A MATLAB Exercise Book

Ludmila I. Kuncheva and Cameron C. Gray

ISBN 978-0-244-25328-8
Second Edition

Preface to the second edition

This is still a book containing exercise problems in MATLAB. The collection of problems covers basic topics and is meant to stimulate student's creativity in designing and implementing algorithms.

The respective elements of the language are briefly covered before the exercise section of each chapter.

In this edition:

- We have revised the problem selection in view of some changes in the new MATLAB releases.
- Solutions are provided for all even-numbered problems.
- We realised that the cost of the book does not make it suitable as an exercise notebook, so we removed the spaces left for notes.

The reader should be aware that there are many ways to solve a problem, and the solutions that we offer in this book are not necessarily the shortest or the most time-efficient ones. Some solutions are chosen for their readability. Like most programming languages, MATLAB is developing from version to version, and some of the commands explained and used here may alter in syntax or functionality in the future.

The book could be useful for MATLAB course instructors as a set of ideas and examples to draw upon when creating their own collections of problems.

Ludmila Kuncheva and Cameron Gray

Bangor, January 14, 2020

Preface to the first edition

The book is meant to be used for exercise by the students taking module 'Algorithm Design with MATLAB' at the School of Computer Science, Bangor University, UK. The module does not go into great details about MATLAB capabilities. Most topics are taught within one or two hour-long lectures. It is difficult to go beyond the basics and into the exciting topics such as image edge detection and segmentation, statistical analyses or intricate graphical user interfaces. Consequently, the exercises at the end of the chapters are meant to stimulate the student's ability to solve problems using the limited subset of the language rather than test their expertise in mastering MATLAB.

Some of the problems assume knowledge of elementary algebra and geometry, or specific algorithms such as bubble sorting, Monte Carlo and evolutionary algorithms. However, we kept the exposition simple and self-contained, so that the book can be useful for a reader with minimal technical or mathematical background.

The problems are of different difficulties. Some can be used in class tests or exams, while others require more time and effort, and are more suitable for coursework. Solutions are provided only for the examples in each chapter. Because the book is intended to be a personal hard-copy, we have left spaces for handwritten answers and notes as shown below.

We enjoyed writing this book and hope that you will enjoy the intellectual workout.

Ludmila Kuncheva and Cameron Gray

Bangor, June 17, 2014

Contents

Chapter 1

Getting Started

1.1 MATLAB

MATLAB® is a software package designed for mathematical and scientific computing. It is also a development environment and a programming language. Its primary specialisation is efficiently handling matrix and vector mathematics.

1.2 Programming Environment

1.2.1 Environment Layout and File Editor

Figure 1.1 shows a version of the default MATLAB programming environment. It consists of four spaces: (1) the MATLAB Command Window with the MATLAB prompt sign », (2) the Workspace displaying the variables in the MATLAB memory, (3) the Current folder box showing the folder's content, and (4) the Command history box showing a list of recent commands.

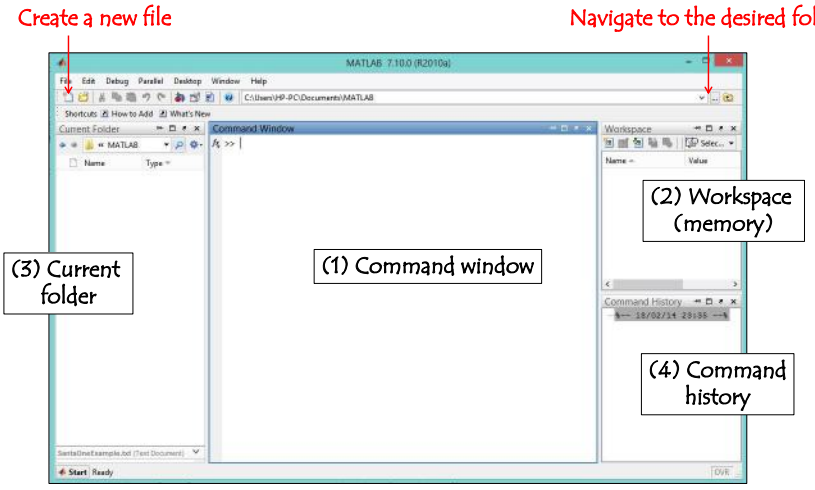


Figure 1.1: Default Layout of the MATLAB Programming Environment.

Although MATLAB can execute commands typed straight in the Command Window, it is best to store the code in a bespoke 'm file' or MATLAB script. A navigation button (top right in Figure 1.1) allows

the user to choose a folder where the work will be stored. Click on the 'New file' icon at the top left corner. The editor window appears as shown in Figure 1.2.[1]

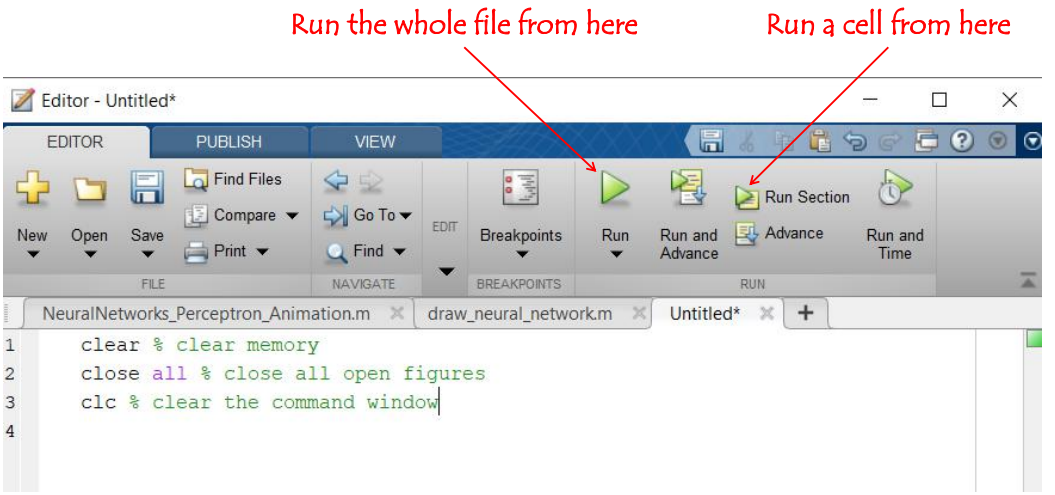


Figure 1.2: The editor window.

1.2.2 Running Your Code

A set of MATLAB commands can be executed by one of the following ways

1. Type the commands directly in the MATLAB Command Window.
2. Highlight and copy the commands from the editor window and paste them in the Command Window.
3. Highlight the commands in the editor window and press function key F9. For Apple/OS X machines, you should use ⇧+F7 (by default you will also need to hold the fn key to avoid the 'special' meaning of F7).
4. Run the file with the commands in the editor window from the run icon, as indicated in Figure 1.2. At the first run, MATLAB will ask you to save the file if you have not done this already. All the commands in the editor will be carried out in the order of appearance.
5. Save the file and run it by typing its name at the prompt in the Command Window. Make sure that the file is in the current folder or there is a path to the folder where the file is saved. The file names and variable names in MATLAB must not start with a number, and must not contain special symbols except for the underscore symbol. For example, your file can be named `lab4_Q2.m`. Then by typing `lab4_Q2` at the » prompt, MATLAB will run the code within.

There is yet another way to execute a part of your code. A section in the code can be executed on its own from the icon indicated in Figure 1.2. A section is delineated by a double percentage symbol

[1] This may differ in newer releases of MATLAB.

followed by a space, '%% '. By placing the cursor within the section and pressing the Run-Section icon, MATLAB will run the code from the beginning of the section to the next section or the end of the code, whichever it encounters first.

If you want to **stop** your code running, press Control-C in the Command Window.

1.2.3 Getting Help

MATLAB has an extensive on-line help system. In addition to the Help item in the menu, MATLAB offers the `help` command. This command requires a single parameter, the name of the command you wish to get help with. For example, `help sin` prints the help article about the command `sin` directly into the Command Window starting: -

```
SIN    Sine of argument in radians.
SIN(X) is the sine of the elements of X.
```

Further on you can use the various options in the Help menu.

1.2.4 Tips

The 'Mantra': Start your code by clearing the MATLAB memory (the workspace) using `clear`, closing previously opened figures using `close all`, and clearing the Command Window using `clc`. The three lines are shown in Figure 1.2 and reproduced here for ease of reference.

```
clear                   % Clear MATLAB Workspace Memory
close all               % Close all Figures and Drawings
clc                     % Clear MATLAB Command Window History
```

Storing the Content of the Command Window: MATLAB command `diary <file_name.txt>` dumps the content of MATLAB Command Window in the file with the specified name, in ASCII text format. All subsequently typed commands and MATLAB answers will be stored there. To end the recording of the MATLAB window dialogue, type `diary off` at the Command Window prompt.

Error Messages: MATLAB displays errors in the Command Window, in red. Always read the error message. It is often a spot-on indication of what is wrong.

1.2.5 Good Programming Style and Design Practices

1. Strive to create readable source code through the use of blank lines, comments and spacing. It is important to put short and meaningful comments. This will help you read your code at a later date.

2. Use consistent naming conventions for variables, constants, functions and script files.

3. Use consistent indentation as provided by the MATLAB editor window. (Highlighting the text with the cursor and choosing Text » Smart Indent will reflow your script automatically, or use the shortcut ctrl + I on Windows / ⌘ + I for OS X.
4. Split your code into readable pieces. Use functions and separate script files where necessary.
5. Limit the creation of unnecessary variables in your code. Aim at minimum script length with maximum simplicity and clarity.
6. Where suitable, use variables instead of hard-coded values as loop limits and array sizes. This will make your code re-usable.
7. When you feel that things are getting out of control, start over.

1.3 MATLAB as a Calculator

Table 1.1 shows the syntax of some frequently used MATLAB mathematical operations.

Table 1.1: MATLAB operations

Operation	Symbol	Example	Maths	Output
Addition	+	`4 + 7`	$4+7$	`11`
Subtraction	−	`12.3 - 5`	$12.3-5$	`7.3000`
Multiplication	*	`0.45 * 972.503`	0.45×972.503	`437.6264`
Division	/	`5 / 98.07`	$\frac{5}{98.07}$	`0.0510`
Power	^	`4^7.1`	$4^{7.1}$	`1.8820e+004`
Square Root	`sqrt()`	`sqrt(15)`	$\sqrt{15}$	`3.8730`
Logarithm*	`log()`	`log(0.67)`	$\ln(0.67)$	`-0.4005`
Exponent	`exp()`	`exp(-2.1)`	$\exp(-2.1) = e^{-2.1}$	`0.1225`
Sine**	`sin()`	`sin(0.8)`	$\sin(0.8)$	`0.7174`
Cosine**	`cos()`	`cos(-2)`	$\cos(-2)$	`-0.4161`

* natural logarithm

** the arguments for all trigonometric functions are in radians

You can type numerical expressions directly at the Command Window prompt and receive the answer after pressing Enter. For example, type the expression below at the Command Window prompt followed by Enter. The answer will be shown in the Command Window, as well as stored in a variable `ans`. (*Note*: `ans` is replaced by the result of each expression that is not already assigned to a variable.)

```
>> (941 - 5.9)/(41 - sqrt(19))
ans =
   25.5205
```

The default display precision of MATLAB is 4 decimal places but the numbers are stored in memory with a much greater precision (double precision, 64-bits/17 significant figures).

The following list of operations can be used to convert real numbers into integers:

`round(a)` Rounds a using the standard rounding rules.
`ceil(a)` Returns the nearest integer greater than or equal to a.
`floor(a)` Returns the nearest integer smaller than or equal to a.

For example, round(3.7) = ceil(3.7) = 4, and floor(3.7) is 3. If a is an integer, then ceil(a) = floor(a) = a.

MATLAB operations may be applied to matrices as well. All operations which are done element-by-element, for example addition, subtraction and multiplication by a number, have the same syntax for both scalars and matrices. The same holds for the trigonometric functions, the logarithm and the exponent. For the multiplication division and power, the matrix operation may be interpreted in two ways. Hadamard product denotes an operation where the matrices are of the same size, and the entries of the resultant matrix are the pairwise products of the elements of the two matrices. MATLAB uses $*$ for 'proper' matrix multiplication, and $.*$ for the Hadamard product. The same holds for division and powers. Element-wise operations are preceded by a dot, for example, '.*', './' and '.^'.

1.4 Exercises

1. Create a standard template for your lab scripts. Put in the heading your name, username and the module title. Save the file with name

   `labXX_<NameSurname>_<ddmmyy>.m`

   inserting your name and today's date. When you run it, the code should clear the memory and the Command Window, and should close any currently open figures.

2. Using only the MATLAB Command Window, find out what the command `imagesc` does.

3. Demonstrate by using several values of angle θ that:

$$\sin^2(\theta) + \cos^2(\theta) = 1 \ .$$

4. Use one MATLAB line to evaluate the expression below:

$$\sqrt{\frac{(4.172 + 9.131844)^3 - 18}{-3.5 + (11.2 - 4.6) * (7 - 2.91683)^{-0.4}}}$$

5. The short-cut calculation for the binomial coefficient $\binom{n}{k}$ is:

$$\binom{n}{k} = \frac{\overbrace{n \times (n-1) \times (n-2) \times \ldots (n-k+1)}^{k \text{ terms}}}{k \times (k-1) \times (k-2) \times \ldots \times 2 \times 1}$$

   Using the short-cut calculation, evaluate $\binom{13}{4}\binom{10}{5}$ and verify your answer using MATLAB command `nchoosek`.

6. Verify that the exponent (`exp()`) and natural logarithm (`log()`) are inverses of one another (cancel one another).

7. Without using the square root (`sqrt()`) or the power command (`^`) find the square root of 555 with precision 4dp. To back your solution, copy the Command Window dialogue. You should save both your typed commands and the MATLAB responses.

8. Use MATLAB as a calculator to find the root of the equation:

$$0.5(x-2)^3 - 40\sin(x) = 0$$

   within the interval $[2, 4]$. You are not allowed to plot the function and gauge the answer from the graph. Give the solution with precision 2dp. To document your solution, copy the Command Window dialogue. You should save both your typed commands and the MATLAB responses.

Chapter 2

MATLAB: The Matrix Laboratory

MATLAB was created as a scientific tool to make matrix algorithms more efficient and easier to program. Therefore, almost every operation is optimised to work with matrices.

2.1 Variables and Constants

2.1.1 Value Assignment

Variables and constants (scalars) in MATLAB do not have to be declared at the beginning of the code as in other languages. Values can be assigned to them in a straightforward assignment operation, for example,

```
my_first_MATLAB_variable = -1.23456789;
m = 12;
string_example = 'My first MATLAB string';
raining_tomorrow = true;
```

MATLAB will create four variables and store the values in memory. The variables can be seen in the Workspace window of the MATLAB environment.

The semicolon at the end of the assignment operation suppresses the output in the Command Window but has no effect on the assignment itself. You can display the value of any existing variable by typing its name at the MATLAB prompt in the Command Window. For example, typing `m` will return 12, and typing `raining_tomorrow` will return 1. MATLAB stores true values as 1 but will accept any non-zero value as 'true'; a 0 value is used for 'false'.

A neater way to display a string or the content of a variable in MATLAB Command Window is the command `disp`. This command takes only one argument, which evaluates to a number or a string. For example,

```
>> x = 9;
>> disp('The value of x is:')
>> disp(x)
```

MATLAB has several predefined constants that can be used in place of variables or values in expressions. The following is a small selection of some of the most useful ones:

pi	π
i	Imaginary number $\sqrt{-1}$.
eps	The smallest number, ϵ, determined by machine precision.
Inf	The largest number, ∞.
NaN	Not a Number. Undefined numerical result.

2.1.2 Names and Conventions

A valid variable name starts with a letter, followed by letters, digits, or underscores. Note that MATLAB is case sensitive.

When choosing a variable name, it is best to avoid words which could be MATLAB commands or reserved words such as: if, end, for, try, error, image, case, plot, all, and so on. If you do this, you will (temporarily) erase the MATLAB command of the same name which you may need later in your code.

Interestingly, variables which are used for loop indices are often chosen to be i, j or k. This convention is probably a FORTRAN legacy where variables holding integer values must have names starting with one of the letters I, J, K, L, M or N. FORTRAN variables starting with any other letter are understood to be 'real' or floating point numbers. There is no such convention in MATLAB.

2.2 Matrices

2.2.1 Creating and Indexing

Let us start with an example of a two-dimensional matrix (array) A with $m = 3$ rows and $n = 2$ columns. The matrix can be entered into MATLAB memory (working space) as follows: -

```
>> A = [6 3; 5 2; 4 1];
A =
     6     3
     5     2
     4     1
```

The semicolon serves as 'carriage-return', separating the rows of A from one another.

The size of a matrix A can be found using `size(A)`. For a scalar, the `size` command will return an array with two values: 1 (number of rows) and 1 (number of columns). For matrix A, the MATLAB answer will be 3 and 2.

```
>> size(A)
ans =
```

```
3     2
```

Vectors are matrices where either $m = 1$, $n > 1$ (vector-row) or $m > 1$, $n = 1$ (vector-column). Matrices may have third, fourth dimension, and so on. For example, images can be represented as three panels for the red, green and blue colours, respectively. Each panel is a matrix of m rows and n columns of pixels. Hence an image is a 3-dimensional matrix[1] of size $m \times n \times 3$. Matrices can be indexed (or subscripted in MATLAB terms) just like in other programming languages, except that in MATLAB, indices begin at 1. Figure 2.1 shows a parallel between array access in Java and in MATLAB.

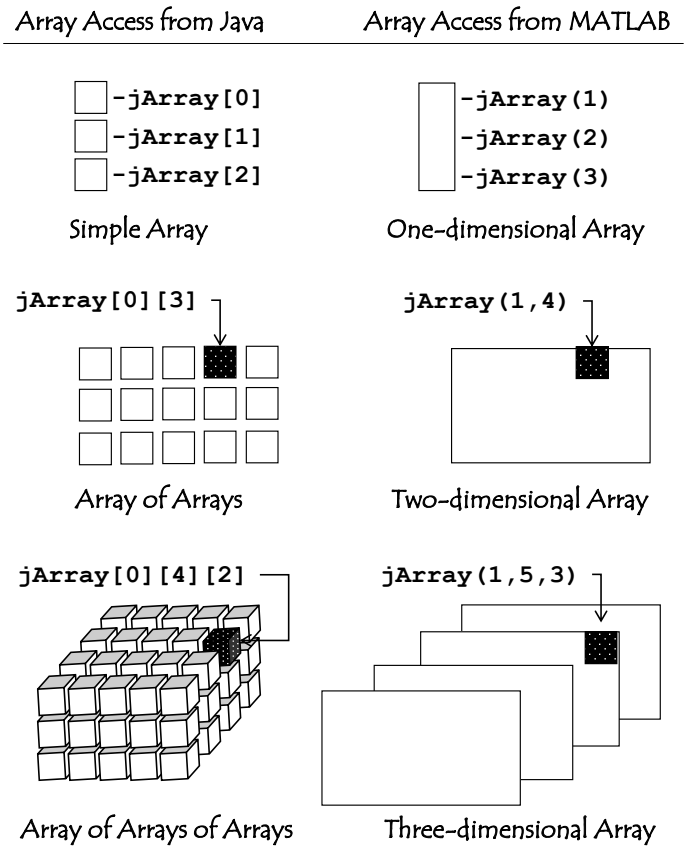


Figure 2.1: Illustration of MATLAB Matrix Dimensions. (Recreated from the MathWorks' image.)

2.2.2 Accessing Matrix Elements

There are several ways to access individual elements of an array. Consider matrix A from the example above. Each element can be accessed using its row and column indexes. For example, the element in row 3 column 1 holds the value 4. The line below stores this element in variable `ele`.

```
>> ele = A(3,1)
ele =
```

[1] Described in more detail later, in Section 8.1.2.

```
     4
```

Alternatively, two-dimensional matrices can be indexed with only one index which goes from 1 to $m \times n$. Consider a vector-column constructed by putting the consecutive columns of the matrix underneath the previous column. For example, arranging A this way will result in, the vector-column $[6, 5, 4, 3, 2, 1]^T$. Therefore, `A(5)` would return the value 2.

One of the main assets of MATLAB is that a set of elements in a matrix can be addressed simultaneously. For example:-

```
>> A([1 4 5])
ans =
     6     3     2
```

Elements in a matrix can be addressed through logical indexing. We'll come back to this method in Section 3.1. Consider here the following example involving matrix A above. First, create a matrix L of the same size as A, containing logical values. Addressing A with L, as `A(L)` will extract only the elements of A where L contains a `true` value.

```
>> L = [true false;true true;false false]
L =
     1     0
     1     1
     0     0

>> A(L)'
ans =
     6
     5
     2
```

A value can be assigned to an element of a matrix by the simple assignment operator, for example;

```
>> A(3,1) = -9
A =
     6     3
     5     2
    -9     1
```

MATLAB will also allow assignment of multiple values in one operation. For example, to replace the elements addressed through L by value 25, we can use:

```
>> A(L) = 25
A =
    25     3
    25    25
    -9     1
```

Better still, you can use three different values to replace the addressed elements. For example:-

```
>> A(L) = [55,111,222]
A =
    55     3
   111   222
    -9     1
```

In addition, you can extract the desired elements in any order and with any number of copies. For example, take two copies of element #6 followed by three copies of element #1.

```
>> A ([6 6 1 1 1])
ans =
    1   1  55  55  55
```

2.2.3 Visualising a Matrix

A matrix can be visualised in MATLAB by transforming it into an image. Each element of the matrix becomes a pixel. Elements of the same value will have the same colour. Try the following code:

```
A = [1 2 3 4;5 6 7 8;9 10 11 12]; % creates matrix A
figure                            % opens a new figure window
imagesc(A)                        % transforms and shows the matrix as an image
axis equal off                    % equalises and removes the axes from the plot
```

See Chapter 8 for more detailed information, examples and exercises dealing with images and visualising matrices.

2.2.4 Concatenating and Resizing Matrices

If the dimensions agree, matrices can be concatenated using square brackets. Figure 2.2 shows an example.

```
A = [1 2;3 4;5 6] % matrix 3-by-2
B = [7 8;9 10] % matrix 2-by-2
C = [A;B] % concatenated, 5-by-2
D = [A [B;0 0]] % concatenated, 3-by-4
```

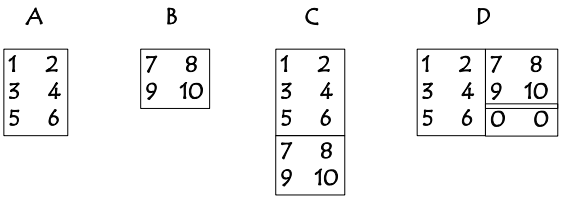


Figure 2.2: An example of matrix concatenation.

A matrix can be used as a 'tile' to form a repeated pattern using the `repmat` command. For example, let A be a 2×3 matrix. The code below uses A as a tile and repeats it in 3 rows and 2 columns.

```
>> A = [0 1 2;-2 -1 0] % 2-by-3 matrix
A =
     0     1     2
    -2    -1     0

>> B = repmat(A,3,2)
B =
     0     1     2     0     1     2
    -2    -1     0    -2    -1     0
     0     1     2     0     1     2
    -2    -1     0    -2    -1     0
     0     1     2     0     1     2
    -2    -1     0    -2    -1     0
```

A matrix can be reshaped using the `reshape` command. The new matrix must have exactly the same number of elements. The way this command works is illustrated in Figure 2.3.

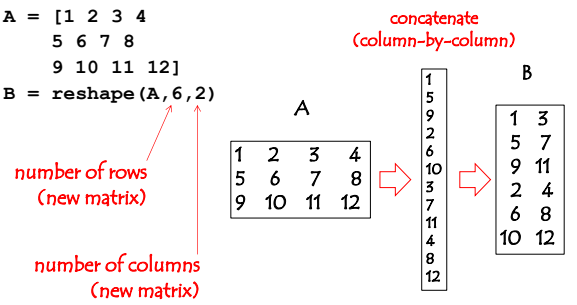


Figure 2.3: An example of matrix reshape.

Notice that the input matrix is first concatenated into a vector-column, taking consecutive columns and placing them underneath the last. The output matrix is constructed column by column, taking consecutive values from the vector-column.

2.2.5 Matrix Gallery

MATLAB offers a wealth of predefined matrices. A useful subset of these is shown below:

`zeros(3,2)`

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

`ones(3)`

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

`eye(3)`

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

`rand(3,4)`

$$\begin{bmatrix} 0.9649 & 0.9572 & 0.1419 & 0.7922 \\ 0.1576 & 0.4854 & 0.4218 & 0.9595 \\ 0.9706 & 0.8003 & 0.9157 & 0.6557 \end{bmatrix}$$

If a square matrix is required, one input argument will suffice, for example, `ones(3)`. This is both the number of rows and the number of columns. The identity matrix is square by definition, hence `eye()`

only takes one input argument. Otherwise, the matrix is created with the specified number of rows and columns. These matrices can be generated in more than two dimensions.

In the matrix with random numbers, all values are drawn independently from a uniform random distribution in the interval [0,1].

2.3 The Colon Operator

The primary use of the colon operator is to create ranges. The basic form of a range is `start:increment:end`. The increment parameter is optional, and by default is 1. For example, to create a vector-row *x* with all integers from 1 to 10, you can use:

```
x = 1:10
x =
     1     2     3     4     5     6     7     8     9    10
```

The three components of the colon operator do not have to be integers, nor must they be positive. For example;

```
>> y = -1.7:0.81:2.452
y =
   -1.7000   -0.8900   -0.0800    0.7300    1.5400    2.3500
```

The vectors generated by the colon operators are row-vectors. The vector contains as many elements as necessary to increase the start value by the increment value to reach the end value without exceeding it. For example:-

```
>> 1:10:55
ans =
     1    11    21    31    41    51
```

Matrices can be addressed with vectors created by the colon operator. The vector can be embedded directly as the index;

```
>> x = 2:4:20;
>> Y = [x;2*x]
Y =
     2     6    10    14    18
     4    12    20    28    36

>> Y(1,2:4)
ans =
     6    10    14
```

In this example, the addressed values are in row 1, and columns 2, 3 and 4.

