf{x}$ have been used during the simulation for each value of the symbol error probability P_e, then a nice smooth SER plot will result. Note that the plots of Figure 1 were obtained by using 16 cores on a high-performance computer for 24 hours. Each data point was obtained by continuing to generate new message vectors $\mathbf{x}$, until 10000 symbol errors had been observed in the corresponding decoded vectors $\hat{\mathbf{x}}$.

In this exercise, your task is to write a Matlab script or function that performs a Monte-Carlo simulation of the wireless communication scheme mentioned in Exercises 1, 2 and 3. This script or function should use an ASCII text file to save the total number of symbol errors observed in the decoded vectors $\hat{\mathbf{x}}$, as well as the total number of symbols in the message vectors $\mathbf{x}$, for each value of the symbol error probability P_e considered. Your Matlab script or function should also plot the SER against the symbol error probability P_e on logarithmic axes, as exemplified in Figure 1. Alternatively, you may prefer to use a second Matlab script or function to perform the plotting. In this exercise, you may find it helpful to use a number of Matlab's built in functions, such as `save`, `load`, `figure`, `subplot`, `title`, `xlabel`, `ylabel`, `xlim`, `ylim`, `box`, `hold`, `loglog` and `legend`.

The functionality of your Matlab code will be marked by assessing the quality of your SER plot(s). More specifically, these should contain nice smooth SER curves that are well annotated, that consider a good selection of parameter values and that include results for a good selection of symbol error probability P_e values. The efficiency of your Matlab code will be marked by looking at your ASCII text file. In particular, each SER value should have been calculated using an appropriate number of symbol errors in the decoded vectors $\hat{\mathbf{x}}$, as well as an appropriate number of symbols in the message vectors $\mathbf{x}$.

When you are finished, include a listing of your Matlab scripts and/or functions in your Word document, together with your SER plot(s) and one example of your ASCII text files. Also, submit these items electronically.

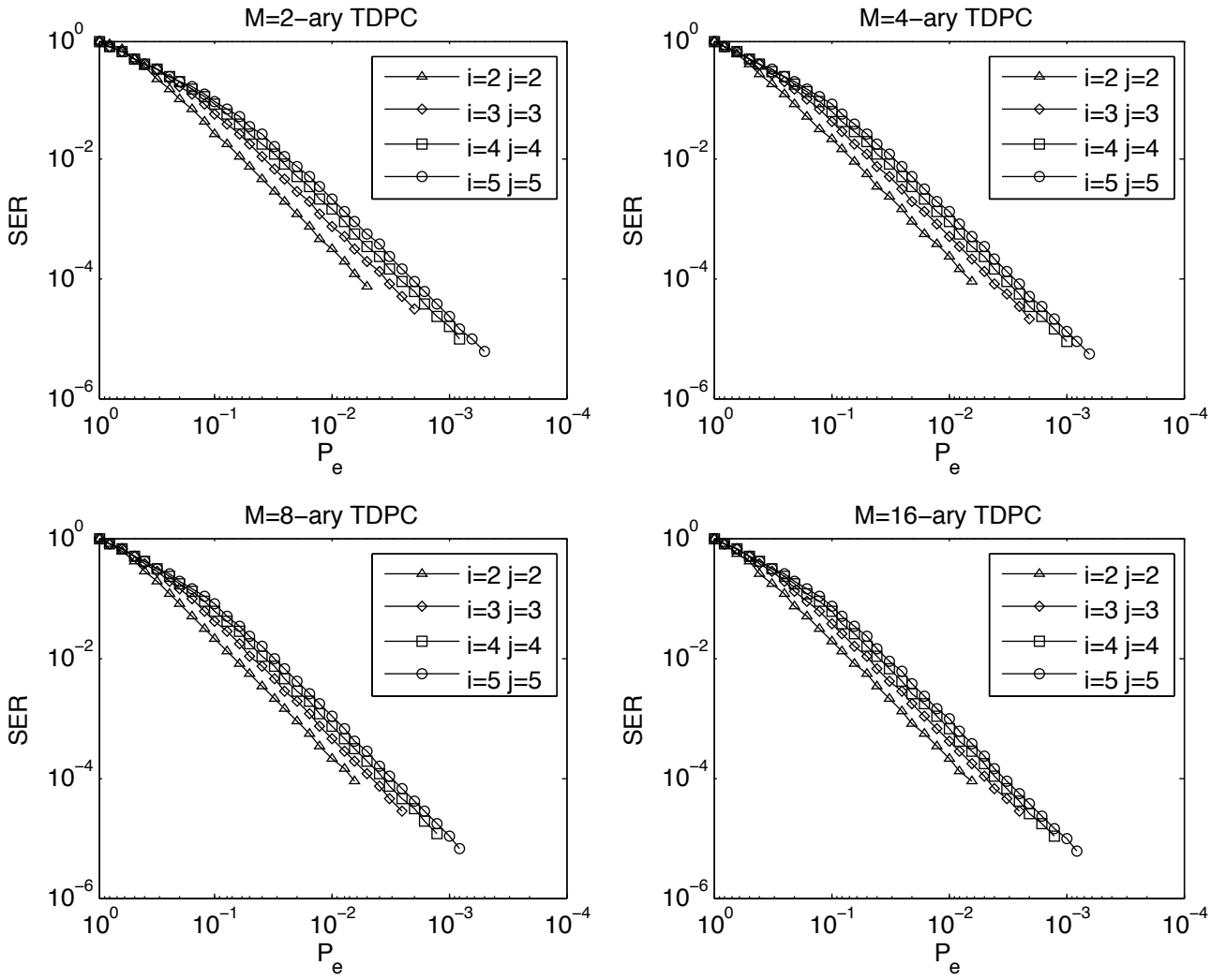

Fig. 1. SER plots for TDPC codes having various combinations of radix $M \in \{2, 4, 8, 16\}$, rows $i \in \{2, 3, 4, 5\}$ and columns $j \in \{2, 3, 4, 5\}$, when communicating over an M-ary symmetric channel.