One particularly useful form of the colon operator is for defining the whole range within a matrix dimension. For example, `Y(:,2)` will return all elements in column 2 of *Y*. In this case, ':' means 'all' (rows):

```
>> Y(:,2)
ans =
     6
    12
```

The colon operator on its own, `A(:)`, reconfigures matrix A into a column vector by placing each column below the last. For example;

```
A =
     9     6     3
     8     5     2
     7     4     1

>> A(:)
ans =
     9
     8
     7
     6
     5
     4
     3
     2
     1
```

2.4 Linear spaces and mesh grid

Instead of creating a range through start, offset and end, sometimes it is easier to specify the start and the end values, and the number of elements of the desired vector. For example, to create a vector with 7 uniformly spaced elements between 0 and 1, you can use:-

```
>> x = linspace(0,1,7)
x =
         0    0.1667    0.3333    0.5000    0.6667    0.8333    1.0000
```

This command ensures that `x(1) = 0` and `x(7) = 1`.

Sometimes we need to generate all (x, y) coordinates of the points on a grid. Suppose that the grid spans the interval from 2 to 12 on x and has 4 points, and the interval from -1 and 6, and has 5 points on y. The `meshgrid` command can be used for generating simultaneously the x and the y coordinates:

```
>> [x,y] = meshgrid(linspace(2,12,4), linspace(-1,6,5))
x =
    2.0000    5.3333    8.6667   12.0000
    2.0000    5.3333    8.6667   12.0000
    2.0000    5.3333    8.6667   12.0000
    2.0000    5.3333    8.6667   12.0000
    2.0000    5.3333    8.6667   12.0000

y =
   -1.0000   -1.0000   -1.0000   -1.0000
    0.7500    0.7500    0.7500    0.7500
    2.5000    2.5000    2.5000    2.5000
    4.2500    4.2500    4.2500    4.2500
    6.0000    6.0000    6.0000    6.0000
```

Try the following example with the `meshgrid` and `linspace` commands.[2]

```
clear all; close all; clc
[x,y] = meshgrid(linspace(2,12,4), linspace(-1,6,5));
figure, hold on, grid on % open and hold a figure with a grid
surf(x,y,x.*y) % plot the surface of function x*y
surf(x,y,zeros(size(x))) % plot the plane of the zero-"ground"
stem3(x,y,x.*y,'k*-') % plot stems at all grid points from ground to surface
rotate3d % allow for rotation of the figure with the mouse
```

Figure 2.4 shows the MATLAB output for the above piece of code. Using the mouse, you can rotate the plot in the MATLAB figure.

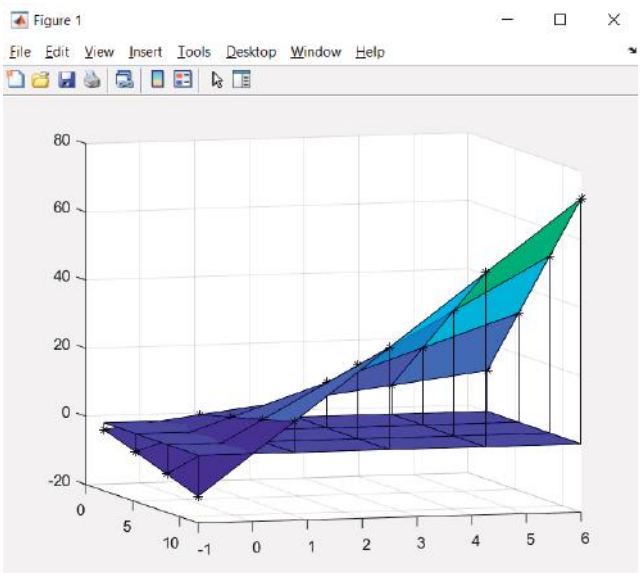


Figure 2.4: Example of using 'meshgrid'.

[2]Plotting will be detailed later in Chapter 5.

2.5 Operations with matrices

Table 2.1 shows some often used MATLAB operations on matrices.

Table 2.1: MATLAB operations for matrices

Operation	Symbol	Example	Output
Addition/Subtraction	+/−	`A +/- B`	Sum/Difference of the two matrices
Addition of a scalar	+/−	`A - 9`	Subtracts 9 from each element of A
Multiplication by a scalar	∗	`4*A`	Multiplies every element of A by 4
Matrix multiplication	∗	`A*B`	Matrix multiplication
Hadamard product	.∗	`A.*B`	Element-wise multiplication of A and B

The following numerical operations are carried out on every element of the matrix: power (`.^`), square root, logarithm, exponent, sine, cosine.

Table 2.2 contains a list of MATLAB functions which operate on matrices.

Table 2.2: MATLAB functions for matrices

Command	Return
a'	Transpose of a
find(a)	Indices of all non-zero elements in a.
fliplr(a)	Matrix a, flipped horizontally
flipud(a)	Matrix a, flipped vertically.
inv(a)	Inverse of a
min(a)	Minimum-valued element of a. †
max(a)	Maximum-valued element of a. †
numel(a)	The number of elements of a.
repmat(a,m,n)	A matrix where matrix a is repeated in m rows and n columns
reshape(a,m,n)	Matrix a reshaped into m rows and n columns.
size(a)	The size of a (#rows, #columns, ...)
sort(a)	Vector a sorted into ascending order. †
sum(a)	Sum of elements of a. †
unique(a)	The list of unique elements of a in ascending order.

† For a matrix, the operation will be carried out separately on each column. For a vector (row or column), the operation will be carried out on the vector.

In addition to the minimum or maximum values returned by the respective commands, MATLAB returns the exact index where this value has been encountered. For example,

```
a = [3 1 9 5 2 1];
m = max(a)
```

returns 9 in `m`. If however, the command is invoked with two output arguments:

```
[m,i] = max(a)
```

MATLAB will return 9 in `m` and 3 in `i` because `a(3)` holds the largest value 9. Recent versions of MATLAB (after 7.9, 2009b) allow for replacing unnecessary arguments with a tilde (~). If only the place of the maximum is of interest, you can use:

```
[~,i] = max(a)
```

If a minimum or a maximum value appears more than once in the array, only the index of the first occurrence will be returned. In the example above, the minimum value 1 sits in positions 2 and 6. Only 2 will be returned as the second output argument of the `min` command.

Similarly, the `sort` command returns as the second argument a permutation of the indices corresponding to the sorted vector. For example,

```
>> [s,p] = sort(a)
s =
     1     1     2     3     5     9

p =
     2     6     5     1     4     3
```

In this example, `s` contains the sorted `a`. `p(1)` is 2 because the first occurrence of the minimum element 1 is at position 2. The next smallest element is the 1 at position 6, hence the second entry in `p`. The third smallest element, 2, is in position 5, and so on.

2.6 Cell Arrays

A cell array is a data structure which can contain any type of data. For example, the code below creates a cell array `C` of size 2×2 which contains strings, arrays and numerical values as its elements.

```
>> C = {1,'Joey';zeros(3),[false true;true true]}
C =
    [         1]    'Joey'
    [3x3 double]    [2x2 logical]
```

The elements can be accessed using parentheses as with numerical arrays.

```
>> C(2,1)
ans =
    [3x3 double]
```

The element is returned as a cell. If you want to access the *content* of the cell, use braces { } instead of parentheses ( ):

```
>> C{2,1}
ans =
     0     0     0
     0     0     0
     0     0     0
```

2.7 Exercises

1. Create a 4×1 column vector, that contains any values of your choosing.
2. Create a cell array of 8 elements such that element i contains the identity matrix of size i, $i = 1, \ldots, 8$.
3. Use *one* MATLAB command to evaluate the sine of 30^o, 45^o, 60^o, and 120^o. Subsequently, evaluate cosine, tangent and cotangent of the same angles.
4. Find the sum of the integers from 1 to 100.
5. Create an example to demonstrate that matrix multiplication is not commutative.
6. Create an example to demonstrate that the following equation holds;

$$(ABC)^T = C^T B^T A^T \ ,$$

   where A, B and C are matrices of different sizes, and the product ABC is feasible.

7. Evaluate the following expression:

$$\left(6 \begin{bmatrix} 10 & -7 & 6 & -9 \\ 0 & -1 & 10 & 7 \\ 7 & 9 & 4 & 9 \end{bmatrix} - 8 \begin{bmatrix} 4 & -2 & 5 & -9 \\ 6 & 4 & -9 & -8 \\ 5 & -6 & -4 & 7 \end{bmatrix} \right) \times \begin{bmatrix} 5 & 4 & -7 & -3 \\ 6 & 4 & 0 & 2 \\ -4 & -6 & 10 & -5 \end{bmatrix}^T$$

8. Create a 1×6 vector v containing the integer values from 20 to 25. Subsequently, create an 1×6 vector whose values are equal to 5 times the values in v.
9. Create a vector that goes at equal steps from -2 to $+2$ containing 50 components.
10. Create a vector spanning the range from 0 to 2π, containing 100 equally spaced components, so that the first value is 0, and the last value is 2π.
11. Input vector $q = [-1, 5, 3, 22, 9, 1]^T$ in the MATLAB memory.
    (a) Rescale q into a unit vector (magnitude 1) using only the matrix operations shown in Tables 2.1 and 2.2.
    (b) Rescale q linearly, so that the minimum is 0 and the maximum is 1 - again using only the matrix operations presented in this chapter.

(c) Rescale q linearly, so that the minimum is -3.6 and the maximum is 105 - using the same functions/operations.

12. Create a matrix of 100 rows and 100 columns. The odd columns should contain values 2, and the even columns, values 0.

13. Create the following matrix using one MATLAB line of code and the `reshape` command.

```
 1     2     3     4     5     6     7     8     9    10
11    12    13    14    15    16    17    18    19    20
21    22    23    24    25    26    27    28    29    30
31    32    33    34    35    36    37    38    39    40
41    42    43    44    45    46    47    48    49    50
```

14. Create and visualise in a figure the following matrix

```
M =

     8     8     8     8     8     8     8     8     8     8
     8     8     8     8     8     8     8     8     8     8
     8     8     0     0     0     0     0     0     8     8
     8     8     0     0     0     0     0     0     8     8
     8     8     0     0     3     3     0     0     8     8
     8     8     0     0     3     3     0     0     8     8
     8     8     0     0     0     0     0     0     8     8
     8     8     0     0     0     0     0     0     8     8
     8     8     8     8     8     8     8     8     8     8
     8     8     8     8     8     8     8     8     8     8
```

15. Write MATLAB code to construct matrix B whose image is shown in Figure 2.5. Subsequently, reproduce the two plots in the figure.

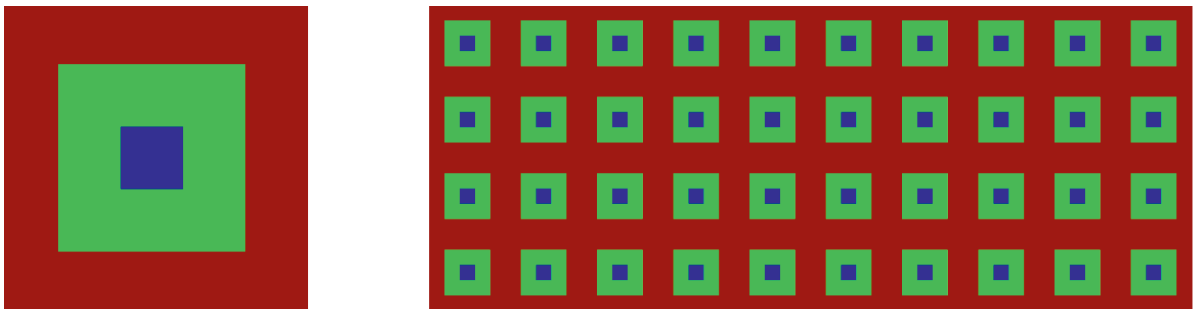


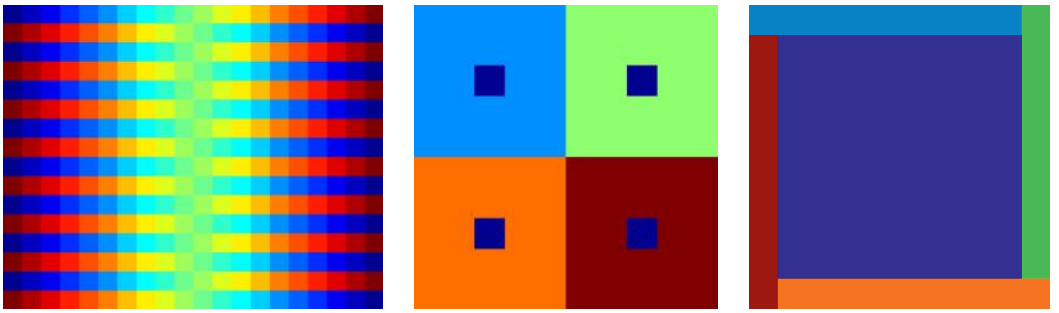
Figure 2.5: Two matrices displayed as images.

16. Write MATLAB code to construct the matrices described below and visualise them using `imagesc`. Do not use loops!

(a) The first matrix should have 16 rows with consecutive integers from 1 to N, where N is given by the user when prompted. The rows should be alternating: the numbers should be in increasing order in row 1, decreasing order in row 2, increasing in row 3, decreasing in row 4 and so on. An example for $N = 20$ is shown in Figure 2.6 (a).

(b) The second matrix should be of size 10 by 10. The exact colour of the blocks does not matter as long as all 4 colours are different and the colour of the central blocks are the same. An example is shown in Figure 2.6 (b).

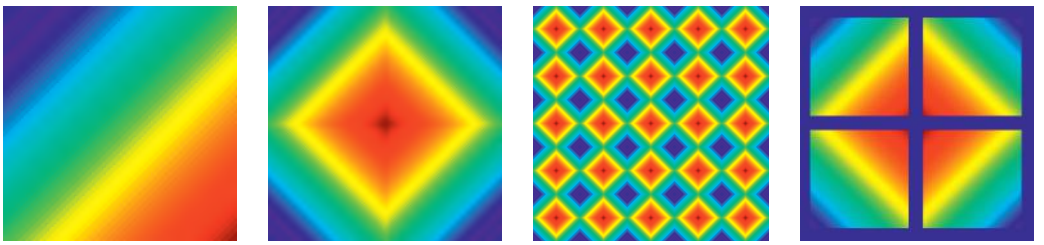
(c) The third matrix should appear as a colour frame of size M (M rows and M columns), where M is given by the user, when prompted. An example for $M = 10$ is shown in Figure 2.6 (c).



(a) (b) (c)

Figure 2.6: Visual representation of three matrices.

17. Knowing that for any square matrix A, $B = A + A^T$ is a symmetric matrix; reproduce the four plots in Figure 2.7. Save the four matrices in a cell array.



(a) (b) (c) (d)

Figure 2.7: Symmetric matrices.

18. Using a matrix equation, find the intersection point of the lines defined by the following equations:

$$\begin{aligned} 7x - 12y + 4 &= 0 \\ 12x - 45y + 26 &= 0 \end{aligned}$$

Note: Command `inv(A)` will return the inverse of matrix A.

19. Run the code below. It will take an image available from the standard MATLAB installation, convert it to grey scale, store the matrix in variable `im`, and show the image as in Figure 2.8 (a).

```
im = rgb2gray(imread('pepper.png')); % read the image into a 2d matrix
imshow(im) % show the grey-level image
```

    (a) Find out the size of the matrix containing the image, and cut (approximately) the part that contains the onion. Use the `imshow` command to display the result as in Figure 2.8 (b). You must not use any image processing methods and commands such as 'crop'.

    If you are unsure whether a command is from the Image Processing Toolkit, type `which <command>` in the Command Window. The result will show the path to the m-file where that command is defined. If the path contains 'toolbox/images' the command is excluded.

(a) Original (b) Onion (c) Framed onion

Figure 2.8: Onion cut-out and framed.

    (b) Add a frame of k rows and k columns of zero values around the onion image and display it, as shown in Figure 2.8 (c). The value of k should be changeable; in the example, $k = 30$. Note: When displaying your new matrix, say z, use:

```
imshow(uint8(z)) % show the grey-level image
```

    (c) Find out how many different grey level intensities (of the possible 256 intensities) appear in your onion image. Compare this with the number of intensities in the original image and give a short comment.

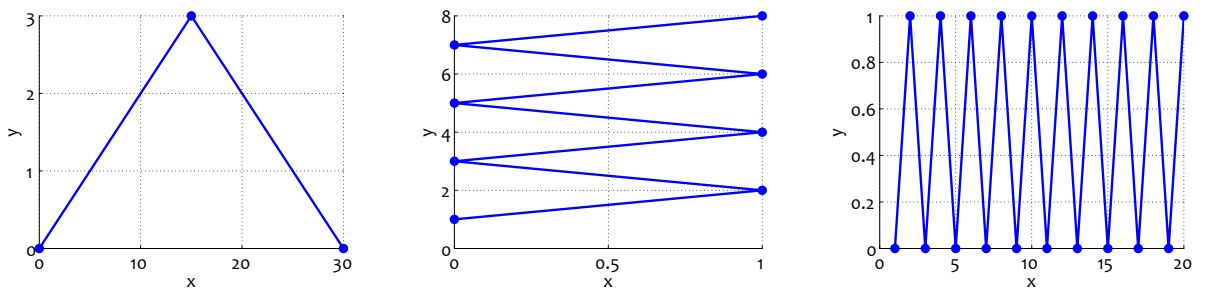
20. Create a matrix A of size $m \times n$, whose elements $a(i, j)$ are calculated from the row and column indices as follows:

$$a(i, j) = (j - 4)^2(i + 1)^{-3} + ij \ .$$

    The parameters m and n should be changeable. (You are not allowed to use loops here. Recall the command 'meshgrid'.)

21. Create vectors x and y, which, when plotted and joined, will show the pattern in Figure 2.9 (a)-(c).

22. Create vectors x and y, which, when plotted and joined, will show the pattern in Figure 2.9 (d)-(f).



(a) (b) (c)

(d) (e) (f)

Figure 2.9: Vector patterns.

Chapter 3

Logical Expressions and Loops

3.1 Logical Expressions

3.1.1 Representation

A logical expression is one that evaluates to either true or false. For example, $v > 0$ is a logical expression that will be true if the variable v is greater than zero and false otherwise. Logical expressions can be assigned to Boolean variables. For example, $s = v > 0$ stores the value of the logical expression $v > 0$ in a Boolean variable s. Some programming languages use special data types, as shown in Table 3.1. These are generally referred to as Boolean data types. Other languages, such as MATLAB, allow general data types to represent logical answers.

Table 3.1: Handling logical values in some programming languages

Language	Data Type	Bytes	Values
MATLAB	single, double or logical	4 or 8	0 or 1
Java	boolean	1	true or false
Basic	Boolean	2	True or False
C++	bool	1	True or False

3.1.2 Type and order of operations

Logical expressions may contain numerical, logical and relational operations. Numerical operations involve numbers and their result is a number. Relational operators compare two numbers and their result is true or false. Finally, logical operations connect two logical variables. The result is again, true or false. Table 3.2 shows the most often used relational and logical operations as well as their MATLAB syntax.

Table 3.2: Relational and logical operations in MATLAB

= 'True for all corresponding elements of a and b ...'

Relational operations	
`a == b`	# which are equal to one another.
`a ~= b`	# which are not equal to one another.
`a <= b`	# where $a(i)$ is less than or equal to $b(i)$.
`a < b`	# where $a(i)$ is strictly less than $b(i)$.
`a >= b`	# where $a(i)$ is greater than or equal to $b(i)$.
`a > b`	# where $a(i)$ is strictly greater than $b(i)$.

Logical operations	
`a & b`	# which are both true (non-zero).
`a && b`	Valid only for scalars. True if both are true.
`a | b`	# where either one or both of $a(i)$ and $b(i)$ are true (non-zero)
`a || b`	Valid only for scalars. True if any or both are true.
`xor(a,b)`	# where one of $a(i)$, $b(i)$ is true and the other is false.
`~a`	True if a is false. Also known as 'not' a.

For example,

```
>> a = [ 0 1 2 2 3 1 0 0 1 0];
>> b = [-2 5 0 0 4 1 0 6 3 0];
>> a&b
     0    1    0    0    1    1    0    0    1    0
```

and

```
>> xor(a,b)
     1    0    1    1    0    0    0    1    0    0
```

Notice that logical operators join two Boolean variables while relational operators join two numerical expressions. In a logical expression, numerical operations are carried out first, then the relational operations, and finally the logical operations. The sequence of operations is illustrated in Figure 3.1.

Of course, if parentheses are present, they have precedence over any operation. The operations in the innermost parentheses are carried out first, the operations within the next innermost, second, and so on. If you are not sure about the order of operations, use parentheses.

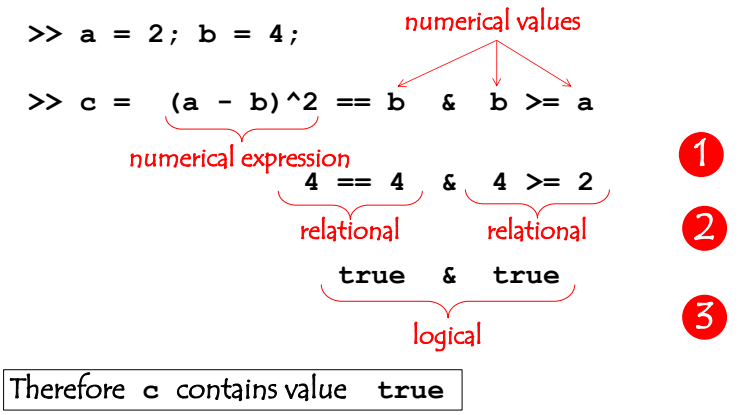


Figure 3.1: Sequence of operations in logical expressions.

Without typing the following lines in MATLAB, try to determine the answers for the three logical expressions:

```
>> a = [4 1; 0 6]; b = [-5 1;0 4];

>> a & b                                % 1
>> a ~= b                               % 2
>> a - b > 3                            % 3
```

3.2 Indexing arrays with logical indices

One of MATLAB's most useful features is the possibility to address an array with a logical index, as explained in Section 2.2.2. For example, suppose that we want to replace all elements of array A which are smaller than 0 with value, say, 22. We can create a logical index of the size of A, containing 1s for all elements that need to be replaced:

```
>> index = A < 0;
```

Next, we can select the relevant elements of A and assign the desired value:

```
>> A(index) =  22;
```

In fact, the whole assignment can be done using just one statement:

```
>> A(A<0) =  22;
```

Any logical expression that evaluates to a true/false matrix of the same size as that of a matrix A, can be used as a logical index into A.

3.3 MATLAB's logical functions and constructs

3.3.1 Logical functions

Table 3.3 contains details of some commonly used logical functions in MATLAB.

Table 3.3: Some logical MATLAB functions.

Command	Return
`all(a)`	True if all elements of *a* are true/non-zero.
`any(a)`	True if any element of *a* is true/non-zero.
`exist(a)`	True if *a* exists in the MATLAB path or workspace as a file, function or a variable.
`isempty(a)`	True if *a* does not contain any elements.
`ismember(a,b)`	True if *b* can be found in *a*.

3.3.2 Conditional operations

Conditional operations act like program switches which respond to certain conditions within the program. The basic `if` and `switch` constructs are shown in Figure 3.2.

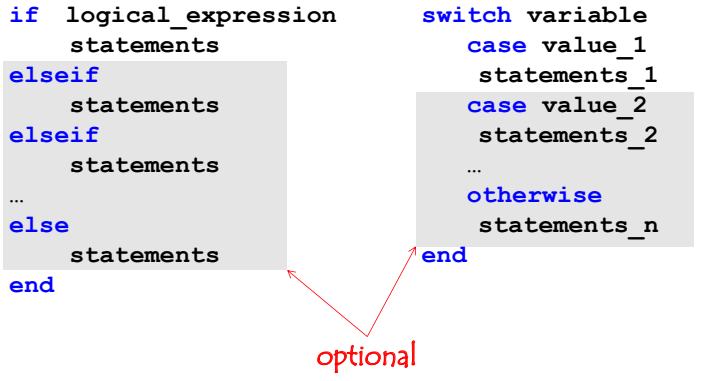


Figure 3.2: Syntax of the 'if' and 'switch/case' operators. The shaded parts are optional.

As an example, consider the following tasks:

Using the MATLAB command `rand` generate a random number between 0 and 1. If the number is greater than 0.5, display the word 'lucky', otherwise display 'unlucky'.

```
if rand > 0.5
    disp('lucky!')
else
    disp('unlucky!')
end
```

Ask the user to input an integer number between 1 and 4, then display the number as a word.

```
n = input('Integer {1,2,3,4} = ');
switch n
    case 1, disp('one')
    case 2, disp('two')
    case 3, disp('three')
    case 4, disp('four')
end
```

3.4 Loops in MATLAB

Unlike other languages, MATLAB only has two types of loop, `for` and `while`. `For` loops should be used when the number of *iterations* is known beforehand - as in 'Loop over these statements five times'. When the required number of iterations is unknown, or may be different for each run of the program, use a while loop.

3.4.1 The *for* loop

The basic syntax for a 'for' loop is: -

```
for var = start_value : end_value
    statements;
end
```

`var` is the name of the counter variable. `var` will take consecutive values starting with `start_value` and proceeding with `start_value` +1, `start_value` +2, and so on, until `end_value` is reached, but not exceeded. For example, let `start_value` = 1 and `end_value` = N. In this case, `var` will take consecutive values 1, 2, 3, ... N. The last iteration will be at `var` = N because $N + 1$ exceeds `end_value`.

The 'for' loop with `i = 1:N` is illustrated in Figure 3.3.

The loop variable does not have to be an integer. Consider the fragment;

```
for w = -0.8:4.1
```

The loop variable w will take consecutive values:

```
-0.8   0.2   1.2   2.2   3.2
```

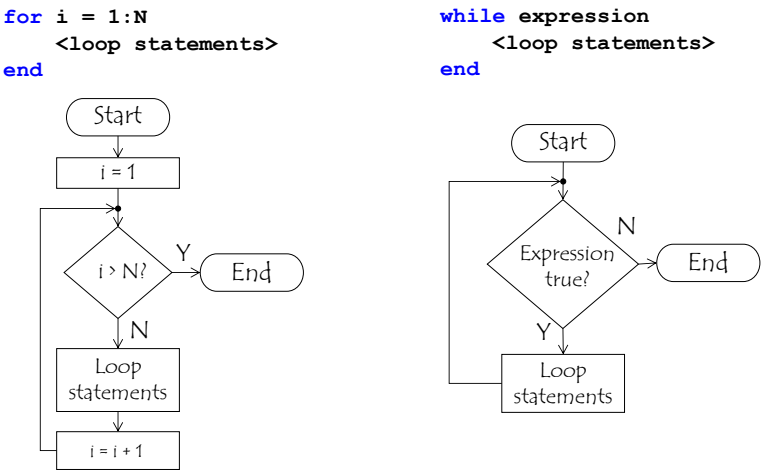


Figure 3.3: An illustration of 'for' and 'while' loops.

There will be only 5 iterations of this loop because $3.2+1 = 4.2$ exceeds the `end_value` of 4.1.

The loop variable does not have to be a number either. Consider the following example:

```
for t = 'a':'h'
   disp(t)
end
```

The loop will display a column of the lower-case letters from 'a' to 'h'.

The default increment of the loop variable is 1. We can specify a different increment value, just as with the colon operator. For example,

```
for w = 6.2:3.1:10.7
```

will run through the loop statements twice, for values of w 6.2 and 9.3. The next value of w would be $9.3 + 3.1 = 12.4$, which exceeds the `end_value` of 10.7.

With the functionality offered by the colon operator, the for loop can run 'backwards'. This fragment,

```
for k = 12:-2:-4
```

will run through the loop statements 9 times, for these values of k: -

```
12     10      8      6      4      2      0     -2     -4
```

An alternative form of the `for` loop, uses a matrix (or vector):

```
for var = matrix
    statements;
end
```

If `matrix` is a vector, `var` takes each subsequent value from the vector. For example,

```
for v = [2 4 6 8 10 6 6 6 14]
    disp([repmat(' ',1,18-v),repmat('.',1,2*v)])
end
```

will plot a little house made of dots, in the MATLAB Command Window.

3.4.2 The *while* loop

The `while` loop uses a logical expression (condition) to determine when to exit. Whilst the expression is true, the loop continues. The statements in the loop must lead to a change in the expression value, eventually rendering it false and exiting the loop.

The syntax for a `while` loop is as follows:

```
while expression
    statements;
end
```

The 'while' loop is illustrated in Figure 3.3.

3.5 Examples

3.5.1 Brute Force Sorting

Code up the 'brute force' sorting algorithm using a loop.[1] Assume that the array to sort is numerical, and is stored in A. Sort the numbers in descending order. Do not use an auxiliary array; you are only allowed to manipulate the values of A. For this task, you are not allowed to use `min`, `max`, `sort`, or any other MATLAB command to that effect. Show an example with a hand-picked array A.

Solution. Suppose that there are N elements to sort. Organise a loop which goes from 1 to $N-1$. Naturally, the loop does not have to go to N because, once $N-1$ values are sorted, so is the last one.

In each pass through the loop, one value will be placed in its destination. The first pass will identify the largest element and place it at the top of the array. The second largest element will be identified in pass 2, and will replace the second element of A. Thus, at iteration i, we will be positioning an element at i in A.

To identify the largest element, set a variable to the smallest value MATLAB can handle, `-Inf`. Search through the unsorted part of the array by comparing the current maximum with the array entry. If array entry is larger, store the value as the new maximum, and the index where this maximum value is found.

[1] This algorithm identifies the largest element of the array and places it first, then the second largest, and places it second, and so on until the smallest element is placed last in the array.

The code is shown below

```
A = [12,-900,4,2016,11,16]; % array to be sorted
N = numel(A);
for i = 1:N-1
    cm = -Inf; % initialise current maximum
    for j = i:N % search the *unsorted* part of the array
        if A(j) > cm
            cm = A(j); % store the new maximum
            index_max = j; % store the index of the new maximum
        end
    end
    A([i index_max]) = A([index_max i]); % swap
end
disp(A') % display the sorted array in a column
```

Listing 1: Brute force sorting in descending order

Note how easy it is to swap two elements in an array in MATLAB: `A([i index_max]) = A([index_max i]);`. If another language was used, we first need to save the content of one of the cells in a temporary variable, for example, `temp = A(i);` Then the ith entry can be replaced as `A(i) = A(index_max);`. Finally, the saved value should be placed in the array as `A(index_max) = temp;`.

3.5.2 When is a while loop useful?

Let A be an unsorted array of 10×2 random numbers between 0 and 1. Starting from the beginning of the array, find and display the numbers of the first 3 rows for which $A(i,1) > A(i,2)$. If there happens to be no such set of rows, print a message to that effect.

To show the work of the code, use the following command to generate A: `A = rand(10,2);`.

Solution. The loop may go through just the three top rows of A and complete the task, or through the whole of A and still not find three rows satisfying the condition. The code is shown below.

```
A = rand(10,2);
rc = []; % set of the indices of the rows where A(i,1) > A(i,2)
i = 1; % index counter for A
while numel(rc) < 3 && i <= size(A,1)
    if A(i,1) > A(i,2)
        rc = [rc,i]; % store the found row
    end
    i = i + 1; % increment the index for A
end
if i == size(A,1)+1 && numel(rc) < 3
```

```
    disp('A set of three rows was not found')
else
    disp(rc)
end
```

Listing 2: While loop example.

Notice how much easier the solution becomes if we use matrix operations (vectorising):

```
A = rand(10,2);
rc = find(A(:,1) > A(:,2)); % rows of interest
if numel(rc) < 3
    disp('A set of three rows was not found')
else
    disp(rc(1:3))
end
```

3.6 Exercises

1. Without running MATLAB, evaluate b_1 to b_6 in the following sequence of expressions:

```
>> a = [0 1 2;2 1 0];
>> b1 = a(1,1) > a(2,1)
>> b2 = a(2,2) && a(2,3)
>> b3 = a(1,1)+a(2,3)||a(2,2)-a(2,1)
>> b4 = a(:,2) > a(:,1)
>> b5 = b3 && a(1,1) < a(2,2)
>> b6 = find(a(1,:) == a(2,:))
```

2. Find and display all integers between 1 and 10000 which divide by 37. Propose at least two different ways to solve this problem.

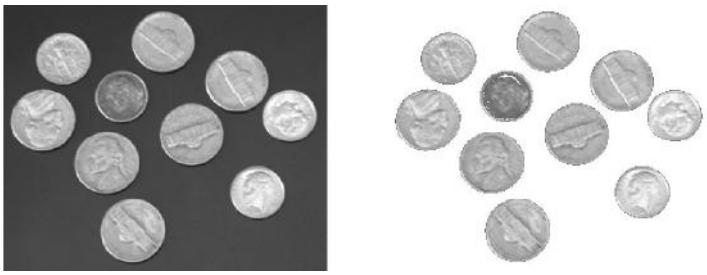
3. Load up and show MATLAB's image 'coins.png' (Figure 3.4 (a)) using the following line:

   ```
   z = imread('coins.png'); figure, imshow(z)
   ```

   This will enter a matrix z in MATLAB's memory. Values close to 0 correspond to dark, and values close to 255, to light pixels. Propose and implement a simple way to replace the background of the image with white, similar to the image shown in Figure 3.4 (b).

4. Similarly to the previous problem, upload image 'peppers.png' (Figure 3.5 (b)) using

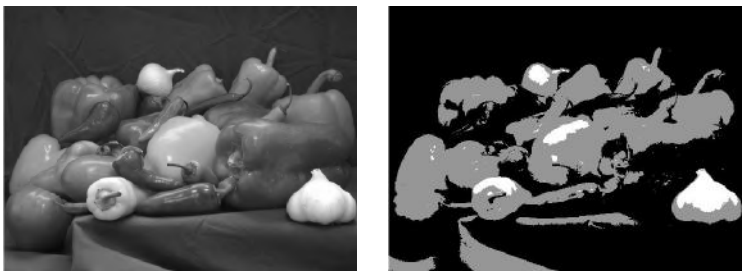
   ```
   z = rgb2gray(imread('peppers.png')); figure, imshow(z)
   ```



(a) Original (b) Desired image

Figure 3.4: The original coin image and the desired result.

This time, convert the image into three shades: black, grey and white, as shown in Figure 3.5 (b). (The appearance of the image only needs to be approximately the same as it will depend on the two thresholds which you choose.)



(a) Original (b) Desired image

Figure 3.5: The original peppers image and the desired result.

5. Fibonacci numbers form a sequence starting with 0 followed by 1. Each subsequent number is the sum of the previous two. Hence the sequence starts as 0, 1, 1, 2, 3, 5, 8, 13, ... Calculate and display the first 10 even Fibonacci numbers.

6. Figure 3.6 shows 8 scatterplots in 2d. Suppose that you are given the coordinates of a point (x, y). For each scatterplot, write a *single* logical expression which will yield TRUE if the point is in the black region and FALSE, otherwise.

7. Ask the user to input an integer in the range from 10 to 500. (Look up and use the `input` command.) If the input number is not an integer or is outside the limits, keep asking for a new number. Store the number in a variable *N*.

8. Write MATLAB code for the 'Guess My Number' game. First, the computer picks a random integer between 1 and 10 using the `randi` command. Next, the user is asked to enter their guess. If the guess matches the chosen number, display a congratulations message. Otherwise, display a 'better luck next time' message.

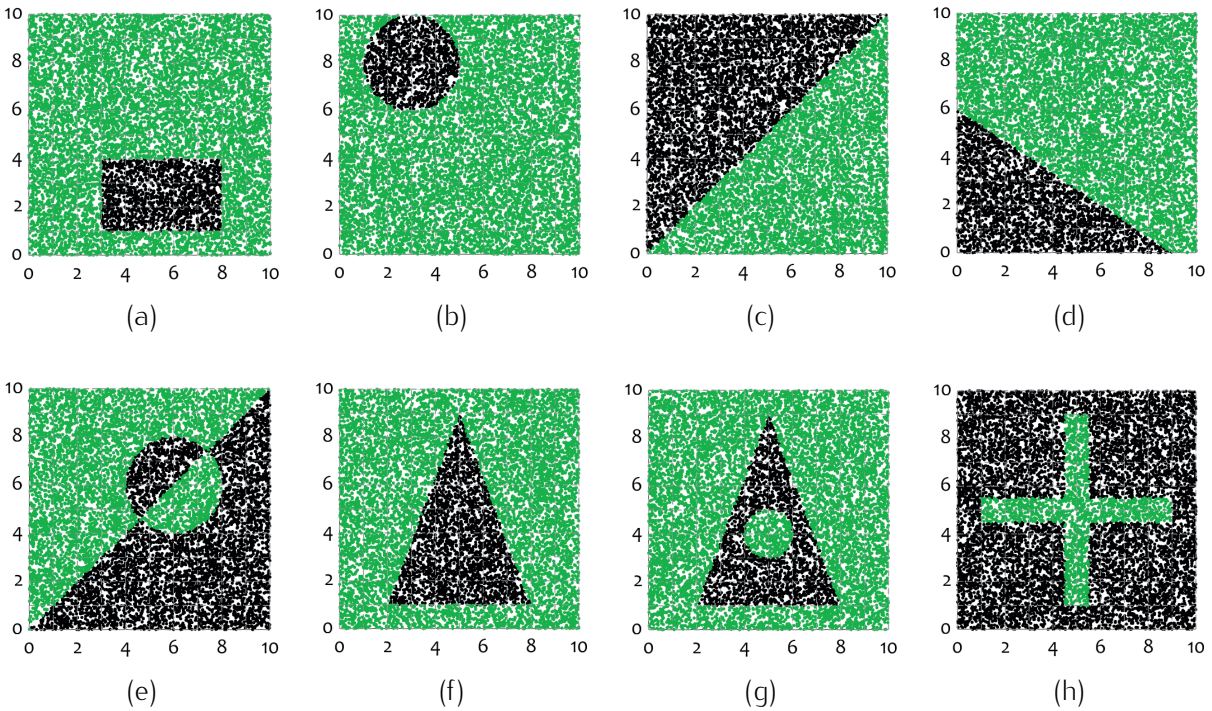


Figure 3.6: Region configurations for the logical expression problem.

9. Simple Image Filter. Load up and show MATLAB's image 'gantrycrane.png' using the following line:

```
z = rgb2gray(imread('gantrycrane.png')); figure, imshow(z)
```

Create a new image where every element of matrix z is replaced by the minimum of its neighbourhood values. The neighbourhood includes the central element and the surrounding 8 elements. Exclude the top and bottom rows of elements, as well the left and the right edges. *Use nested loops.* Repeat the process, but this time replace the value with the maximum within the neighbourhood. The resultant images should look like the ones in Figure 3.7.

(a) Original (b) Minimum filter (c) Maximum filter

Figure 3.7: The output of the minimum and maximum filters.

10. Consider a grid of size $n \times m$ with virtual bugs. Each bug lives in a grid cell. An example of the grid for $n = 20$ and $m = 30$ is shown in Figure 3.8. The grid is given to you in a form of a matrix A of size $m \times n$, with 0s in the empty cells and 1s in the cells occupied by bugs.

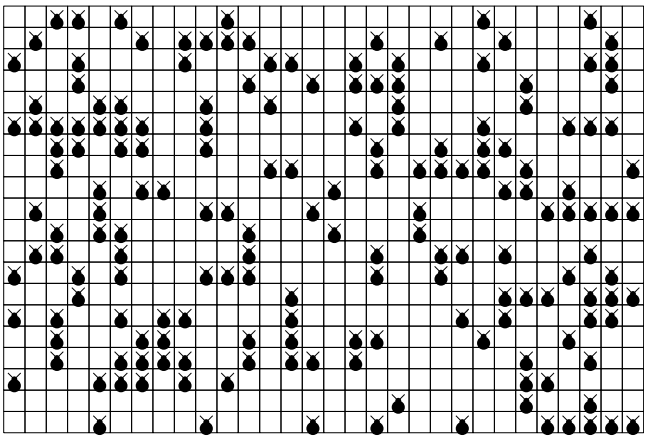


Figure 3.8: The virtual bugs grid.

(a) Find out the average number of neighbours per bug. Neighbours are considered to be the bugs in the 8 surrounding cells.

To demonstrate your work, create grids of different densities using:

```
A = rand(m,n) < 0.1; % sparse
A = rand(m,n) < 0.5; % medium
A = rand(m,n) < 0.7; % dense
```

(b) MATLAB includes a version of Conway's Game of Life. It is started by typing `life` in the command window. The rules are as follows:[2]

- Any bug with fewer than two neighbour bugs dies from isolation.
- Any bug with two or three neighbour bugs lives on to the next generation.
- Any bug with more than three neighbour bugs dies from overcrowding.
- Any empty cell with exactly three neighbour bugs becomes a bug, as if by reproduction.
- The rules apply in the same way to the edge and corner cells even thought they have fewer physical neighbour cells.

Using these rules, calculate the next generation of bugs for your randomly populated grid.

(c) Use the command `spy` to see the grid. Include it in a loop where you evolve the population with each pass, visualise it with `spy`, and pause the execution with command `pause(0.2)`.

[2] `http://en.wikipedia.org/wiki/Conway's_Game_of_Life`

(d) Glider gun.[3]

Run your code from the previous sub-problem to evolve the 'glider gun' population whose starting layout and the first five generations are shown in Figure 3.9.

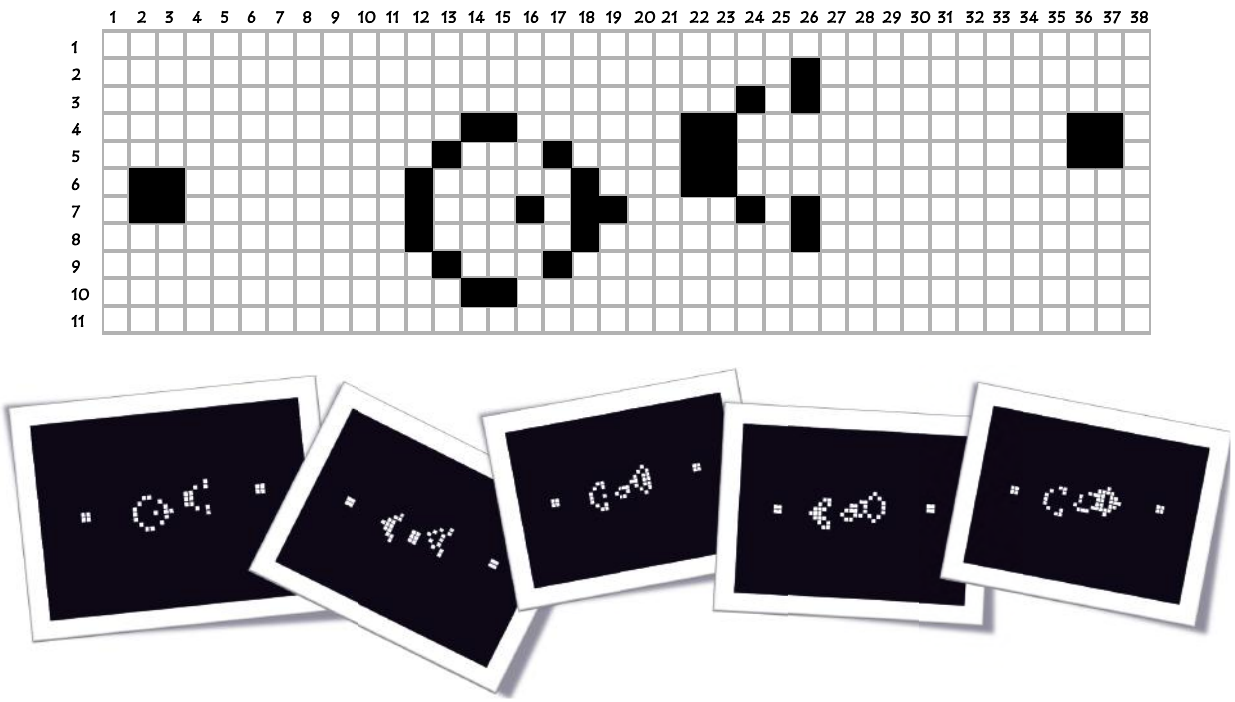


Figure 3.9: The glider gun's initial configuration and first five generations.

[3] http://commons.wikimedia.org/wiki/File:Game_of_life_glider_gun.svg

Chapter 4

Functions

Functions are useful when a certain part of the code must be repeated many times or if we want to make it into a basic tool available for future use.

4.1 Syntax

Functions in MATLAB are stored in separate files. Unless specifically declared as `global`, all variables are local, which means that they are valid only within that function. Think about a function as a 'watertight' piece of code. Its communication with the outside world are the input and the output variables.

The formal syntax for a function definition is:

```
function [list_of_output_arguments = ]function_name([list_of_input_arguments])
```

Neither of the lists is compulsory. If there are output arguments, they must receive values in the body of the function. An interesting property of MATLAB functions is that not all input and output arguments need to be assigned in the call. For example, consider a one-dimensional array A.

`m = min(A);`	will return the minimum of array A
`[m,i] = min(A);`	will return the minimum of array A in m, and the index where the minimum is found in i
`[~,i] = min(A);`	will return only the index of the minimum of A in i.

Replacement of the output argument by a tilde was introduced fairly recently (2009b, MATLAB Version 7.9). Until then, a dummy variable had to be used in its place.

An illustration of a simple function is shown in Figure 4.1.

A call to `fu` is made in the MATLAB script and the output is assigned to variable `m`. The function is stored in a separate file of the same name (`fu.m`). It returns two outputs: the sum of the three input arguments and their product. As the function is called with one output argument only, it will return in `m` the sum $4 + 2 + 1 = 7$.

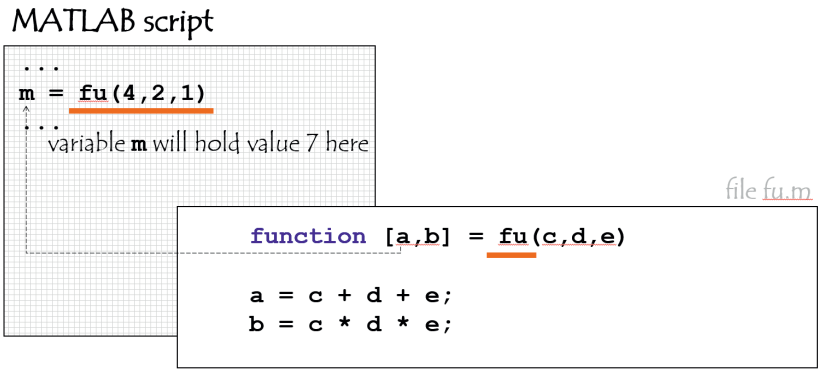


Figure 4.1: Illustration of a simple function.

4.2 Naming

A call to a function will look for a *file* with the function's name in the MATLAB path. Therefore, it makes sense to set the file name and the function name to be the same. For example, if your function is called `draw_trapeze` but the file is named `trapeze.m`, MATLAB will not accept a call to `draw_trapeze`. It will, however, accept a call to `trapeze.m`, and will subsequently run file `trapeze` regardless of the name you have specified as the function declaration within the file (`draw_trapeze`).

4.3 Multiple Functions

Multiple functions can be included in the same *function* file. As explained above, MATLAB will refer to that collection of functions only through the file name. This structure is useful when functions in the file need to call one another.

An example of multiple functions in the same file is shown in Figure 4.2.

In this example, the script calls function `fu2`. Function `fu2` will be visible to MATLAB while function `fu` will not. Function `fu` is accessible only within `fu2`. Function `fu2` returns two output arguments. In `p`, it stores the sum of the three input arguments divided by their product, if the product is not zero. If the product is zero, an empty value is returned (`p = []`). The second output argument, `q`, is the sum of the sum and the product of the three inputs. In this example, only second argument of `fu2` is requested by the MATLAB script, therefore we store in `s` $4+2+1+4\times 2\times 1 = 15$.

Multiple functions are particularly useful for writing graphical user interface (GUI) code.

4.4 Inline (Anonymous) Functions and Function Handles

An inline function is defined as in the following example:

```
fu = @(x) x.*sin(x)-exp(-x.^2);
```

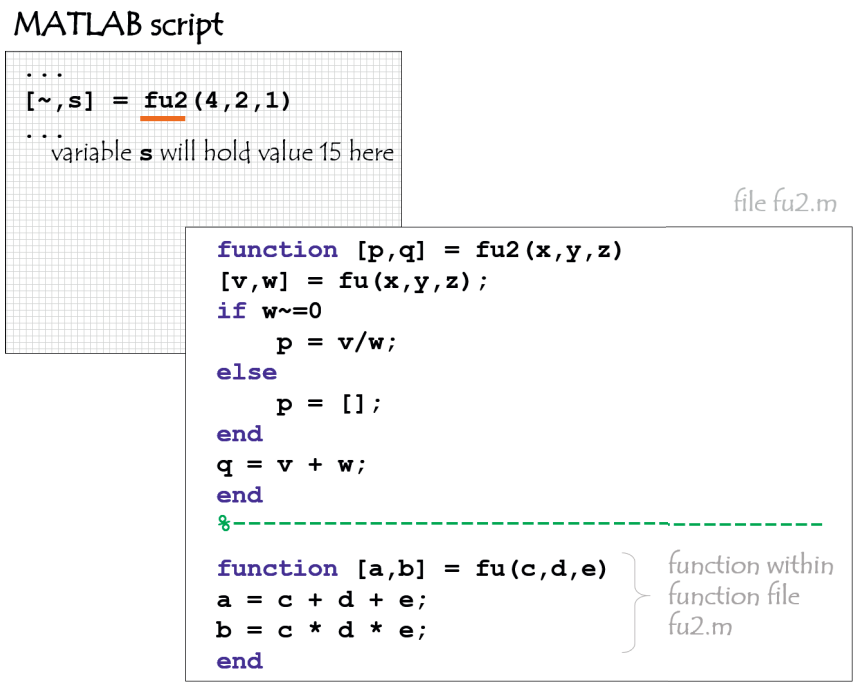


Figure 4.2: Illustration of multiple functions in the same file.

From this point onwards, you can use `fu(arg)`, where the argument can be a single number or an array. (The array operations are accounted for by using '`.*`' instead of just '*', and '`.^`' instead of '^'.) For example,

```
figure, plot(0:.3:20,fu(0:.3:20)), grid on
```

will open up a figure and plot the value of the function $f(x) = x\sin(x) - \exp(-x^2)$ for $x \in \{0, 0.3, 0.6, 0.9, 1.2, \ldots, 19.8\}$.

In the function definition, instead of a single `x`, we can use a list of arguments after the `@` sign (`@(list_of_arguments)`). For example,

```
fu = @(p,q) p.^q - 2*(p-q);
```

4.5 Recursion

As many other programming languages, MATLAB allows recursion, which means that a function can call itself. Recursion usually leads to very elegant code but this does not offer computational advantage.

Only a small number of iterative algorithms can be solved using recursion. An example is the bisection algorithm for finding a zero of a function within a given interval. Consider the function

$$f(x) = \sin(2x)\ \exp(x-4)\ .$$

There is one zero of the function in the interval [1,2]. Find the zero x^*, such that $f(x^*) \approx 0$, with precision[1] $\epsilon = 10^{-6}$. This means that an interval with centre x^* and length ϵ contains the true point where $f(x)$ crosses the x-axis.

The algorithm goes through the following steps:

1. Input the precision ϵ, the function $f(x)$ and the bounds a and b of the interval containing the zero.
2. Calculate the middle of $[a, b]$
$$m = \frac{a+b}{2} .$$
3. If the length of interval $[a, b]$ is smaller than ϵ,
   (a) then return m.
   (b) else, if $f(a)f(m) < 0$ (the zero is in $[a, m]$) call the function with interval $[a, m]$, else call the function with interval $[m, b]$.

The code of the MATLAB function is given below.

```
function x = bisection(e,f,a,b)
x = (a + b) / 2;
if (b - a) > e
    if f(a)*f(x) < 0
        x = bisection(e,f,a,x);
    else
        x = bisection(e,f,x,b);
    end
end
```

The following script calls function `bisection` for equation function $f(x) = sin\ 2x \exp x - 4$ (LHS of the equation) and interval $[1, 2]$ with precision 10^{-3}:

```
e = 10^-3; % precision
f = @(x)  sin(2*x) * exp(x-4); % equation LHS
xstar = bisection(e,f,1,2);
disp(xstar)
```

The MATLAB output is 1.5708. If you substitute 1.570 and 1.571 for x, $f(x)$ should have function values with different signs.

4.6 Exercises

1. Write a MATLAB function for calculating the Euclidean distance between two points in the n-dimensional space. The points are given as the input arguments a and b. Both should be arrays

[1]The word 'precision' here is used to denote the length of the interval containing the solution. This may differ from the meaning of this word in other contexts.

with n elements. It should not matter for your function whether either input is a row or a column. Demonstrate the work of your function by an example.

2. Write a MATLAB function for calculating the Euclidean distance between two two-dimensional arrays A and B given as input arguments. A is of size $N \times n$, and B is of size $M \times n$. The function should return a matrix D of size $N \times M$ where element $d(i, j)$ is the Euclidean distance between row i of A and row j of B.

3. Write an inline function that will calculate $6x - 4y + xy + cos^2(x - k)$. Assume that x and y may scalars, vectors or matrices and that all operations should be carried out element-wise.

4. Write a MATLAB function for checking if a given point (x, y) is within the square with a bottom left corner at (p, q) and side s. The input arguments are x, y, p, q and s, and the output is either true (the point is in the square) or false (the point is not in the square).

5. Write a non-recursive MATLAB function to calculate the Fibonacci sequence and return the number with a specified index, for example, the 4th number in the series (this number is 5).

6. Write a recursive MATLAB function to calculate the Fibonacci sequence and return the number with a specified index.

7. Write a *short* MATLAB function to find out whether a given number (up to 1,000,000) is a prime number. The function should return true or false. Bear in mind that 1 is not considered a prime number. (Hint: Use the brute force approach and divide the number (K) by all integers from 2 to K-1. Check the remainders for 0s.)

   Subsequently, apply this function to list all prime numbers between 1 and 100.

8. Write your own function for the bubble sort algorithm and demonstrate its work.

Chapter 5

Plotting

5.1 Plotting Commands

5.1.1 Plot

The `plot` command is easily one of the most useful MATLAB commands. It needs at least one argument as shown in Figure 5.1. If there is no open figure, MATLAB will open a new one and will plot the argument (an array) versus its index. If there are two arrays as input arguments, MATLAB will take the first array to be the x-coordinates, and the second array, the y-coordinates.

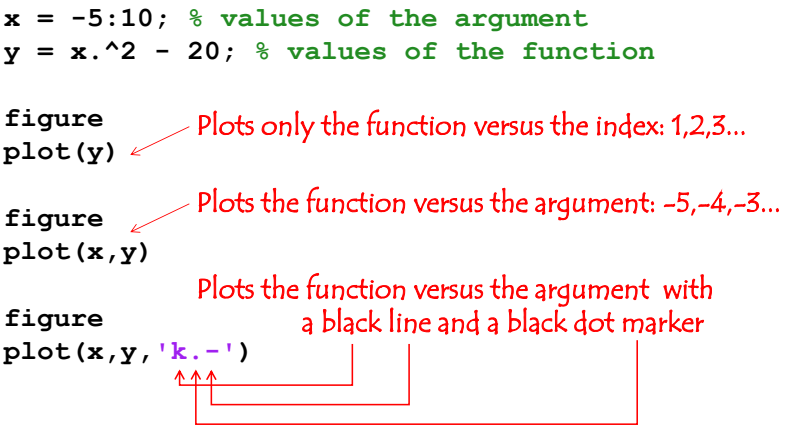


Figure 5.1: Plot command

A third string argument can specify the type of line, colour and marker of the plotted line. The three figures in the example in Figure 5.1 are shown in Figure 5.2.

If you want to plot more than one thing on the same figure, use the command `hold on`. The grid lines can be toggled on and of with the command `grid` (use `grid on`).

Figure 5.3 gives some more detail about the `plot` command and the appearance of MATLAB figures.

There is lot more to be learned about the plot command as shown by the examples in the next section.

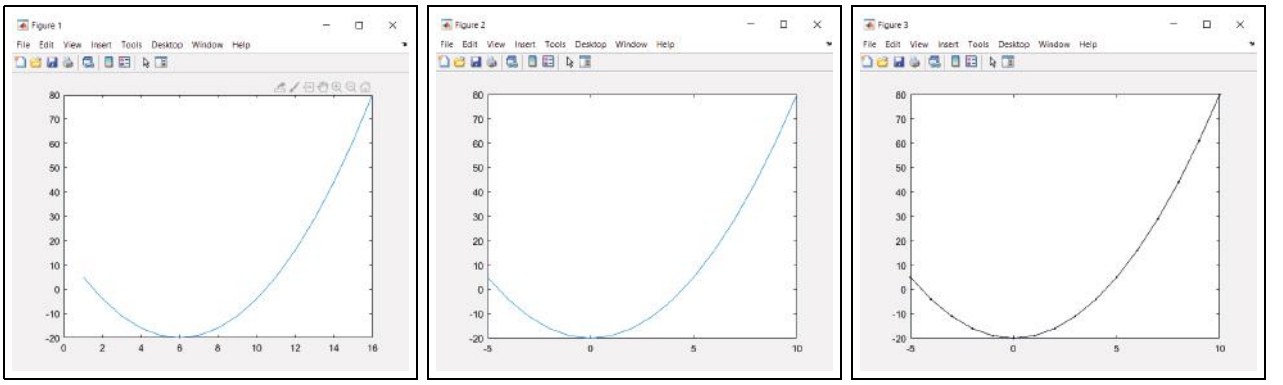


Figure 5.2: Output from the code in Figure 5.1

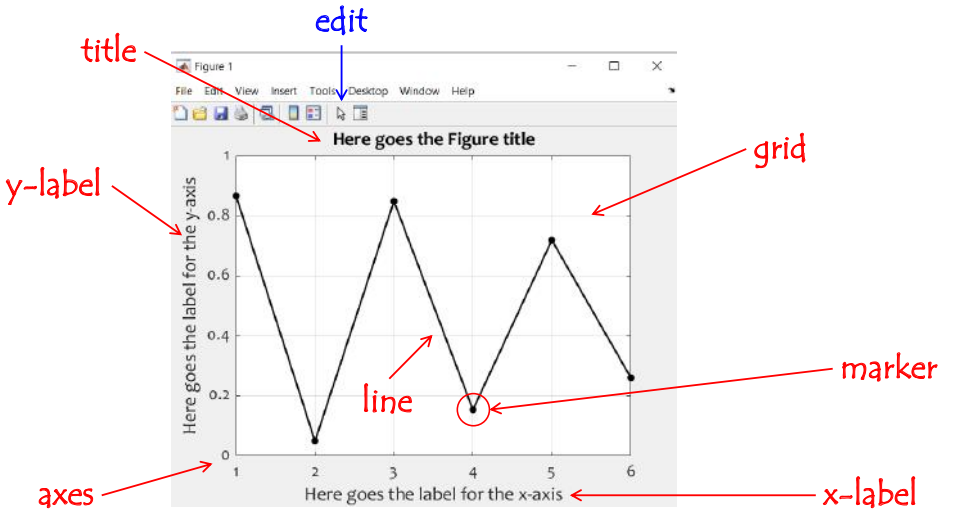


Figure 5.3: Figure with axes.

5.1.2 Fill

The `fill` command fills a specified region with a specified colour. The syntax is:

```
fill(x,y,colour)
```

The first argument is an array with the x-coordinates of the shape to be filled, and the second argument is an array of the same size with the y-coordinates. The third argument is either one of the pre-defined colour strings:

'k'	black	'r'	red	'g'	green	'b'	blue
'w'	white	'm'	magenta	'y'	yellow	'c'	cyan

or an array $[a, b, c]$ with three numbers between 0 and 1. These three numbers make up the colour, by mixing a amount of red of possible 1), b amount of green and c amount of blue. For example, colour

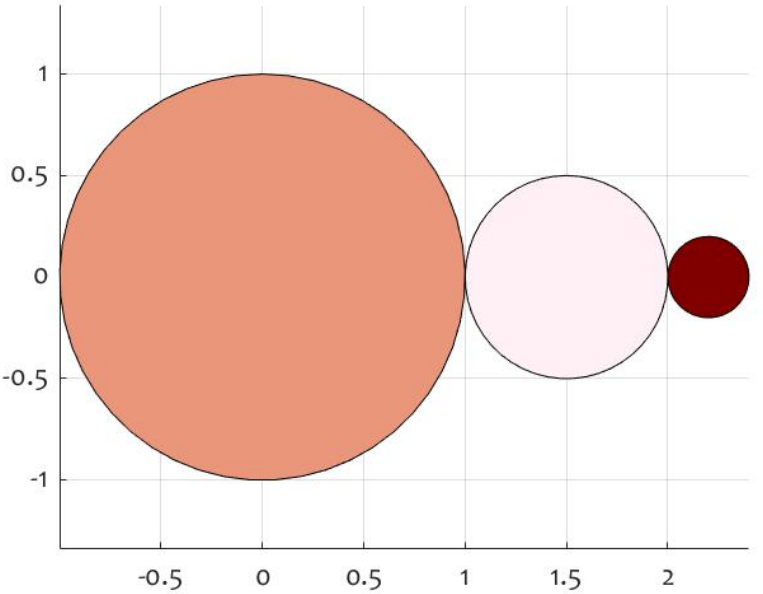


Figure 5.4: Example of the output of the fill command.

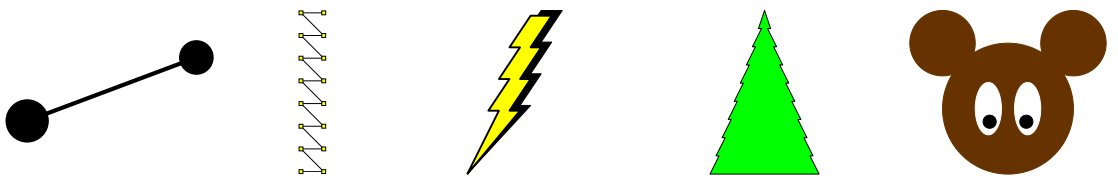
'dark salmon' is made by [0.9137, 0.5882, 0.4784]. Figure 5.4 shows three circles filled respectively with colours dark salmon, lavender bush and maroon. The code is shown below.

```
figure, hold on
t = linspace(0,2*pi,50); % 50 angles from 0 to 2*pi (in radians)

fill(sin(t),cos(t),[0.9137, 0.5882, 0.4784]) % circle centred at [0,0]
% with radius 1, filled with dark salmon colour.

fill(0.5*sin(t)+1.5,0.5*cos(t),[1 0.9412 0.9608])
% circle centred at [1.5,0] with radius 0.5, filled with lavender bush
% salmon colour.

fill(0.2*sin(t)+2.2,0.2*cos(t),[0.5020 0 0])
% circle centred at [2.2,0] with radius 0.2, filled with maroon colour.

grid on
axis equal
```

Notice two things. First, there is black outline (by default) fo all 'fill' objects. Second, we used command `axis equal`. This makes the units on the x-axis and y-axis to become of equal size. This way the circles looks like circles and not ellipses.

5.2 Examples

Reproduce the shapes and plots in Figure 5.5.



(a) Weights (b) Zigzag (c) Lightning bolt (d) Christmas tree (e) Bear's face

Figure 5.5: MATLAB plot examples.

Plot (a) can be constructed with one thick line and circle markers. To create the illusion of perspective, the ball which is closer to the viewer should be larger. This can be achieved by plotting a larger circular marker over the smaller one. The line width and the sizes of the two markers are determined by trying out different values until the figure meets the designer's approval. The code is shown below:

```
% Weights
figure, hold on, axis equal off % set up the figure and format the axes
plot([0 0.8],[0 0.3],'k.-','markersize',200,'linewidth',8)
plot(0,0,'k.','markersize',250) % plot a second, larger marker
```

The zigzag in plot (b) is based on a repeated pattern of x coordinates, e.g., [0 1 0 1 0 1 ...], while the y coordinate must increase as [1 1 2 2 3 3 ...]. Both patterns can be created using matrix manipulation as shown in the code below:

```
% Zigzag
figure, hold on, axis equal off % set up the figure and format the axes
x = repmat([0,1],1,8); y = [1:8;1:8];
plot(x,y(:),'ks-','markersize',8,'MarkerFaceColor','y')
```

For the lightning bolt in plot (c), the fill command should be used. The shadow has the same shape as the yellow bolt, and is displaced on both x and y. The bottom point of the shadow is 'stretched' to match the tip of the yellow spear. Note; that the shadow must be plotted first. The code is shown below;

```
% Lightning bolt
figure, hold on, axis equal off
x = [-2 3 2 4 3 5 4 6 4,2,3,1,2,0,1,-2]; % yellow X
y = [-1 1 1 2 2 3 3 4 4 3 3 2 2 1 1 -1]*3; % yellow Y
xsh = x + 1; xsh(1) = -2; xsh(end) = -2; % shadow X
```

```
ysh = y + 0.5; ysh(1) = -3; ysh(end) = -3; % shadow Y
fill(xsh,ysh,'k') % plot shadow first
fill(x,y,'y','edgecolor','k','linewidth',3) % plot yellow bolt
```

The Christmas tree in plot (d) is formed from two symmetrical parts. The x-values can be constructed for one of the parts and flipped for the other part. In the code below, both x and y are constructed initially as arrays with two rows. Then, with the help of the colon operator, the two rows are concatenated column-by-column to make the needed sequence of vertices. For example, if x has values [0 2 4 6] in the first row and [0 1 3 5] in the second row, the concatenated (and transposed) vector will be [0 0 2 1 4 3 6 5]. This gives the sawtooth pattern for the periphery of the tree. The code for the Christmas tree is as follows:

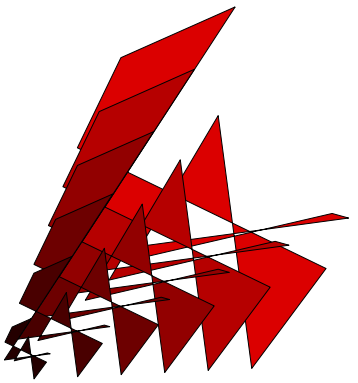
```
% Christmas tree
x = [0:2:18;0 1:2:17]; y = [20:-2:1;20:-2:1]*3;
x = [-fliplr(x(:)') x(:)']; y = [fliplr(y(:)') y(:)'];
figure, hold on, axis equal off, fill(x,y,'g')
```

The Bear's face in plot (e) would be difficult to plot with markers of different sizes. It is better to use filled circles as shown in the code below:

```
% Bear's face
brown = [0.4 0.2 0]; % colour definition
figure, hold on, axis equal off
t = linspace(0,2*pi,100);
fill(sin(t),cos(t),brown,'EdgeColor',brown) % face
fill(sin(t)*0.5+1,cos(t)*0.5+1,brown,'EdgeColor',brown)
fill(sin(t)*0.5-1,cos(t)*0.5+1,brown,'EdgeColor',brown)
fill(sin(t)*0.2+0.3,cos(t)*0.4,'w','EdgeColor','w')
fill(sin(t)*0.2-0.3,cos(t)*0.4,'w','EdgeColor','w')
fill(sin(t)*0.1+0.28,cos(t)*0.1-0.2,'k','EdgeColor','k')
fill(sin(t)*0.1-0.28,cos(t)*0.1-0.2,'k','EdgeColor','k')
```

Random art can be created using filled polygons, not necessarily convex, with random vertices. For example, `fill(rand(10,1),rand(10,1),rand(1,3))` will create a random shape of joined line segments, where some of the closed spaces will be filled with a random colour. Figure 5.6 shows the output of the following piece of code:

```
x = rand(10,1); y = rand(10,1); % vertices of the polygon
figure, hold on, axis equal off
k = 6; % # of repetitions of the same shape
for i = 1:k
    fill(x*(k-i+1),y*(k-i+1),[k-i+1 0 0]/(k+1))
end
```



Note: To plot the shape at a different position, add the desired offset to the x and the y coordinates, respectively.

Figure 5.6: Repetitions of a shrinking random shape filled with progressively darkening red colour.

5.3 Exercises

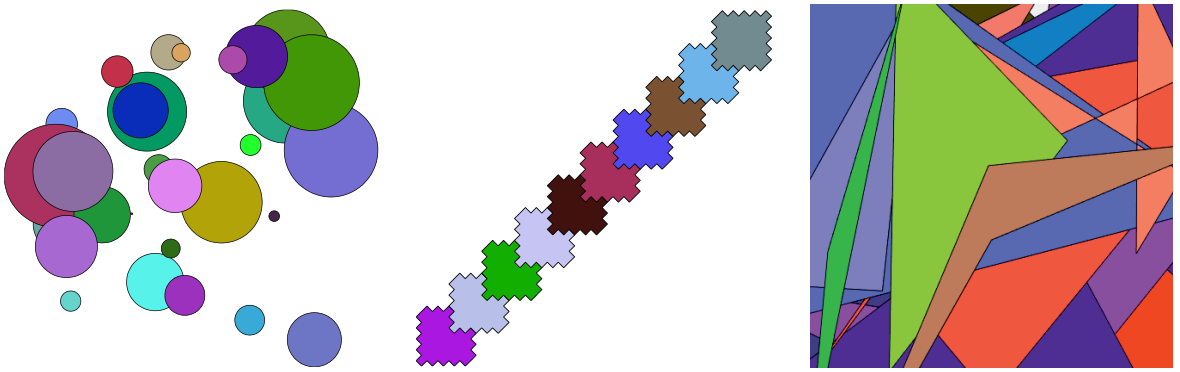
1. Plot the six European flags as in Figure 5.7. The names of the countries should be displayed as well. All flags should be plotted in one figure. This task should be completed using the the `subplot` command rather than adjusting spacing between the flags yourself.

Figure 5.7: Six European flags.

2. Write a function which will draw a circle in an open figure. The input arguments are x, y, r, c; x and y are the coordinates of the centre, r is the radius, and c is a three-element vector with the colour. Demonstrate your function by using it to plot 30 circles at random positions, with random radii, and with random colours, as in Figure 5.8 (a).

3. Create an art figure by plotting 10 filled squares with jagged edges as shown in Figure 5.8 (b). The fill colours should be random. The squares should be arranged approximately as in the example in the figure.

4. Plot a Random Art Square similar to the example in Figure 5.8 (c). There should be 20 random forms with random colours in the square. Note that some of the forms are not contained fully

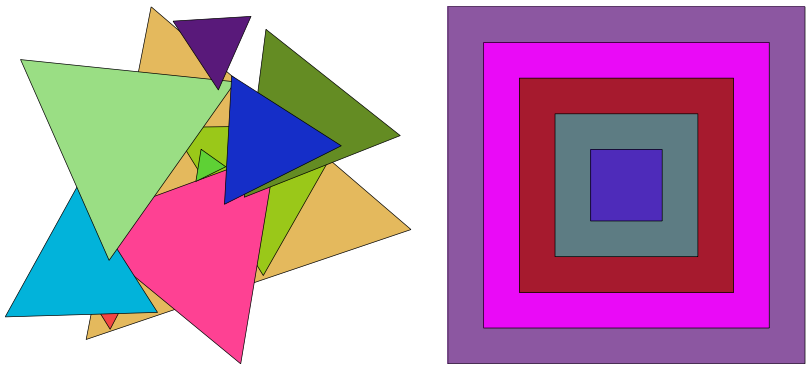


(a) Circles (b) Jagged edge art (c) Random art square

Figure 5.8: Output for problems 5.3.2, 5.3.3, and 5.3.4

in the figure. Each form must have between 3 and 6 vertices. The number of vertices should be random.

5. Create a loop to plot 10 triangles with random vertices in the unit square. Not every triangle should be plotted. Plot *only* triangles which are nearly equilateral. To do this, check whether the three edges differ by less than a chosen small constant, for example $e = 0.01$. The triangles should be filled with random colours. An example of the desired output is shown in Figure 5.9 (a).



(a) Equilateral triangles (b) Nested squares

Figure 5.9: Output for problems 5.3.5 and 5.3.6

6. Write a MATLAB function which takes an integer k as its input argument and plots k filled squares of random colours, nested as shown in Figure 5.9 (b).

7. Using the function from the previous problem, reproduce Figure 5.10. The number of squares k varies from 3 to 12. All colours are random.

8. Plot a shape consisting of four filled polygons. The polygons are mirror versions of one polygon with k random vertices, where k is a parameter. The figure should be symmetrical about the x *and* y axes. The polygons should touch in the middle point as shown in the examples in

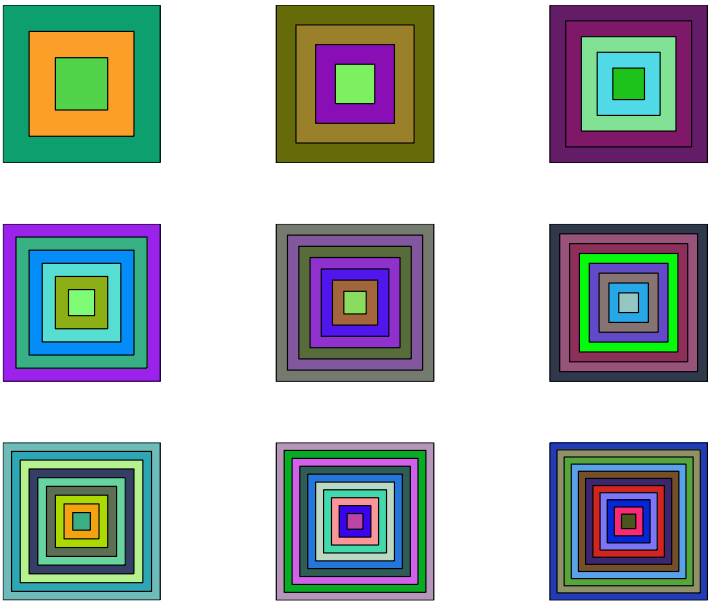
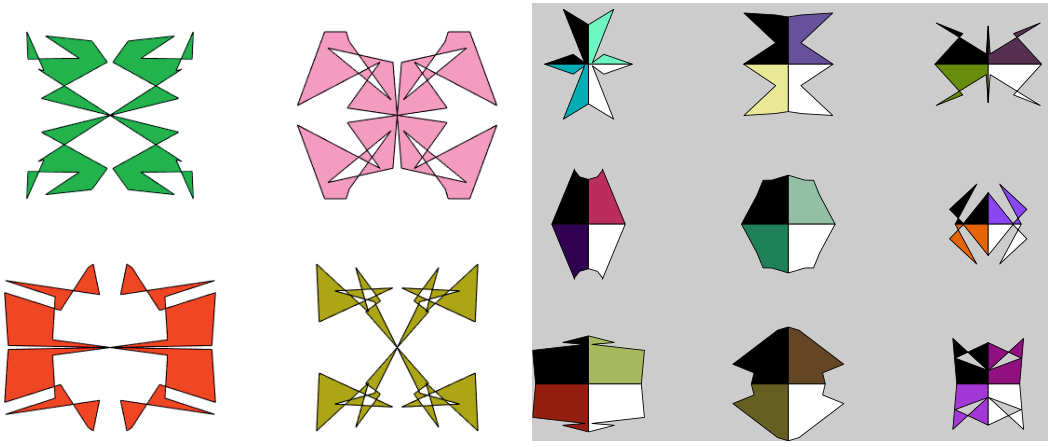


Figure 5.10: 3-12 filled nested squares.

Figure 5.11 (a) ($k = 10$). Try to accomplish this task using matrix operations, not geometric functions such as `fliplr`.



(a) Symmetric polygons (b) Logos

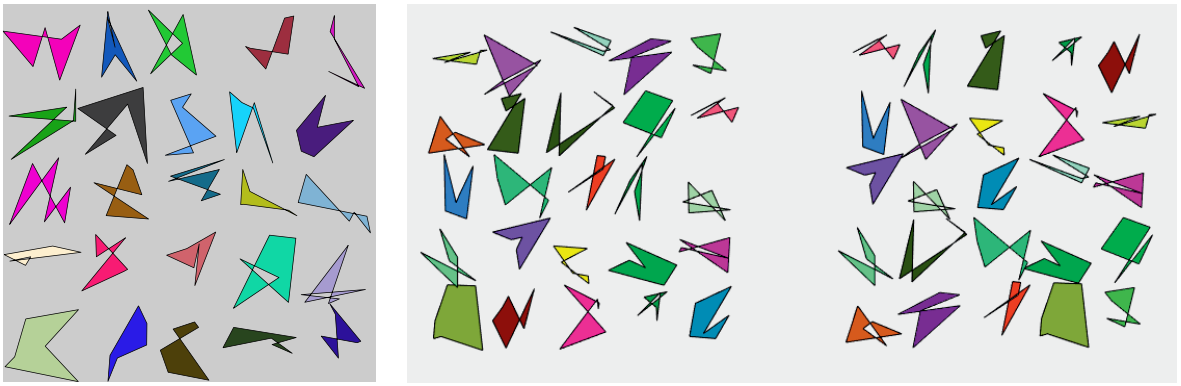
Figure 5.11: Output for problems 5.3.8 and 5.3.9

9. Write MATLAB code to produce a 4-part logo. Nine examples are shown in Figure 5.11 (b). You can use your own design of the basic shape, or pick it at random. The top left part should be black and the bottom right should be white. The other two colours should be chosen randomly by your program. The four shapes should have a common edge on the x-axis and on the y-axis. The length of each of these edges should be at least half of the span of the shape on the respective axis.

10. Random Shapes on a Grid

    (a) Create a set of 25 random shapes, each one having 6 random vertices and filled with a random colour. Plot the shapes on a 5-by-5 grid as shown in Figure 5.12 (a). This should not be achieved using the `subplot` command and can be accomplished with a single loop.

    (b) Make a figure with two subplots. The first subplot should contain the original shapes, and the second subplot should contain the same shapes, in a random order on the grid. An example is shown in Figure 5.12 (b).

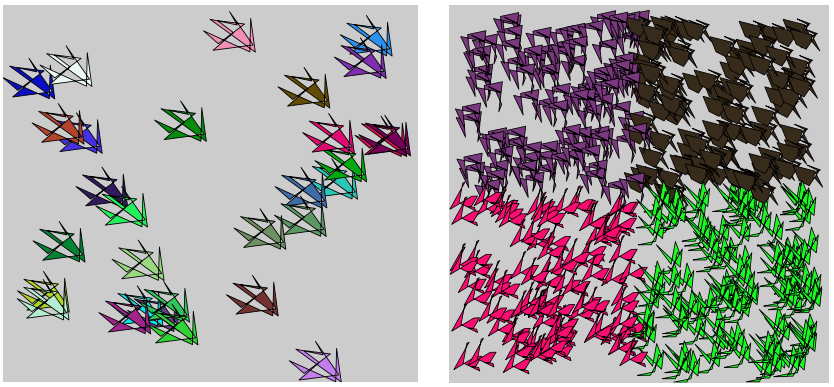


(a) 25 shapes on a grid (b) Original and shuffled

Figure 5.12: Output for problem 5.3.10

11. Birds, Butterflies or Flying Baba Yagas

    (a) Figure 5.13 (a) shows a field with randomly distributed copies of a filled small shape or creature. The shape is random, but fixed for the figure, while the colours and the positions of the replicas are random. Write MATLAB code to produce a similar figure.



(a) 25 Small-shape art (b) Four armies of creatures

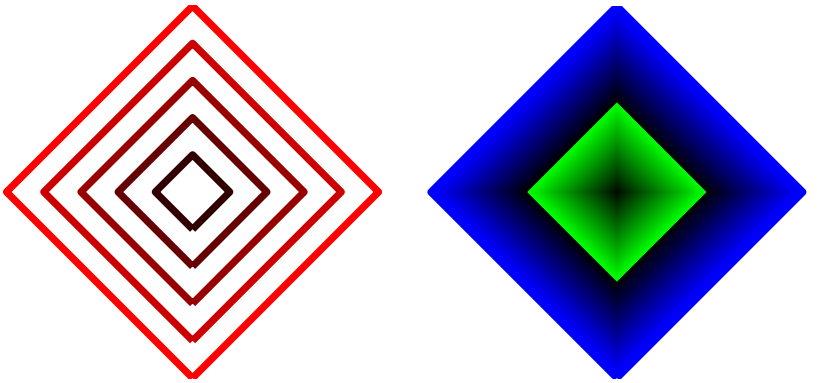
Figure 5.13: Output for problem 5.3.11

(b) Subsequently, design a battle scene, where four 'armies' of creatures are distributed in four parts of the space as shown in Figure 5.13 (b). The creatures from each army should have the same (random) shape and colour. The positions of the creatures within the regions are random too. (Note: The creatures are allowed to overlap near the borders.)

12. Diamonds in a Loop

(a) Use a loop to create 5 diamonds as in Figure 5.14(a) (one diamond in each pass through the loop). The innermost diamond is black, and the outermost is red. Each diamond has its own fixed colour. The colours of the diamonds go gradually from black to red.

(b) Subsequently, use one loop to create the pattern in Figure 5.14(b). The colour goes gradually from black to green followed by black to blue. The figure should be plotted as succession of diamonds.



(a) (b)

Figure 5.14: Diamonds.

13. Try to reproduce the row of Christmas trees in Figure 5.15.

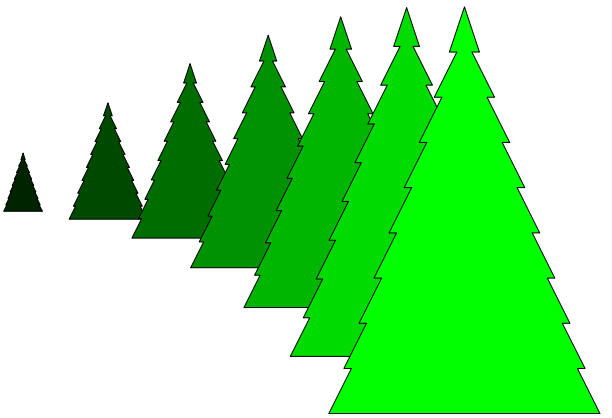


Figure 5.15: A row of Christmas trees.

14. Balloons

    (a) Create a MATLAB function called `draw_balloon`. The function should take as its input arguments: the x and the y coordinates of the centre, the radius r, the colour c, and the length of the string l. The function should plot the balloon on the current axes (held and equalised). An example is shown in Figure 5.16 (a). The function is called using the following line:

    ```
    figure, hold on, axis equal, draw_balloon(1,2,4,[0.2 0.6 0.9],7)
    ```

    (b) Subsequently, write a MATLAB script to produce a figure with 20 balloons with random colours and sizes (Figure 5.16 (b)).

    (c) Finally, produce another figure with 20 random balloons, all of which have reached the ceiling, as shown in Figure 5.16 (c).

(a) (b) (c)

Figure 5.16: Balloons for problem 5.3.14

15. Write a MATLAB function called 'dice'. The function should open a figure and display the given face of a regular six-sided die. The face 'number' is the only input argument, k. Examples of the desired output are shown in Figure 5.17. Notice the rounded corners. The orientation of faces 2, 3 and 6 does not matter as long as the white dots form the desired pattern.

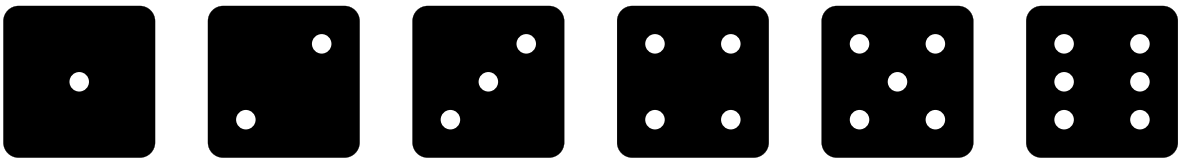


Figure 5.17: The 6 dice faces.

start CHALLENGE

Dice Face

The challenge for this problem is to write the shortest possible code for the Dice function. The length of the code is the number of characters ignoring the white spaces and new lines. (In real competitions, the variable names of any length are counted as one character, and comments are not counted at all. In our competition both of these will count.)

Current record (including the function declaration line) is 104 characters.

end CHALLENGE

Chapter 6

Data and Simple Statistics

6.1 Random Number Generation

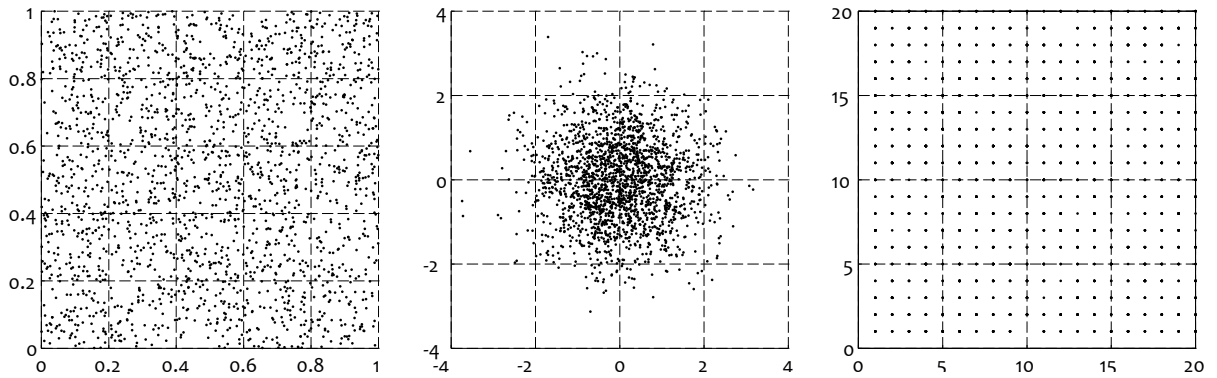
Some MATLAB commands for random number generation were mentioned before. Below is a list and a short description of these commands:

rand	Generates a random number with uniform distribution in the unit interval (interval $[0, 1]$).
rand(n)	Generates a square $n \times n$ matrix with random numbers in the unit interval.
rand(m,n)	Generates an $m \times n$ matrix with random numbers in the unit interval.
randn	Generates a random number from a standard normal distribution (mean 0 and standard deviation 1).
randn(n)	Generates a square matrix with random numbers from a standard normal distribution.
randn(m,n)	Generates an $m \times n$ matrix with random numbers from a standard normal distribution.
randi(a)	Generates a random integer from a uniform distribution between 1 and a.
randi(a,n)	Generates a square matrix with random integers between 1 and a.
randi(a,m,n)	Generates an $m \times n$ matrix with random integers between 1 and a.
randperm(a)	Generates a random permutation of the integers from 1 to a.
randperm(a,k)	Generates a random permutation of the integers from 1 to a and returns the first k elements.

Figure 6.1 shows an example of the output of the three random generators (rand, randn and randi).

6.2 Simple statistics and plots

The list below details MATLAB commands for calculating some simple statistics. All operations produce a single value if the argument a is a vector, and operate on each column separately, if a is a matrix.



(a) rand (b) randn (c) randi ($a = 20$)

Figure 6.1: Two-dimensional scatterplots for the three random generators.

Measures of Central Tendency	
mean(a)	Calculates the mean of a.
median(a)	Calculates the median of a.
mode(a)	Calculates the mode of a.

Measures of Variability	
std(a)	Calculates the standard deviation of a.
var(a)	Calculates the variance of a.
range(a)	Calculates the range of a.

Data can be summarised and visually presented using bar charts, pie charts and glyph plots, among many. Examples are shown in Figure 6.2.

Histograms summarise the data by splitting the range of the variable into bins, and then counting the numbers of data points in each bin. An illustration is shown in Figure 6.3. The histogram is calculated from a vector with 1000 value generated through the `randn` command using the following code:

```
a = randn(1000,1); figure, hist(a)
```

6.3 Examples

Generate 5000 random points in the unit square. Plot the data so that the points below the diagonal joining points (0,0) and (1,1) are shown with blue crosses, and the ones above the diagonal, with red triangles, as in Figure 6.4.

The first step after clearing the memory, the Command Window and closing the current figures, is to open a new figure, and format the axes:

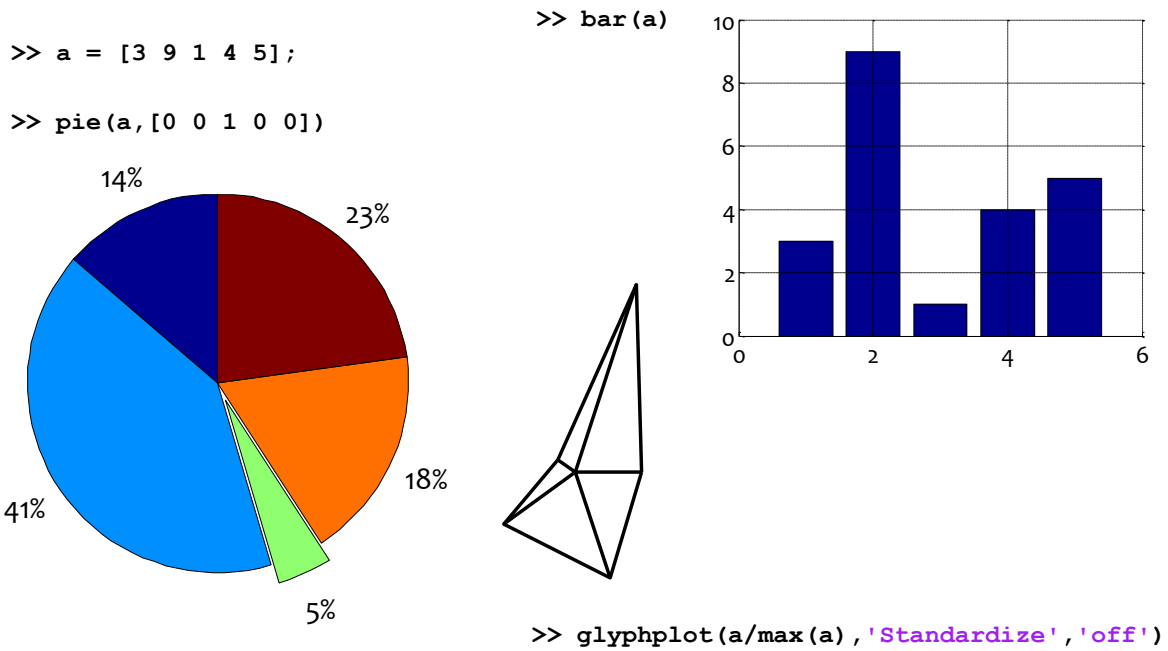


Figure 6.2: Examples of bar graph, pie chart and glyph plot

```
figure, hold on, axis([0 1 0 1],'square')
```

You can format the axes further by setting the font name and size. Compare the following two continuations for this problem:

1. Generate and plot the random points in a loop.

```
for i = 1:5000
     x = rand; y = rand; % generate a random point
     if x > y
         plot(x,y,'bx')
     else
         plot(x,y,'r^')
```

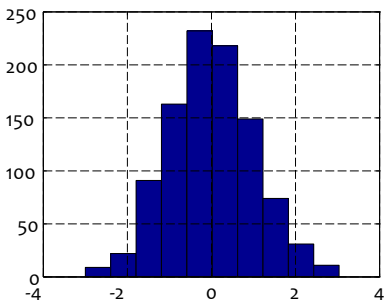


Figure 6.3: An example of a histogram

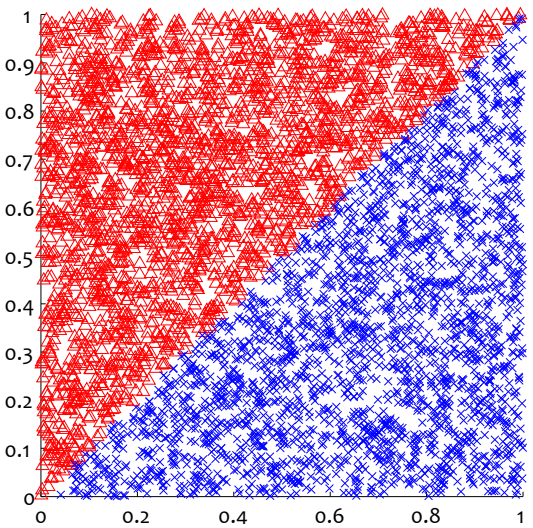


Figure 6.4: Random points separated by a diagonal

```
    end
end
```

2. Pre-generate the 5000 data points, and incorporate the condition into the plotting function.

```
x = rand(5000,1); y = rand(5000,1); % pre-generate both coordinates
plot(x(x>y), y(x>y), 'bx')
plot(x(x<=y), y(x<=y), 'r^')
```

Just for benchmarking the two versions against one another, regardless of the hardware, take the ratio: 1.603 seconds (first version) divided by 0.159 seconds (second version). The matrix calculation is over 10 times faster than the loop.

6.4 Exercises

1. Generate an array with 10 rows and 7 columns with random numbers between -1 and 5. The numbers must not be integers.

2. Generate a vector column with 30 elements containing random numbers with a normal distribution centred at 100 and with standard deviation 20.
   (Hint: To offset the data, add the desired constant. To change the standard deviation, multiply the data by the desired number. Think about the order of carrying out these operations.)

3. Plot the function $y = \sin(x^3 - 2)$ for 100 equally spaced values of x in the interval [0,2], using solid black line. Draw a random sample of 10 values of x, and display the respective (x, y) points

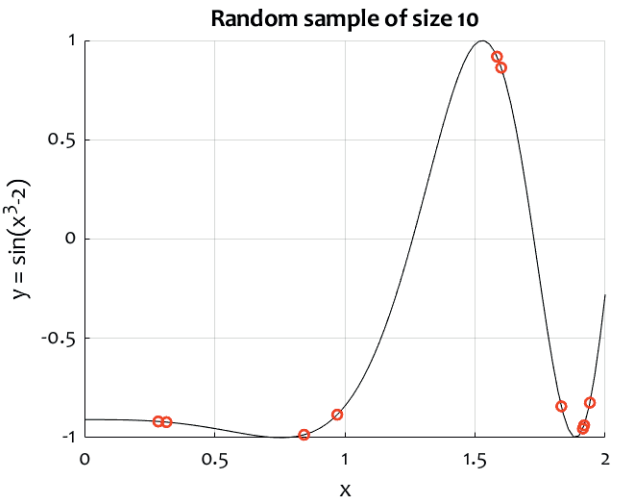


Figure 6.5: Expected output for problem 6.4.3

on the graph with marker red circle. Label the axes and add a title to the graph. The expected output is shown in Figure 6.5.

4. Write MATLAB code to do the following:

   (a) Generate a random number k between -30.4 and 12.6.

   (b) Generate an array A of size 20-by-20 of random integers in the interval $[-40, 10]$. Subsequently, replace by 0 all elements of A which are smaller than k.

   (c) Find the mean of all non-zero elements of A.

   (d) Pick a random element from A.

   (e) Visualise A using a random colour map containing exactly as many colours as there are different elements of A.

   (f) Extract 4 different random rows from A and save them in a new array B.

   (g) Find the proportion of non-zero elements of B.

   (h) Display in the Command Window the answers of (a), (c), (d) and (g) with a proper description of each one.

5. Sub-plots

   (a) Generate an array A with 200 random points in 2d, where both x and y vary from -100 to 100. Plot the points with black dots. As in sub-plot (a) in Figure 6.6.

   (b) Calculate and plot the mean of A as in sub-plot (b).

   (c) Plot lines connecting the mean of A to each point as in sub-plot(c).

(d) Plot in sub-plot (d) only those line segments from the previous question, whose length is less than 50.

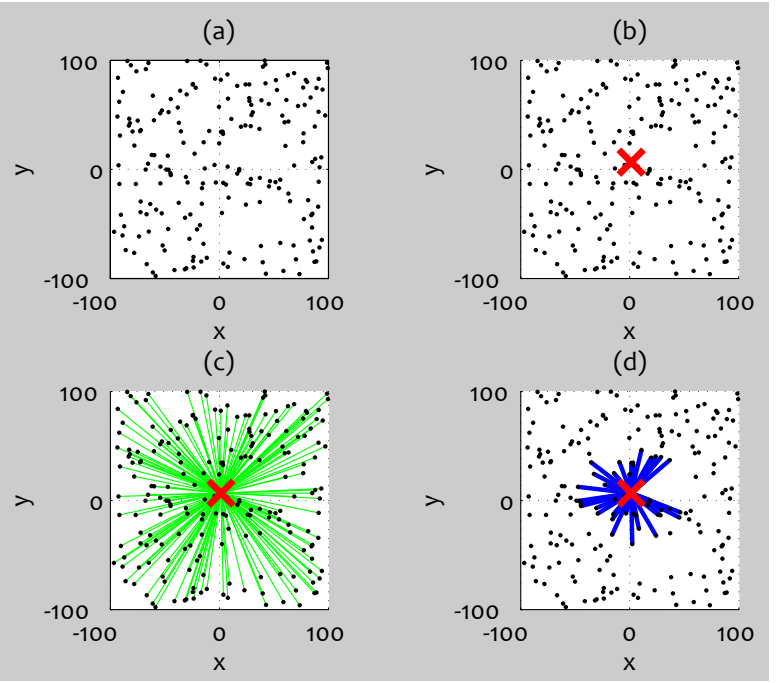


Figure 6.6: Sub-plots with random data.

6. Suppose that you are testing a slot machine. The machine has 6 types of fruit. Appearance of three of the same fruit guarantees a prize.

   (a) Generate an array of 10,000 random outcomes of the three slots of the machine.

   (b) Find the total number of winning combinations among the 10,000 outcomes.

   (c) Assume that the entry fee for each run is 1 unit of some imaginary currency. Each winning combination is awarded a prize of 10 units except for the combination of three 1s, which is awarded a prize of 50. Assuming you are the owner of the slot machine, calculate your profit after the 10,000 runs of the game.

7. Produce a game board similar to the one in Figure 6.7. The board should have 81 squares arranged in a 9×9 matrix. Nine random squares should contain smaller red squares within, and other nine random squares should contain a blue star symbol. The command 'figure' should be included in your code.

start CHALLENGE

Two Lines

Use only 2 MATLAB lines (up to 75 characters including spaces) to produce the game board.

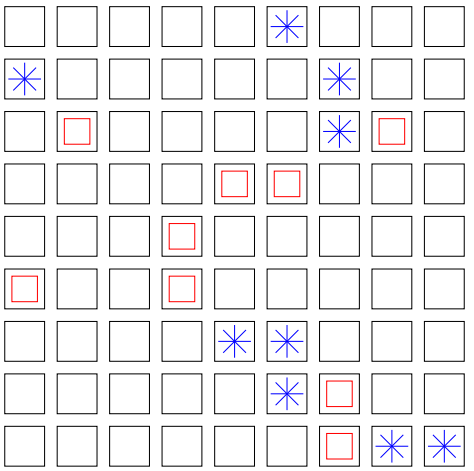


Figure 6.7: Game board for problem 6.4.7

end CHALLENGE

8. Try to reproduce Figure 6.8. The inside green disk has a radius of 0.7 units, and the white disk has a radius of 1.5 units. Do not use loops.

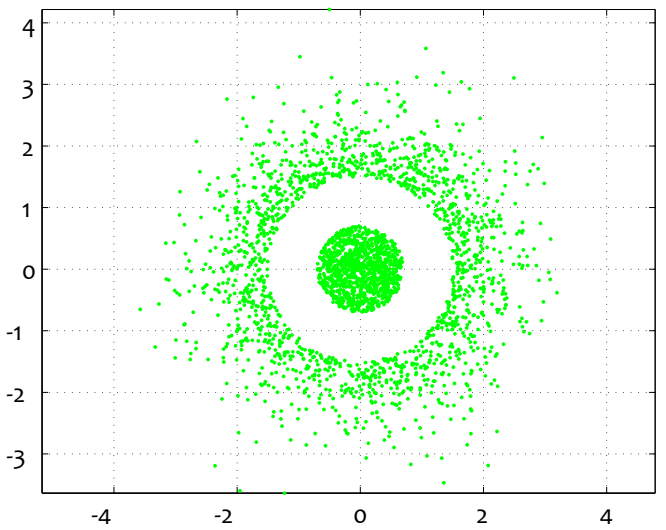


Figure 6.8: Two disks

9. Create an imitation of a noisy signal as shown in Figure 6.9. Try to reproduce the figure. Do not use loops.

10. Create an imitation of a noisy signal as shown in Figure 6.10. Try to reproduce the figure. Mark the minimum and the maximum of the signal with yellow square markers.

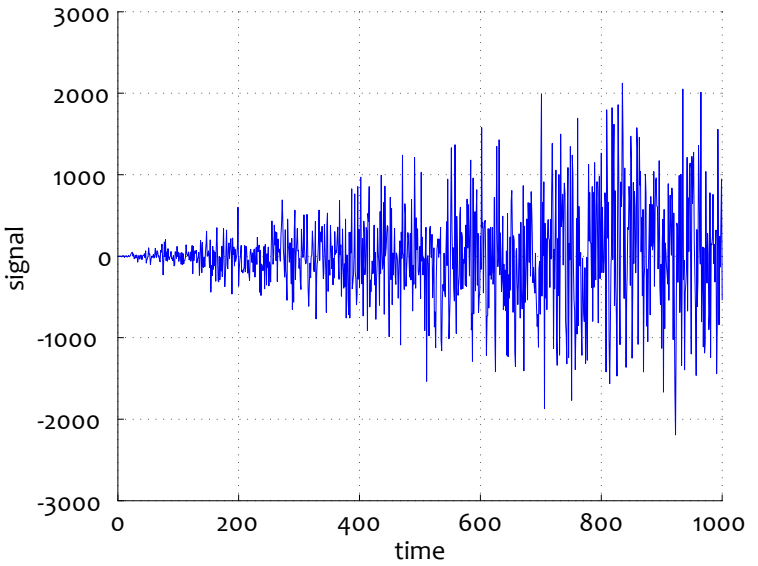


Figure 6.9: Noisy signal #1

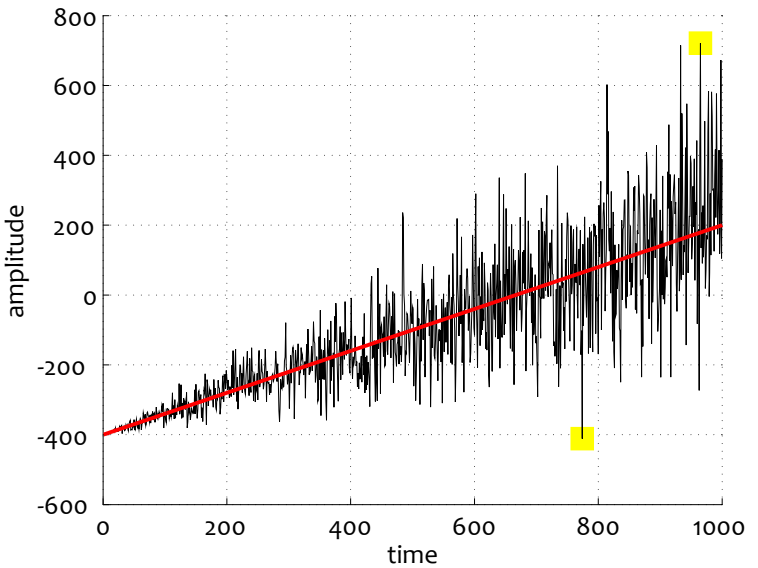


Figure 6.10: Noisy signal #2

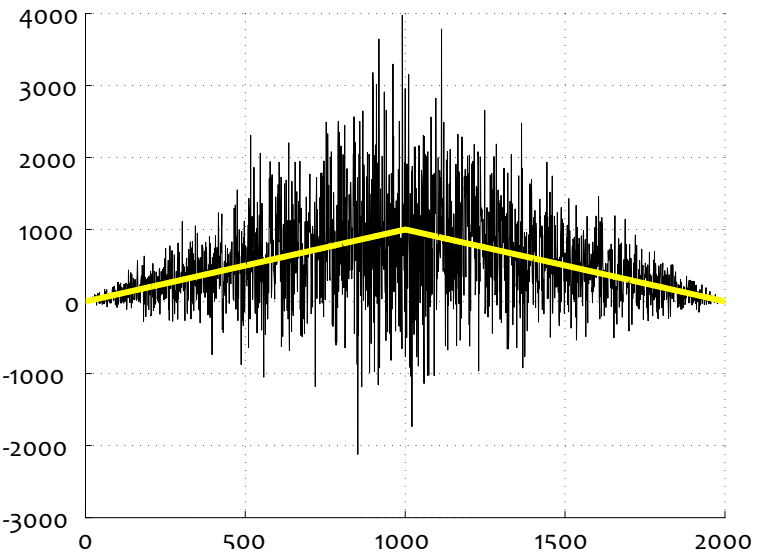


Figure 6.11: Noisy signal #3

11. Create an imitation of a noisy signal as shown in Figure 6.11. Try to reproduce the figure.

12. Craft *one* `fill` command to produce the grey shading as shown in Figure 6.12.

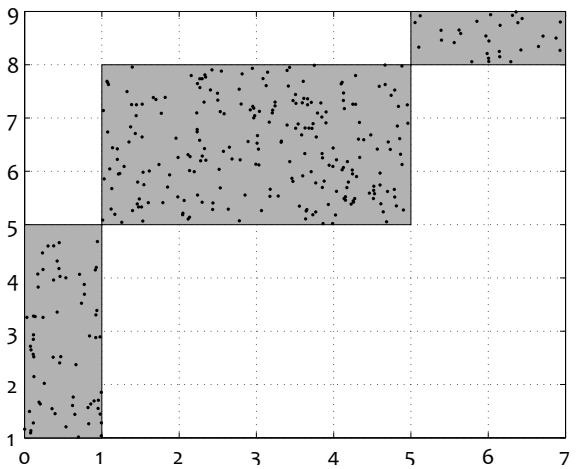


Figure 6.12: Grey regions.

Next, generate two random vectors x and y. Each must have 1000 elements, so that the (x, y) points are within the limits of the axes shown in the figure. Use *one* logical expression to determine if a point is in the grey region. Plot only the points that are in the shaded regions on your figure.

13. Generate 10 points in the unit square and plot them with black dots. Generate another random point and plot it with a red x. Your code should identify the closest black point and draw a red circle around it. One possible output is shown in Figure 6.13.

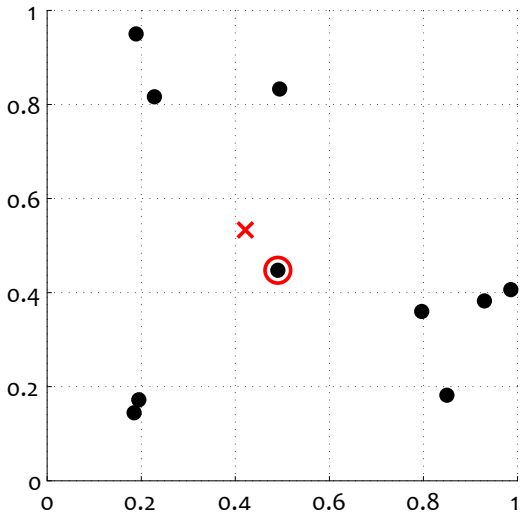


Figure 6.13: Example of the output of the nearest-point problem

14. Generate 2000 random values (not just integers) for x in the interval $[-35, 165]$ and y in the interval $[-20, 80]$. Figure 6.14 contains two 'flowers'. One is a circle centred at (30,40), with radius 30. The other is a circle centred at $(-10, 0)$, with radius 40. Each flower (seen in red) has green petals in the form of a circle at the same centre and radius 8. Reproduce the figure depicting x and y with the respective colours. Do not use loops.

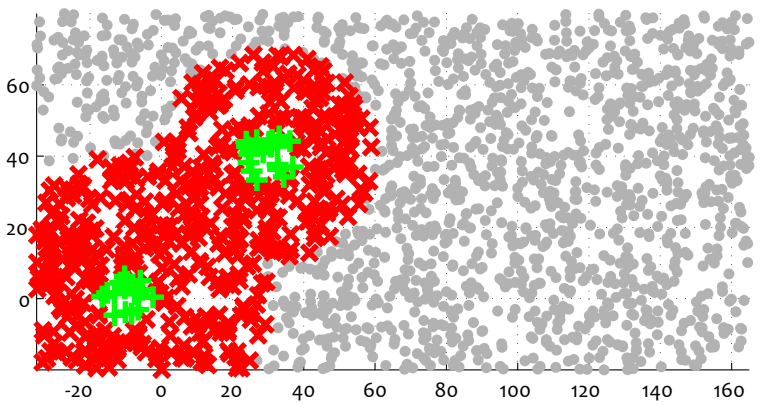


Figure 6.14: The two-flowers figure

15. Carry out random search to find a minimum of the following function:

$$f(x_1, x_2) = 2\sin\left(4x_1^2\right)\ \cos\left(6x_2^3\right)\ -\ \sqrt{|x_1|}\,(x_2 - 5)$$

using ranges: $-4 < x_1 < 4$ and $-4 < x_2 < 4$. Apply 1000 trials. Print the results in the MATLAB Command Window. Plot in a figure the best value of the function versus the number of trials. An example of the plot is shown in Figure 6.15.

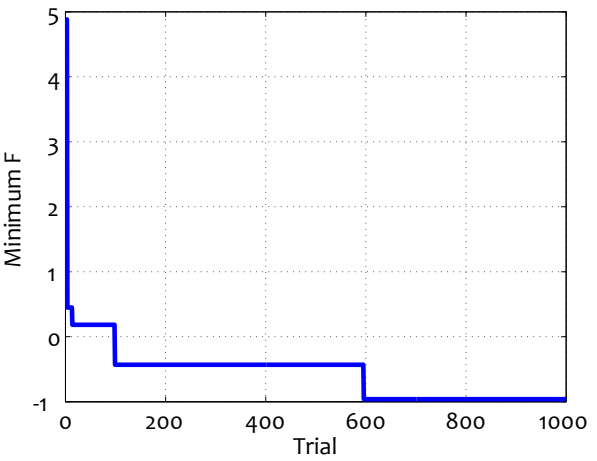


Figure 6.15: Current minimum as a function of the number of iterations

16. Two-by-Two Matrices

    (a) Using Monte Carlo simulations (a large number of random solutions), estimate what proportion of the integer-valued 2×2 matrices are singular, where the matrix entries are in the interval $[-10, 10]$. Format and display your answer in the MATLAB Command Window. (Use at least 10000 random solutions.)

    (b) Consider the integer-valued 2×2 matrices whose entries are in the interval $[-k, k]$. Calculate the proportion of singular matrices of this type for $k = 1, 2, \ldots, 50$. Plot the results on a graph and give a short comment.

17. Two Needles in a Haystack

    The following code is used by the teacher to generate the same dataset for an entire class.

```
% prepare the data file
Data = randn(1000,11);
rp = randperm(11); % choose which variables to modify
Data(:,rp(1)) = -Data(:,rp(2)) + randn(1000,1)*.05;
Data(:,rp(3)) = Data(:,rp(4)).*cos(Data(:,rp(4)))  + randn(1000,1)*.05;
clear rp;
save DataFile
```

    You will need to run the snippet to obtain `DataFile` before clearing your workspace to continue with this problem.

(a) Import 'DataFile', containing the array Data, into your workspace. Each column corresponds to a variable and each row is a data point described by the variables in the columns. Find the means of all variables and display them in a bar chart. An example of the desired output format is shown in Figure 6.16.

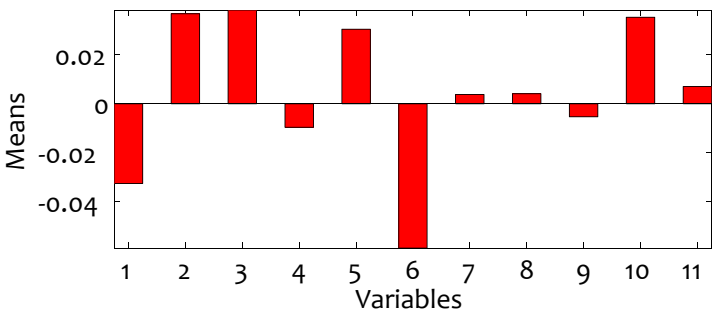


Figure 6.16: An example of the desired bar chart output for problem 6.4.17 (a)

(b) While most of the variables are random noise, there are relationships between two pairs of variables (the 'needles in the haystack'). Find a way to visualise all pairs of variables in order to discover which pairs have the relationships plotted in Figure 6.17. Show the code that you used for this visualisation. Plot the relationships you discovered as shown in the figure. Put the true numbers of the variables instead of $\#X$ and $\#Y$.

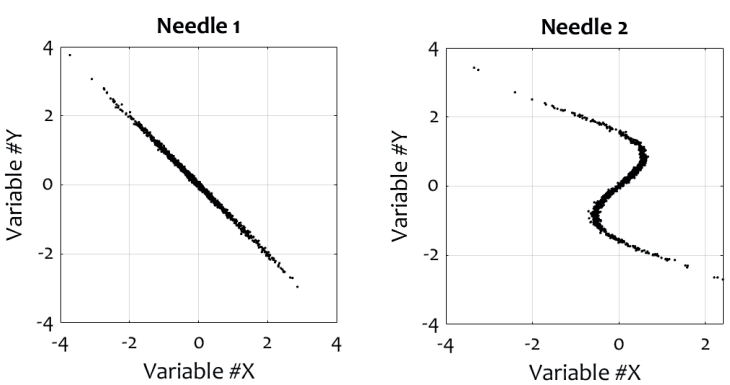


Figure 6.17: The relationship between two pairs of variables (the 'two needles in the haystack')

18. Poker Hands

(a) Write MATLAB code to draw randomly a poker hand: 5 cards out of a standard deck of 52 cards (four suits: clubs ♣, diamonds ♢, hearts ♡ and spades ♠, and 13 values for each suit: 2, 3, 4, ... 10, J, Q, K, A). Check whether the hand contains 3-of-a-kind. Keep sampling until a hand with 3-of-a-kind is generated. Display the result the Command Window. For example, the hand (10♢, 10♣, 2♢, K♠, 10♠) should be shown as `10D, 10C, 2D, KS, 10S`.

Also, print the number of hands sampled before reaching the 3-of-a-kind. Note; you should guard against a 'full house' pattern where the remaining two cards are of the same value, for example (10♢, 10♣, K♢, K♠, 10♠). This is not acceptable as a 3-of-a-kind hand.

(b) Rank a poker hand into one of these categories:-

1. high card (none of the following 8)
2. one pair
3. two pairs
4. three of a kind
5. straight (consecutive cards, mixed suits)
6. flush (same suit, any value)
7. full house (three of a kind and a pair)
8. four of a kind
9. straight flush (consecutive cards, same suit)

Your code should draw a random poker hand, display it as in part (a), and display its value in words. For example:-

```
10D, 10C, 2D, KS, 10S
three of a kind
```

19. A Welsh Village

Consider a hypothetical Welsh village with 10,000 inhabitants, of which 50% are male and 50% are female. 20% of the male inhabitants are bald. 30% of female inhabitants are blond. 37% of the inhabitants from the whole village population wear glasses. 10% of the inhabitants share the surname Jones. 5% of the female population are called Carys, and 7% of the male population are called Dafydd. (Bear in mind that the percentages are exact, not approximate figures.)

(a) Create a random matrix V that will hold the information about all the 10,000 inhabitants of the village. Each row of V represents a person, and the columns represent the information about that person from the description above.

(b) Take a random sample of 200 different villagers from V. Within that sample, find and display in the Command Window (with precision 2 decimal places) the percentage of the following:

(i) People called Carys Jones or Dafydd Jones.
(ii) Blond ladies wearing glasses.
(iii) Bald gentlemen who are not called Dafydd.

20. Estimating Areas of Intersection

(a) Pick a random centre and radius of a circle in 2d. Pick also the coordinates of the bottom left corner of a rectangle, as well as its width and height. All values should be randomly drawn integers between 1 and 10.

(b) Plot the circle and the square.

(c) Run Monte Carlo simulations to estimate the area of the intersection between the circle and the rectangle. If the circle is contained entirely within the rectangle, or the rectangle is contained entirely within the circle, the area should be calculated (not estimated).

(d) Visualise the result as in the example in Figure 6.18.

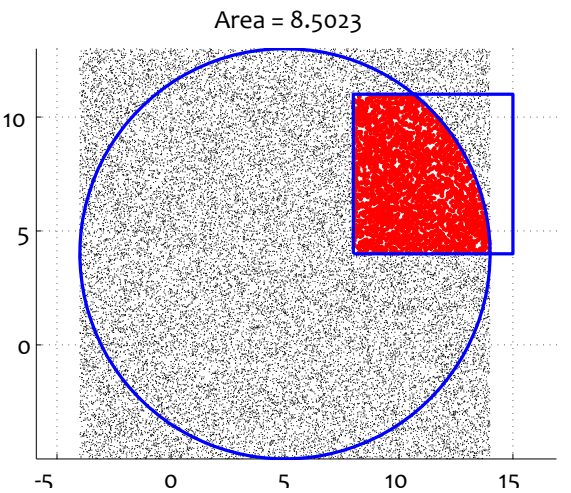


Figure 6.18: An example of the output for the area estimation problem in problem 6.4.20

21. Government Agency X is planning to dispose of radioactive waste in the form of particles at location (p, q). The pollution pattern follows a normal distribution centred at (p, q), with standard deviation s km. The agency is concerned about a small village situated m km south and n km west of the source of pollution. The zone of concern is a circle with radius k km around the village.

(a) Give values to the parameters p, q, s, m, n, and k, and plot a figure to illustrate the pollution pattern and the zone of concern.

(b) Create a function which will take the parameters as input, together with a number of released particles. The output should be the number of particles in the zone of concern.

The values of the parameters for the remaining sub-problems are:-

```
p = 10; q = 50; s = 60; m = 90; n = 80; k = 20;
```

(c) Run Monte Carlo simulations to estimate the pollution rate in the zone of concern (proportion). Note that you will have to call your function many times for this estimate to be accurate.

(d) Evaluate the pollution as the number of particles per square kilometre (PPSK) if N thousand particles were released by the source. Show a graph by varying N. Annotate the axes properly. Show a progress bar during the calculation (`waitbar`).

(e) The waste is expected to release one cloud of 15 thousand particles. Assume that the legal pollution limit is 0.04 particles per square km. The agency has an option to move the waste point north. Find (to the nearest kilometre) the southernmost possible position so that the pollution in the zone of interest does not exceed the limit.

(f) Format and print in the Command Window a short report for Government Agency X, containing your findings for the values given in (e). A few lines will suffice, for example

```
Currently chosen location of the waste point: (10,50).
Number of particles per square kilometre (PPSK): ...
Current PPSK safety limit: 0.04.
Suggested new location of the waste point: (...,...)
PPSK for the new location: ...
```

22. Largest Number of 1s

Demonstrate the operation of an evolutionary algorithm for the following problem.

- The chromosome is a binary vector of length 625.
- The fitness function is the number of 1s in the chromosome – the larger, the better.
- Start with a random population with 10 chromosomes.
- Use only mutation; set the mutation probability to 0.15.
- Run your algorithm for 20 generations.

At each new generation, plot the best chromosome in the current population using the 'spy' command. Format the chromosome as a 25×25 matrix. An ideal chromosome will have all spaces filled. The worst chromosome will be an empty square in the figure.

At the end, print out the fitness value of the best chromosome, and show the chromosome as explained above.

23. How important is the mutation probability P_m?

Take the code from the previous problem and estimate the role of the mutation probability P_m. Run it for values of P_m between 0.01 and 0.35 and plot a graph of the fitness of the best chromosome against P_m. Give a short comment on the result.

24. Simulation of virtual bugs.

Write a MATLAB script to simulate the behaviour of virtual bugs. The rules are:

- The bugs are initially randomly spread on a grid of 25×25 cells. Each grid cell receives a bug with probability 0.4.
- At each step, a bug moves to a random neighbouring cell: up, down, left or right.
- If the move happens to be outside the edge of the grid, the bug disappears.

- If more than one bug fall in the same cell, the cell destroys all of them.

Run consecutive steps until there are no bugs left on the grid. Display each iteration using the 'spy' command, and pause for 0.05 seconds to see the bugs moving. At the end of the run, print the number of steps in the Matlab Command Window.

25. The travelling salesman problem (TSP) is defined in the following way. The salesman has to visit n given cities. The order of visiting does not matter. The goal is to find an order of visiting so that the distance travelled is minimum.

    Write a function which implements a Monte Carlo simulation for the TSP. The function should take as its input argument an array of (x, y) coordinates of the n cities, and should return a permutation of the integers from 1 to n. The algorithm should check 50000 random solutions in a loop. A graph should be plotted and updated along the run, every time a better solution is found. An example of the end solution for $n = 10$ cities is shown in Figure 6.19. The title of the figure should show the iteration number (loop counter) for the solution currently displayed, as well as the tour distance. Run your function using $n = 10$ cities. The cities should be generated at random in the unit square.

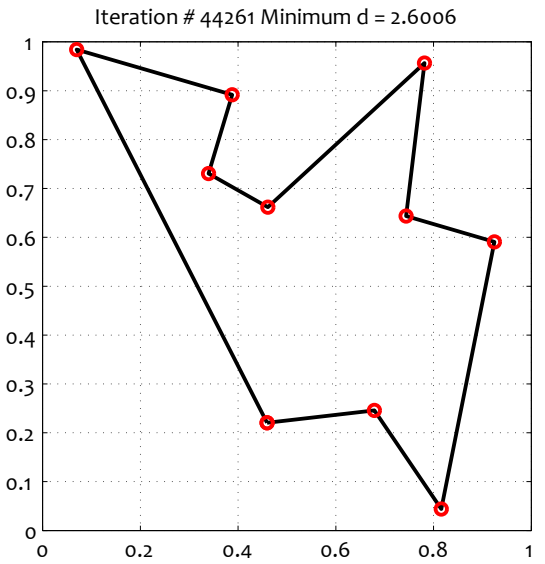


Figure 6.19: A TSP problem

26. Write a function for the TSP (see the previous question) using a 'greedy' approach. Starting with the first city, identify the nearest city and add it to the list. Identify, among the non-visited cities, the nearest one to the last city on the list. Keep growing the list until all cities are placed in it. Close the tour to calculate the tour distance. Run your function using $n = 10$ cities. The cities should be generated at random in the unit square.

27. Carry out a comparative study of the two approaches to the TSP problem using tour distance and execution time as your two criteria. Choose a format to present your results.

Chapter 7

Strings

7.1 Encoding

In most programming languages, as with MATLAB, characters are actually represented by numbers. There are numerous encoding schemes that are used in different scenarios and languages. MATLAB (being primarily American) uses the ASCII - the American Standard Code for Information Interchange. This scheme represents 256 possible character codes as numbers between 0 and 255, with each character occupying one byte. Of these, 32 are so called 'non-printable' characters as they do not produce any output on screen or when sent to a printer. There are 95 useful characters that can be entered on a western QWERTY-style keyboard. Table 7.1 shows a selection of these 95 characters - Latin numbers and letters only.

Table 7.1: ACSII characters and their decimal codes

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
	!	"	#	$	%	&	'	(	)	*	+	,	−	.	/
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	
p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

As characters are simply numbers, a matrix can contain a string. Each element will represent a single character, and will be an integer between 0 and 255. Your other choice when dealing with multiple strings would be to use a cell array (see Section 2.6).

7.2 Useful String Functions

MATLAB has string handling functions, just like any other language. As with most languages the function names (in the most part) start with *str*. Table 7.2 contains a list of the most common and useful string handling functions in the MATLAB language.

Table 7.2: Common String Functions.

Command	What it does
`strcmp(a,b)`	Compares **a** and **b**, returning 1 if identical.
`strfind{a,b}`	Finds occurrences of **b** in **a**, returning a vector of the start positions.
`strrep(a,b,c)`	Forms a new string, replacing all instances of **b**, in **a** with **c**.
`strtok(a)`	Returns the first and the second parts of **a** split by a token.
`strtrim(a)`	Returns a copy of **a** with the leading and the trailing whitespace removed.
`isstr(a)`	Returns 1 if **a** is a string, and 0 otherwise.
`str2num(a)`	Returns the number represented by the string of digits **a**.
`input(p, 's')`	Inputs a string from the Command Window where **p** is a string prompt for the user.
`sprintf(f,d)`	Constructs a formatted string.
`disp(s)`	Output **s** in the Command Window.

7.3 Examples

7.3.1 Imaginary Planet Names

Suppose you are writing a science fiction novel, and need a collection of 10 exotic planet names. Each name should be made of 4 to 7 letters, where the string alternates between vowels and consonants. Each run of the code below will generate a random collection of names, for example: Azine, Sunaru, Isunot, Tuso, Medutas, Abafora, Darewoz, Amiza, Tanata and Danem.

```
cnsnt = 'rtnmktfwbnptrgmpxndsxzzrtzsd'; % consonants to be used
for i = 1:10
    name_length = randi([4 7]);
    vx = vwls(randperm(numel(vwls))); % choose the vowels
    cx = cnsnt(randperm(numel(cnsnt))); % choose the consonants
    planet = '';
    if rand > 0.5 % start with a vowel
        planet(1:2:name_length) = vx(1:2:name_length);
```

```
        planet(2:2:name_length) = cx(2:2:name_length);
    else
        planet(1:2:name_length) = cx(1:2:name_length);
        planet(2:2:name_length) = vx(2:2:name_length);
    end
    planet(1) = upper(planet(1)); % capitalise the first letter
    fprintf('%s\n',planet);
end
```

7.3.2 String Formatting

Print the Receipt. Suppose that you have a small business and you sell six different products. Choose your products and their prices within the range of 20p to £25.00 (these could be completely fictitious). Your shop has 4 employees, one of whom will be at the till at the time of purchase. Your task is to write the code to prepare a receipt for a fictitious transaction as explained below.

There is a customer at the till. They want to purchase 3 random products with specific quantities for each. To prepare your receipt:

1. Select randomly 3 products from your list. For each product choose a random quantity between 1 and 9.
2. Calculate the total cost.
3. Choose randomly the staff member to complete the transaction.
4. Suppose that the price includes 20% VAT. Calculate the amount of VAT included in the price.
5. Prepare the receipt as text in the MATLAB Command Window. Use the current date and time (check the command `datestr(now,0)`).

Your code should output the receipt in the format shown in Figure 7.1. There should be 60 symbols across. Choose your own shop name.

Solution.

```
    'Tips on Life','Album of Cat Photos','Ice cream','Diamond Necklace'};
prices = [3.50, 2.48, 12.40, 10.90, 5.63, 11.50];
staff = {'Henrieta','Esmeralda','Katrina','Johann'};

% Entries
rp = randperm(6); % first 3 products were purchased
Quantity = ceil(rand(1,3)*9); % how many of which
ItemPrices = Quantity .* prices(rp(1:3));

% Frame
Rec = repmat(' ',16,60); % empty template
[Rec(1,:),Rec(end,:)] = deal('-'); % side frame
```

```
+-----------------------------------------------------------+
|                                                           |
| 06-Dec-2019 21:24:17                Grandma's Little Shop |
|                                                           |
|                                                           |
|       Album of Cat Photos  (5) x 10.90 =  £  54.50        |
|          Diamond Necklace  (9) x 11.50 =  £ 103.50        |
|                 Ice cream  (8) x  5.63 =  £  45.04        |
|                            --------------------------     |
|              Total to pay                 £ 203.04        |
|                       VAT                 £  33.84        |
|                                                           |
|                                                           |
| Thank you! You have been served by Henrieta               |
|                                                           |
+-----------------------------------------------------------+
```

Figure 7.1: The print layout for the receipt

```
[Rec(:,1),Rec(:,end)] = deal('|'); % side frame
[Rec(1,1),Rec(end,1),Rec(1,end),Rec(end,end)] = deal('+'); % corners

% Fill in
Rec(3,3:length(datestr(now,0))+2) = datestr(now,0); % today's date
shop_name = 'Grandma''s Little Shop';
Rec(3,end-length(shop_name)-1:end-2) = shop_name;
for i = 1:3
    S = sprintf('%25s  (%1i) x %5.2f =  Â£ %6.2f',products{rp(i)},...
        Quantity(i),prices(rp(i)),ItemPrices(i));
    Rec(i+5,3:length(S)+2) = S;
end
Rec(9,29:54) = '-'; % line under the list
Rec(10,16:27) =  'Total to pay';
Rec(10,43:52) =  sprintf('  Â£ %6.2f',sum(ItemPrices)); % total price
Rec(11,25:27) = 'VAT';
Rec(11,43:52) = sprintf('  Â£ %6.2f',sum(ItemPrices)/6); % VAT
bye = sprintf('Thank you! You have been served by %s',...
    staff{ceil(rand*4)});
Rec(14,3:length(bye)+2) = bye;
disp(Rec)
```

7.4 Exercises

1. Coded Messages

   (a) Create a random coded message as a 10-by-50 matrix with integers corresponding to ASCII codes. See Table 7.1 for the required codes. Using the command `char`, display the message.

(b) Find the number of occurrences of capital letters. Replace all such occurrences with the symbol # (The ASCII code for # is 35. To find the ASCII code for a character, type `double(<the_character>)`.

(c) Turn your original message into a table with k columns, as shown below for $k = 5$. Notice that the table has the same size as the original message (10-by-50). The top and bottom rows are replaced with dashes, and the respective columns are replaced by vertical bars. (You should be able to change the value of k in the code and display a table with the desired number of columns.)

```
+---------+---------+--------+---------+---------+
|YJDxLsTbY|kyTXDLBuq|zRCEASUM|QEYZAFlGo|hDFhuccfE|
|zBkRfAVvM|qBhgYtPGr|jGvvsPHC|PzpSfMgdn|XOIYmDKdN|
|eGWdsvmKM|ixVwBSajA|udWKvIwH|XUqUMwizt|zaqUEBoIB|
|qOjwjmtJO|LJZWDalLi|EIbJtsFJ|fKtwFueRI|dfaLbblDK|
|ZJqIPOsnh|xLzryivOD|PYewnPoO|QAqbHsJto|vqvrudsTm|
|ZlZToSxDm|RcmFJGsey|AgaWoZFB|qAVKxlRUZ|trOjsrZib|
|zGsWgtavw|TeqyJhyQE|yQReYONR|KWJgiKwKI|WCTNIhKUo|
|xqxqmopTB|ZhUUSlovO|ACxarMmR|bZonYojEx|zCJxRmXAq|
+---------+---------+--------+---------+---------+
```

2. Extract the first e-mail address from a string. Check whether the e-mail is from the UK (ending with '.uk') and display the extracted address in the Command Window, indicating 'UK' or 'NON-UK'. Demonstrate the work of your code on strings of your choice.

3. Construct arrays containing (i) quantifiers (e.g., all, few, many, several, some, every, each, any, no, etc.), (ii) nouns or expressions (animated), (iii) verbs and (iv) another array with nouns or expressions. Draw randomly one word from each array to construct a funny random proverb. A few examples:

```
No students hide from the relativity theory.
Some cats eat football.
All politicians adore MATLAB.
Most politicians are scared of the French.
Many zebras adore the French.
All zebras look like Adele's songs.
```

4. Write a primitive Chat Bot. The conversation will start with:

```
>> And you were saying? ...
```

displayed in the Command Window. It then progresses taking a sentence from the user and displaying the last word in a question:

```
>> Really, <last word>?
```

For example, if the user inputs 'It is snowing today.' your program should print 'Really, today?'

5. Write MATLAB code which will do the following. Ask the user to input a short sentence. Replace the spaces with a dash and display the text as shown in the example below.

   Suppose that the text is 'Joey is a super cat!' Your output should be:

```
                !
               t!              ...
              at!                                  a-super-cat!
             cat!                                 -a-super-cat!
            -cat!                                s-a-super-cat!
           r-cat!                               is-a-super-cat!
          er-cat!                              -is-a-super-cat!
         per-cat!                             y-is-a-super-cat!
        uper-cat!                            ey-is-a-super-cat!
       super-cat!                           oey-is-a-super-cat!
      -super-cat!                          Joey-is-a-super-cat!
```

6. Manipulating Strings

   (a) Enter the following text into a variable.

   > *Once upon a time, a very long time ago now, about last Friday, Winnie-the-Pooh lived in a forest all by himself under the name of Sanders. "What does 'under the name' mean?" asked Christopher Robin. "It means he had the name over the door in gold letters, and lived under it."*

   Make sure that your code does not exceed the width of the MATLAB editor's page (75 characters).

   (b) Use the names of three celebrities of your choice to replace in the string the three names: Winnie-the-Pooh, Sanders and Christopher Robin.

   (c) Find the number of characters, with and without counting the spaces. Display your answer in the following format in the Command Window:

```
The string contains XXX characters if counting the spaces
and XXX characters without the spaces.
```

   (d) Find the total number of words in the string (repeated or not). Assume that any two words are separated by a space. Display your answer in the Command Window.

7. Count the number of words in a string, excluding 'the', 'a' and 'and', regardless of capitalisation. Demonstrate the working of your code with an example string.

8. Create an Anagram game using MATLAB. You will need to create a cell array with the names of the countries in Europe shown in Table 7.3. Next, randomly choose a country, mix-up the letters to make an anagram, display it in capital letters, and ask the user to recognise the country. The

user is allowed 3 attempts. If the user inputs the correct name (at any of the three attempts), display a message of congratulation and stop. If all three attempts are unsuccessful, display an appropriate message and stop.

Table 7.3: Countries of Europe

Albania	Hungary	Portugal
Andorra	Iceland	Republic of Macedonia
Austria	Ireland	Romania
Belarus	Italy	Russia
Belgium	Kosovo	San Marino
Bosnia and Herzegovina	Latvia	Serbia
Bulgaria	Liechtenstein	Slovakia
Croatia	Lithuania	Slovenia
Cyprus	Luxembourg	Spain
Czech Republic	Malta	Sweden
Denmark	Moldova	Switzerland
Estonia	Monaco	Turkey
Finland	Montenegro	Ukraine
France	Netherlands	United Kingdom
Germany	Norway	Vatican City
Greece	Poland	

9. Write MATLAB code to print, in the Command Window, a decorated Christmas tree as shown below. The candles (i) and the balls (O) should be at random places. Notice the star (*) at the top and the stump (I) at the bottom of the tree. Aim to write the shortest possible code.

```
          *
         ^iO
        i^^^^
       ^^^i^^O
      ^^^^^^^O^
     ^^^^^^^^^i^
    ^^O^^^^^^^^^i
   ^^^O^^^^^^^^^^
  ^^^^O^^^i^^^^^iiO
          I
```

10. Write a function which will take a matrix *A* as its input argument, and will print in the Command Window a LaTeX script for this matrix. In LaTeX syntax. The columns should be right-aligned and the matrix should be given in large square brackets. For example, the matrix

```
A =
     7     9    -4    10
     9     3     1    -6
    -7    -8    10    10
```

should be coded in LaTeX as the script below. When placed in the equation environment (opened with '\[' and ended with '\]'), the matrix should look like the one shown to the right.

```
\left[
\begin{array}{rrrr}
7&9&-4&10\\
9&3&1&-6\\
-7&-8&10&10\\
\end{array}
\right]
```

$$\left[\begin{array}{rrrr} 7 & 9 & -4 & 10 \\ 9 & 3 & 1 & -6 \\ -7 & -8 & 10 & 10 \end{array}\right]$$

11. ASCII Art #1

Write a function which takes a text string and a template as input, and places the words in the shape of the template. The template should be a binary matrix with ones where the ascii symbols should be. The function should return a string. An example is shown below:

```
                   If i    t w
                 asn't fo  r t
               he coffee, I'd
             have no identifia
           ble personality what
         sover.If it wasn't for t
       he coffee, I'd have no ident
     ifiable personality whatsover.If
        it wasn't for the coffee, I
       'd hav      e no identifiabl
       e pers      onality whatsove
       r.If i      t wasn't for the
        coffe      e, I'd have no i
       dentifiable pers        onal
       ity whatsover.If         it
       wasn't for the c        offe
       e, I'd have no i        dent
       ifiable personal        ity
       whatsover.If it wasn't for t
  he coffee, I'd have no identifiable personality
whatsover.If it wasn't for the coffee, I'd hav
```

Calling the function using:

```
[x,y] = meshgrid(1:18,1:18);
a = ascii_art_form('Christmas forever!', [fliplr(x < y) x < y]);
disp(a)
```

should print in the command window the pattern shown below.

```
                Ch
               rist
              mas fo
             rever!Ch
            ristmas fo
           rever!Christ
          mas forever!Ch
         ristmas forever!
        Christmas forever!
       Christmas forever!Ch
      ristmas forever!Christ
     mas forever!Christmas fo
    rever!Christmas forever!Ch
   ristmas forever!Christmas fo
  rever!Christmas forever!Christ
 mas forever!Christmas forever!Ch
ristmas forever!Christmas forever!
```

Chapter 8

Images

8.1 Types of Image Representations

MATLAB supports the image representations detailed in the following sections. The core version has a limited set of image commands. To check whether you have a license for the Image Processing Toolbox, type in the Command Window:

```
license('test','image_toolbox').
```

An answer of '1' means 'yes, you have a license'.

8.1.1 Binary Images

A binary image in MATLAB is a matrix containing 0s and 1s. An example is shown in Figure 8.1. Zeros indicate black and 1s indicate white. Command `imshow()` will show the black-and-white image in the currently open figure. An alternative way to show the non-zero elements of a matrix is to use the command `spy()`. This command shows the matrix on a pair of coordinate axes. The non-zero elements are plotted with blue stars. The total number of such elements is shown as the label of the x-axis as illustrated in Figure 8.1.

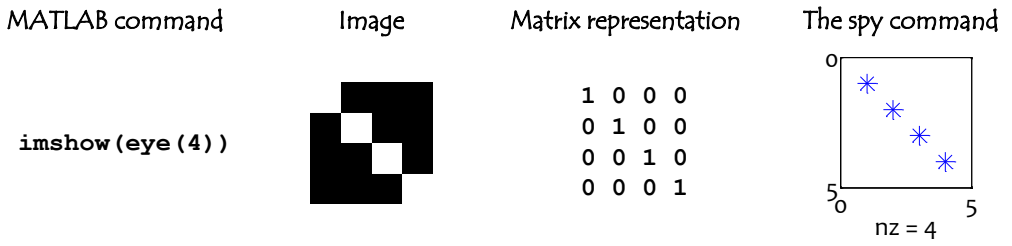


Figure 8.1: Binary image representations in MATLAB.

8.1.2 RGB Images

An RGB (red-green-blue) image is stored as three matrices (colour planes) of the same size $M \times N$, where M is the number of rows of pixels and N is the number of columns of pixels. Hence, an RGB image A should be addressed with three indices $A(i, j, k)$, where $i \in \{1, \ldots M\}$ is the row, $j \in \{1, \ldots N\}$ is the column of the pixel, and $k \in \{1, 2, 3\}$ is the colour plane. The colour of pixel at i, j is determined by

the combination of the red intensity $A(i, j, 1)$, green intensity $A(i, j, 2)$ and blue intensity $A(i, j, 3)$. Each value is stored in 8 bits, as unsigned integer, format `uint8`. Value (0,0,0) for $(A(i, j, 1), A(i, j, 2), A(i, j, 3))$ makes pixel (i, j) black, and value (255,255,255), makes it white. Equal values in the three planes will make the pixel grey, with intensity determined by that value. An example of an RGB image and its MATLAB representation are shown in Figure 8.2.

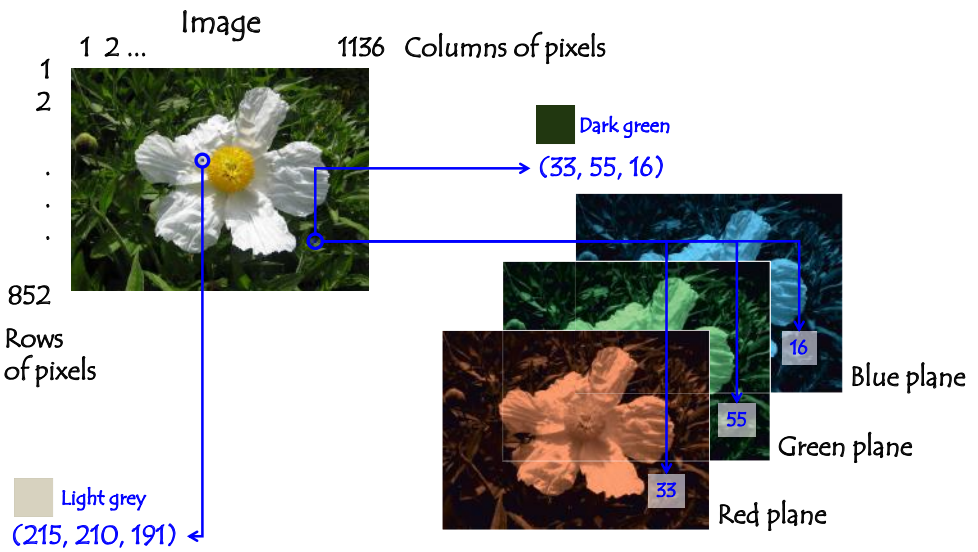


Figure 8.2: An RGB image representation in MATLAB using uint8 format.

8.1.3 Grey Intensity Images

A grey intensity image is represented as a matrix A of size $M \times N$. Again, M is the number of rows of pixels and N is the number of columns of pixels. Using the `uint8` format, $A(i, j)$ takes integer values between 0 (black) and 255 (white), specifying the grey level intensity of the pixel in row i and column j. Each of the three planes of an RGB image is an intensity image itself.

An example of constructing a grey intensity image is shown in Figure 8.3.

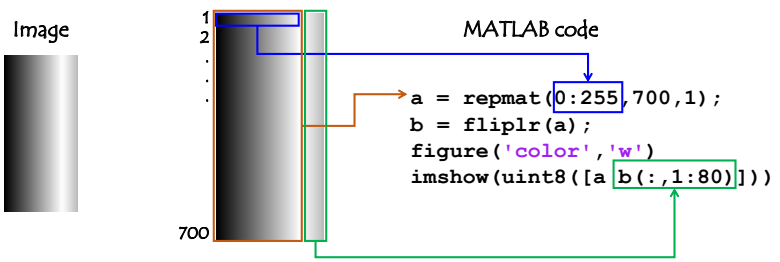


Figure 8.3: Construction of a grey intensity image.

8.1.4 Indexed Images

An indexed image is a matrix A of size $M \times N$, accompanied by a matrix C of size $k \times 3$, called 'the colour map'. Each row in the colour map matrix defines a colour. The values are between 0 and 1. White is encoded as [1,1,1], and black, as [0,0,0]. All colours can be represented as combinations of three floating point numbers between 0 and 1. For example, [0.5,0,0.5] is purple, and [0.4,0.1,0] is brown. The entries in A are taken to be row index of C. An example of indexed image is shown in Figure 8.4.

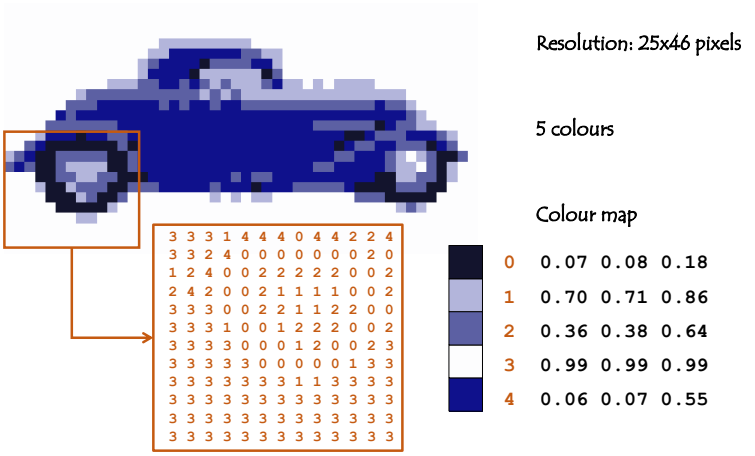


Figure 8.4: Examples of an indexed image.

8.2 Useful Functions

To extract the three planes from an RGB image A, use:

```
R = A(:,:,1); % red plane
G = A(:,:,2); % green plane
B = A(:,:,3); % blue plane
```

To (re-)assemble an image from three planes, R, G, and B, use:

```
A = cat(3,R,G,B);
```

This command will concatenate the three planes on the third dimension.

Sometimes it is necessary to store the coordinates of the pixels in an image. Consider the following command

```
[x,y] = meshgrid(1:5,1:3);
```

The output are two arrays of size 3×5 containing coordinates:

```
x =
     1     2     3     4     5
     1     2     3     4     5
```

```
     1     2     3     4     5

y =
     1     1     1     1     1
     2     2     2     2     2
     3     3     3     3     3
```

Notice that the y coordinate starts from top and increases with the row index. This is the ij-coordinate system available in MATLAB. To plot in a figure, with this system, set the axes by `axis ij`.

Table 8.1 contains useful commands and functions for handling images in MATLAB.

Table 8.1: Common Image Functions.

Command	What does it do?
`imread(i)`	Loads an image into a matrix.
`imshow(A)`	Displays an image matrix in a figure.
`imagesc(A)`	Scales a matrix into an image and displays the image.
`colormap(m)`	Sets the active colormap.
`rgb2gray(A)`	Converts an RGB image into a grayscale image.

8.3 Examples

8.3.1 Image Manipulation

Load an RGB image and display it so that the top diagonal half is grey, and the bottom part is unchanged. An example is shown in Figure 8.5.

Figure 8.5: An example of a half-grey image.

One potential approach for solving this problem is shown in Figure 8.6. The code is shown below.

```
A = imread('flower.jpg'); % load the RGB image in matrix A
B = rgb2gray(A); % convert to grey
s = size(B);
```

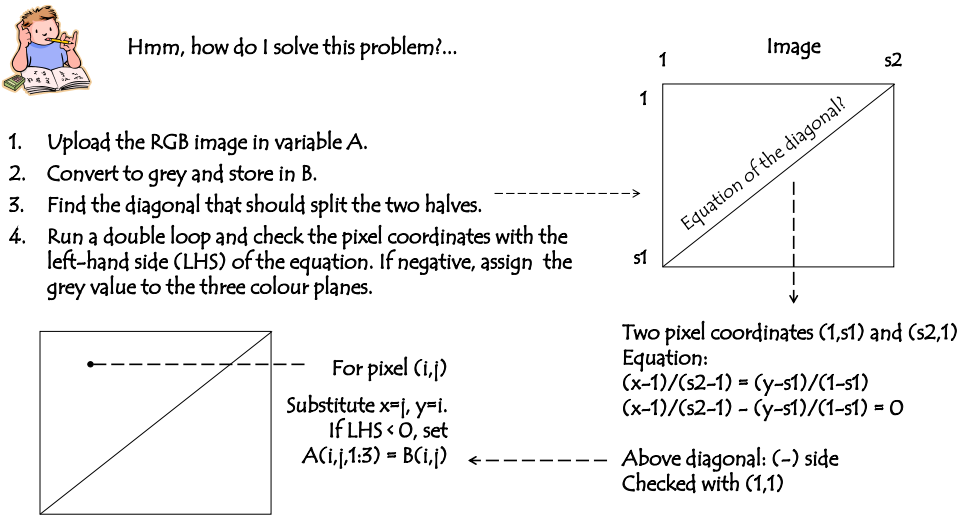


Figure 8.6: A possible approach for the solution of the half-grey-image problem.

```
for i = 1:s(1)
    for j = 1:s(2)
        if (j-1)/(s(2)-1) - (i-s(1))/(1-s(1)) < 0; % top half
            A(i,j,:) = B(i,j);
        end
    end
end
figure,imshow(A)
```

While this approach works; it is slow, and not in the spirit of the language. Instead of the double loop, we can create a mask which will have values TRUE for the top diagonal half. Then we will replace the top diagonal halves of the RGB planes of A with the the corresponding values in B, and finally re-assemble A. The code for this version is shown below.

```
A = imread('flower.jpg'); % load the RGB image in matrix A
B = rgb2gray(A); % convert to grey
s = size(B);
[x,y] = meshgrid(1:s(2),1:s(1)); % x-y coordinates for all pixels
mask = (x-1)/(s(2)-1) - (y-s(1))/(1-s(1)) < 0; % top half
r = A(:,:,1); r(mask) = B(mask); % red plane
g = A(:,:,2); g(mask) = B(mask); % green plane
b = A(:,:,3); b(mask) = B(mask); % blue plane
figure,imshow(cat(3,r,g,b)) % open a figure and show concatenated image
```

8.3.2 Tone ASCII Art

ASCII art can be created using the tone of the image. The grey levels are matched to characters. Darker characters are, for example, '@' and '#', and the lightest are '.' and the blank space.

To create an ASCII version of a grey image, rescale it to a desired resolution, and convert it to index image using `gray2ind` command. The number of shades in the colour map should be the same as the number of symbols used to represent grey values. For example, you may wish to use the following set of characters:

```
S = '#n*:. ';
```

An example using this character set is shown in Figure 8.7. To achieve a good result, the background of the original image should be removed. The colours should be preferably in patches. Good candidates for ASCII art are cartoon images.

Figure 8.7: An example of a tone ASCII art.

The code is shown below:

```
I = imread('Parrot4.png'); figure, imshow(I) % choose an RGB image
A = rgb2gray(I); figure, imshow(A)            % convert to grey
B = imresize(A,[60 120]);                     % resize (tune by hand for now)
S = '#n*:. ';                                 % character string from dark to light
C = gray2ind(B,length(S));                    % convert to index image
S(C+1)                                        % display the ASCII in the Command Window
```

Notice the particularly elegant way to construct the ASCII output, `S(C+1)`. The set of characters S is indexed with the index image values. These values are meant to be entries in the colour map, also sorted from dark to light. The output is shaped as the index C.

The indexed image, however, starts the counting from 0, while the array with the characters S must be addressed, according to the MATLAB rules, from 1. Therefore we add 1 to C when using it as index.

8.4 Exercises

1. Ask the user for a number between 1 and 4. Depending on the entered number, create and display a matrix of a random colour, where the respective quadrant has a different random colour. Examples of the four outputs are shown in Figure 8.8. Use the switch-case operator.

Figure 8.8: An example of possible outputs for quadrants 1-4

2. Reproduce Figure 8.9 by creating manually an indexed image and setting the respective colour map.

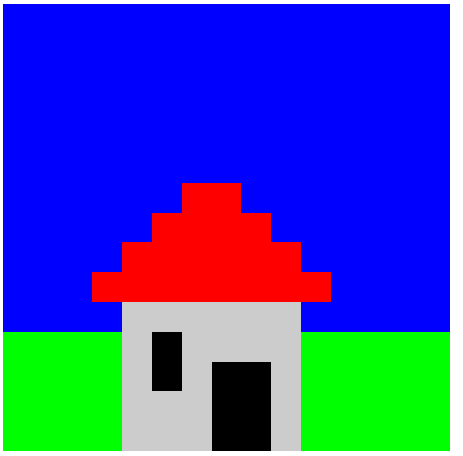


Figure 8.9: An example of indexed image

3. Generate a matrix and colour it so that it resembles the tartan pattern in Figure 8.10 or another similar pattern.

4. Load a JPEG image and plot the histograms of the red, green and blue panes of the image as shown in Figure 8.11. Note; the histograms should appear on one figure.

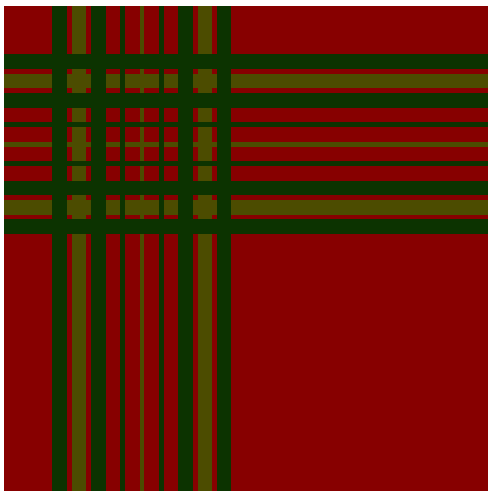
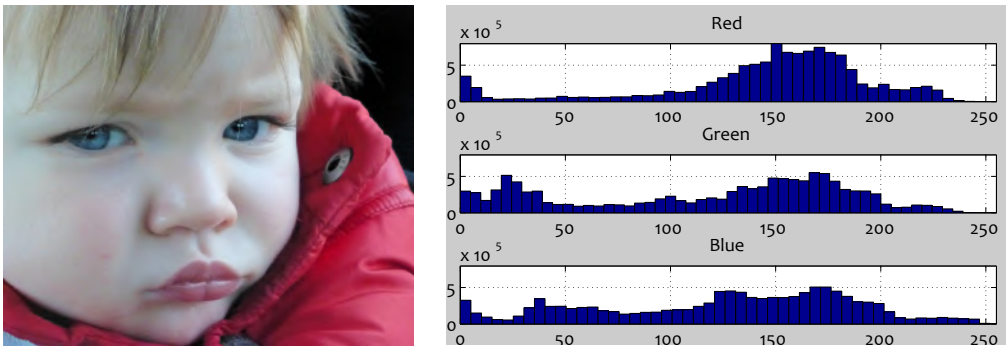


Figure 8.10: An example of a tartan-like pattern



(a) (b)

Figure 8.11: The original image (a), and the histograms of the three panes (b)

5. Load a JPEG image and convert it to an indexed image with 5 colours. Create a random colour map for the new image. Show the original and the new image. An example is shown in Figure 8.12.

6. Load a JPEG image and reduce its intensity to make it into a watermark image as demonstrated in Figure 8.13. Do not use the 'brighten' command; manipulate the image with your own code.

7. Load a JPEG image of your choice. Convert it to grey and calculate the mean and the standard deviation of the grey level intensity. Display a new image where all pixels within one standard deviation from the mean are coloured in red, and the remaining pixels stay unchanged.

   An example of an original and the manipulated image is shown in Figure 8.14.

8. Write a function that will take an RGB image and a character, which can be only R, G or B, as input arguments. The output of the function should be an RGB image of the same size as the input image, where the indicated panel (red, green or blue) is replaced by a random matrix. Demonstrate the work of your function by writing a script, calling the function with each of the

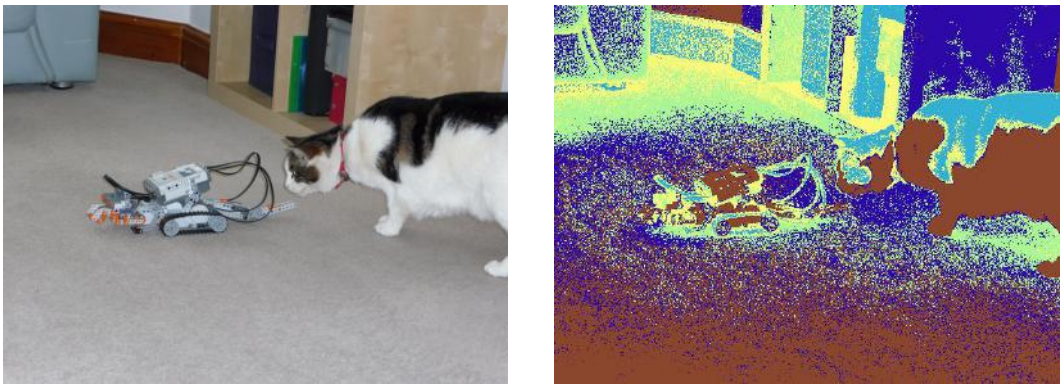


Figure 8.12: The original image and the indexed image with 5 random colours

Figure 8.13: The original image and the watermark image

Figure 8.14: The original image and the manipulated image.

three character values. Organise the output into a 3-by-1 montage and show it in a new figure. An example is shown in Figure 8.15.

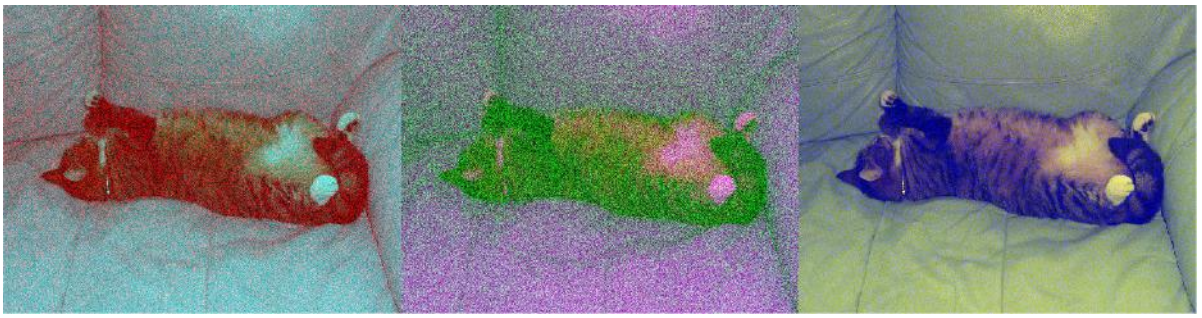


Figure 8.15: A montage with random red, green and blue panels

9. Write a function that will load an image chosen by the user and create an 'old movie' effect: tinted, faded, scratched and torn at the bottom. An example of the original image and the desired effect is shown in Figure 8.16.

Figure 8.16: An example of the 'old-movie' effect

10. Take a grey image and inset 6 progressively smaller versions of it into the top left corner, as shown in Figure 8.17. Each subsequent image should be half of the size of the previous image in both dimensions.

11. Construct and display the image in the left plot of Figure 8.18, containing red, green and blue panels where the colour appears gradually from left to right, starting with black.

    Next, add three more panels combining the RGB colours as in the right plot of Figure 8.18. The colours in the bottom row should be approximately brown, purple and tobacco.

12. Take a JPEG image and tint the four quadrants with transparent overlays as shown in Figure 8.19.

13. Frame Factory

    (a) Create a function which frames an image. The input arguments are the image, a proportion p that defines the frame size, and the frame colour. The frame colour should be given as a vector of three numbers between 0 and 1. The proportion for the frame width should be

Figure 8.17: A grey image with progressively smaller copies inset within

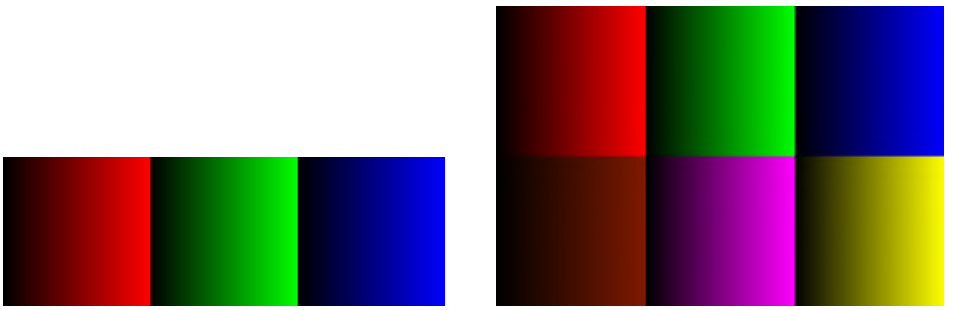


Figure 8.18: Three colour panels

Figure 8.19: Four transparent colours

taken from the smaller of the two dimensions of the image. The frame should be *inside* the image. Examples of framed images are shown in Figure 8.20. Demonstrate the output of your function in a similar way with three different sets of parameters.

(a) frame 20%
[0.8 0.3 0.2]

(b) frame 10%
[0.4 0.7 1.0]

(c) frame 3%
[0.5 0.2 0.2]

Figure 8.20: Frame Factory

(b) Use your function in a loop to create a montage as shown in Figure 8.21. The frame width and the frame colour should be random. The frame width should be no larger than 30%.

Figure 8.21: A montage of framed images

14. Load a JPEG image and draw a grid with 10 rows and 10 columns of cells on it, as shown in Figure 8.22. The grid lines should be embedded in the image, and not merely plotted on the same axes. The width of the lines should be chosen in such a way that the lines are visible. Also, make your code re-usable so that it can work on any image you upload. (This means that there should be no hard coded constants in your function/script.)

15. Play a game with your friends. Encrypt a colour image of size $[M, N, 3]$ using a random permutation of the integers from 1 to $M \times N \times 3$. Save the encrypted image in a mat file, together with M, N and the permutation used. Challenge your friends to decrypt the image from the mat file. You can run a contest to find the first person to show the correct original image.

16. Construct the function `shuffle_image` that will take an RGB image and two integers, M and N. The function should split the image into M rows and N columns of 'tiles'. It should return an

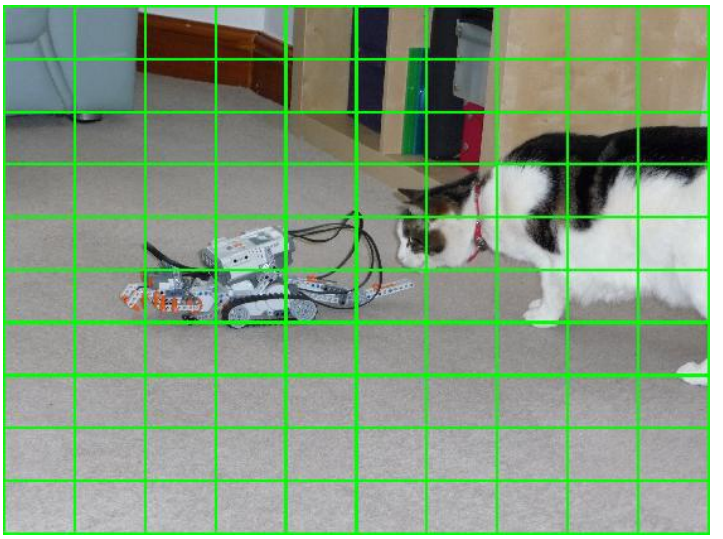


Figure 8.22: A 10-by-10 grid imposed on an image

image of the size of the original input but with shuffled tiles. An example of the original image and the shuffled image, for $M = 4$ and $N = 5$ is shown in Figure 8.23.

Note: If needed, make the image sizes multiples of M and N, respectively, by losing a small number of bottom rows and right-hand side columns of pixels.

Figure 8.23: The original image and the 4×5 shuffled image.

17. Create a function named `image_blocks`. The input arguments are: x, an RGB image; N, number of rows of blocks; M, number of columns of blocks; and p, a parameter to chose between mean/median/mode.

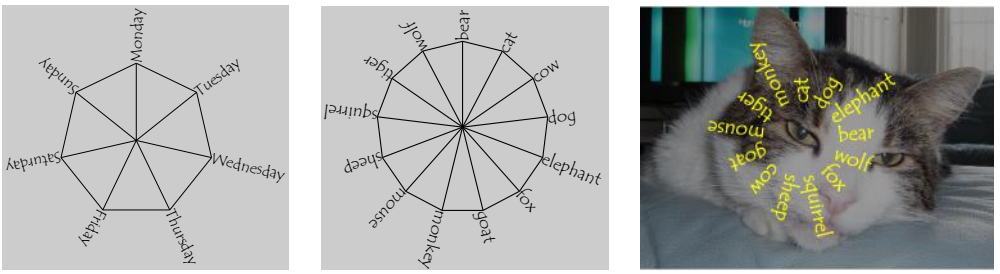
    The output should be an RGB image y of the same size as x, split into N rows and M columns of blocks of colour. The colour of each block should be the mean/median/mode colour of the pixels within this block in the original image. The value of p will determine which one of the three

options is used. Examples of an original image and the outputs for the three options are shown in Figure 8.24.

(a) original image (b) mean (c) median (d) mode

Figure 8.24: Examples of the output for the block-image problem.

18. Create a MATLAB function which takes as input a cell array with words (strings) and an integer mode. The length of the array is not limited. The function should display the words around a shape as shown in Figure 8.25. If the switch mode is 1, the words should be sorted alphabetically before displaying. Finally, load and display an image, create axes within it, and call the function to display the entries in the cell array as in the Figure. (Hints: (i) You may need to darken the image for the text to be clearly visible. (ii) The function output should have the handles to the lines so that their visibility can be turned off.)



(a) mode 0 (b) mode 1 (c) image with mode 0

Figure 8.25: Examples of the output for the circular text problem.

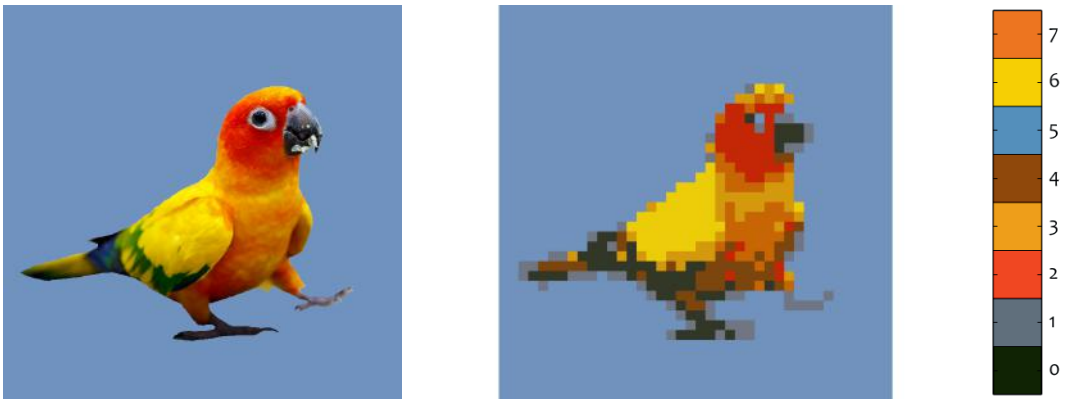
19. Paint by Numbers

    (a) Load a JPEG image (for best effect, this should be a low resolution, nearly square cartoon image). Choose the number of rows, M, and the number of columns, N, for the painting grid. Resize the image to the required grid size.

    (b) Convert the JPEG image into an indexed image with 8 colours and show it as in Figure 8.26 (b).

    (c) Prepare a figure that displays the colour map as shown in Figure 8.26 (c).

    (d) Prepare a figure that shows the grid and the numbers of the colours as in Figure 8.27.



(a) Original image (b) 40-by-40 pixels (c)

Figure 8.26: Painting by numbers

start CHALLENGE

Paint by Numbers - with as few Strokes as Possible

Write the script for problems (a)-(d) using the minimum possible number of lines. The rules are: (1) Each line has a maximum of 75 symbols. (2) The number of characters does not matter. (3) The figures may be produced in any order but each figure must be opened with the `figure` command. (The authors' current record is 7 lines.) Best of luck!

end CHALLENGE

20. 3D Colours

Each pixel in an image can be regarded as a point in a 3-dimensional space: RGB. Thus the pixels can be plotted using command `plot3` or `scatter3`. In addition, each pixel can be plotted with its own colour. Examples of three images and the respective 3D plots are shown in Figure 8.28.

Create a similar colour cube for an image of your choice. If you are plotting in a loop using `plot3`, make sure that you rescale the image to a much smaller size so that the number of plotted points does not exceed, say, 50,000. Otherwise the plotting will be too slow.

5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	1	6	3	6	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	3	3	7	3	7	3	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	2	2	7	2	2	2	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	7	2	2	0	5	2	1	1	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	2	2	2	2	1	2	0	0	0	1	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	1	2	2	2	2	2	2	0	0	0	1	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	3	2	2	2	2	2	2	0	1	1	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	1	7	2	2	2	2	2	4	4	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	6	7	7	2	2	2	2	2	7	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	6	6	3	7	7	2	2	7	7	3	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	6	6	6	6	3	7	7	7	7	3	3	3	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	6	6	6	6	6	3	3	3	3	3	3	3	3	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	3	6	6	6	6	6	6	3	7	3	3	3	7	7	3	6	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	6	6	6	6	6	6	6	6	3	7	3	7	7	7	7	7	3	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	1	6	6	6	6	6	6	6	6	6	6	7	7	7	7	7	3	2	4	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	1	0	0	4	3	6	6	6	6	6	6	6	6	3	7	7	7	7	7	7	4	1	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	0	0	1	4	0	6	6	6	6	6	6	3	3	7	4	2	7	7	7	4	4	1	5	5	5	5	5	5	5	5	5
5	5	5	5	5	1	1	3	0	0	0	0	3	0	3	6	6	3	3	3	3	3	4	2	7	7	4	7	4	1	5	5	5	5	5	5	5	5	5	5
5	5	1	1	4	4	4	4	4	0	0	0	0	0	0	0	3	4	3	0	0	4	4	4	4	4	7	4	2	5	5	5	5	5	5	5	5	5	5	5
5	5	1	0	0	0	0	1	1	1	5	1	1	0	4	0	0	0	0	0	4	4	4	2	4	4	4	4	2	3	5	5	5	5	5	5	5	5	5	5
5	5	5	5	1	5	5	5	5	5	5	5	5	5	1	0	0	0	0	0	4	4	4	0	4	4	7	0	4	7	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	1	0	0	4	4	0	0	0	1	1	5	5	5	5	1	5	5	5	1	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	1	0	4	4	1	5	5	5	5	5	5	5	1	1	1	1	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	1	0	0	1	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	1	0	0	0	1	1	1	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	1	0	0	0	0	1	0	1	1	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5

Figure 8.27: Paint-by-numbers grid for the Parrot image.

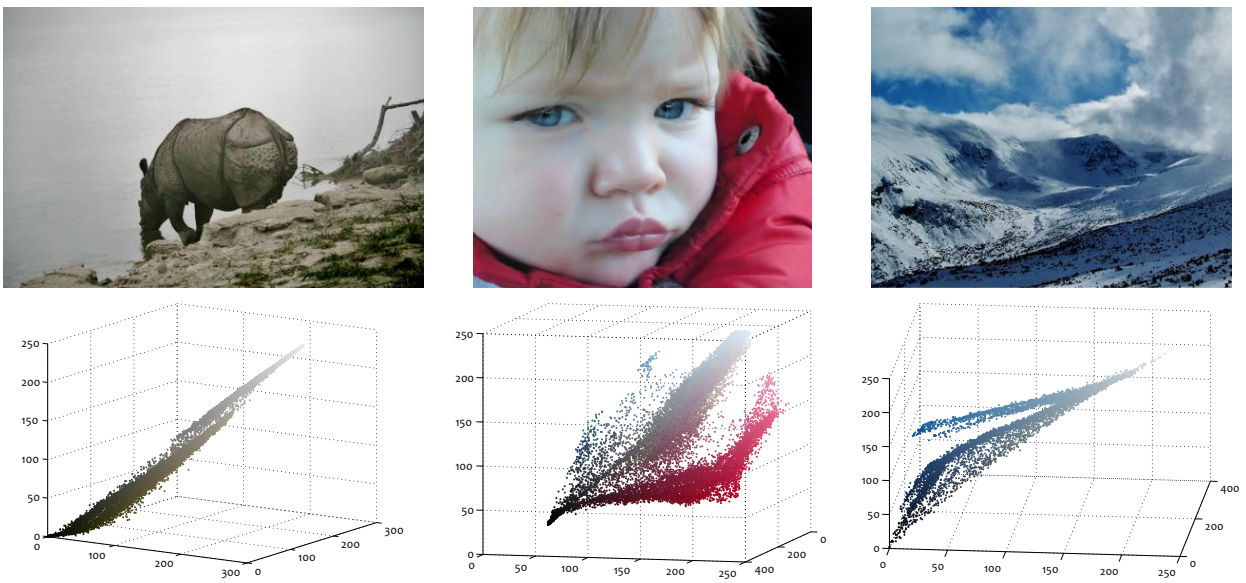


Figure 8.28: Images and 3d colour cubes.

Chapter 9

Animation

9.1 Animation Methods

There are several ways to produce animation in a figure in MATLAB. The figure can be redrawn with the new positions of the objects being animated. Another (and more elegant) way is to keep the figure and change only the positions of the objects. To do this, we need to take *handles* of the objects of interest. These handles contain information about all properties of the respective objects, including position and colour. An example of creating a handle is shown below;

```
h = plot(0,0,'k.');
```

To view the properties available and their values, type:

```
get(h)
```

Among the many properties displayed in the Command Window, there are:

```
                Color: [0 0 0]
            LineStyle: 'none'
            LineWidth: 0.5000
               Marker: '.'
           MarkerSize: 6
      MarkerEdgeColor: 'auto'
      MarkerFaceColor: 'none'
                XData: 0
                YData: 0
```

The coordinates of the marker are in properties `XData` and `YData`. Each property can be modified using the `set` command. For example;

```
set(h,'Marker','d','Markersize',40,'Linewidth',3)
```

will replace the dot marker in the figure with a diamond marker of size 40, drawn with a thick black line. Note that any number of properties can be changed with one `set` command. MATLAB is not case sensitive with respect to the properties' names, so `XData` is the same as `xdata`. (In some cases the properties can even be abbreviated; however until you are more familiar with them use, the full property name.)

To make the animation work, the changes must be displayed with a short time delay. Use the command

```
pause(s)
```

where s is a number of seconds. You will usually use a fraction of a second between two consecutive appearances of the animated objects.

When an object is drawn using a sequence of x, y coordinates, the `set` function will assign all the new values together. For example, if the object h is a triangle, both the XData and the YData will contain three values. Then

```
set(h,'XData',[3,2,9],'YData',[12,4,3])
```

will assign the new coordinates to the respective vertices of the triangle.

9.2 Mouse Control

The execution of a script or function can be paused in anticipation of a mouse click or a key-press by using:

```
waitforbuttonpress
```

This function returns 0 when terminated by a mouse click, or 1 when a key is pressed.

Mouse clicks can be used to grab an object. For example, upon a mouse click, the variable `gco` contains the handle of the object on which the click fell. If the click was not on any drawn object, `gco` will return the handle to the figure. For example, try the code below. It will draw a triangle and change its colour every time you click on the object. The loop will stop when a key is pressed.

```
h = fill(rand(1,3),rand(1,3),rand(1,3));
axis off
while ~waitforbuttonpress
    if gco == h
        set(h,'FaceColor',rand(1,3))
    end
end
```

The coordinates of the last mouse click can be read into a variable using;

```
point = get(gca,'CurrentPoint');
```

In two dimensions, the x coordinate is in point(1,1), and the y coordinate is in point(1,2). Figure 9.1 shows an example of a line drawing using mouse clicks.

```
figure, hold on, axis([0 1 0 1]), axis square, grid on
last = [];
while ~waitforbuttonpress
    point = get(gca,'CurrentPoint');
    if isempty(last), last = point; end
    plot([last(1,1) point(1,1)],[last(1,2) point(1,2)],'k.-')
    last = point;
end
```

mouse position (in axes coordinates)

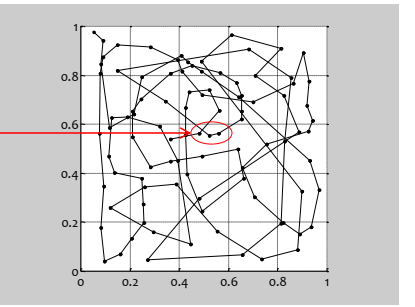


Figure 9.1: An example of a line drawing using mouse clicks.

Another useful command that you should look up is;

```
[x,y] = ginput(n);
```

start CHALLENGE

Without running the code below through MATLAB, try to figure out what it will do *within 2 minutes*. Draw on a piece of paper your predicted output. Subsequently, run the code and check whether you were correct.

```
figure('color','k')
axes('Position',[0 0 1 1]), hold on
axis([-1 2 -1 2],'off')
for i = 1:15
    pl = ginput(1);
    fill(pl(1)+rand(1,3)-0.5,pl(2)+rand(1,3)-0.5,...
        rand(1,3),'EdgeColor','w')
end
```

end CHALLENGE

9.3 Examples

9.3.1 Shivering Ball

To plot a 'ball', we can use one large marker. After fixing the axes, the x and y coordinates will receive different random values, and the ball will appear to 'shiver'. To make the example more interesting, let us change the colour of the ball to a random colour at each move. The code for this animation is shown in Figure 9.2.

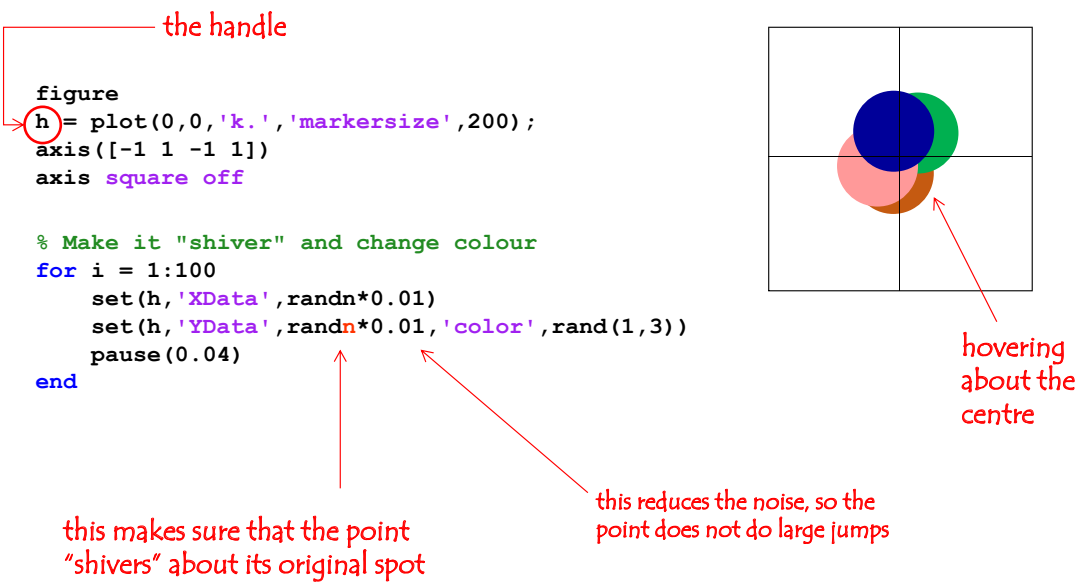
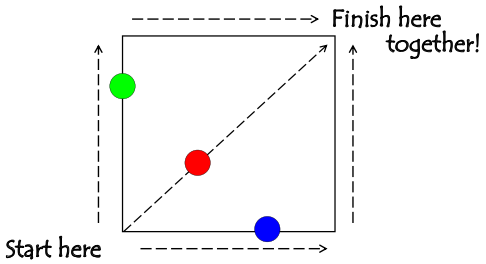


Figure 9.2: The shivering ball animation

9.3.2 Three Moving Circles

The task is sketched in Figure 9.3. Three circles, red green and blue, start from the bottom left corner of a square, move along the sides and the diagonal, and arrive together at the top right corner.



Hmm, how do I solve this problem?...

1. Plot the three dot markers and save the handles.
2. Run a loop where the blue marker moves right and the green marker moves up reaching the respective corners. The red should be half way through the diagonal at the end of the loop.
3. Run a second loop to complete the movement.

Figure 9.3: The three circles animation

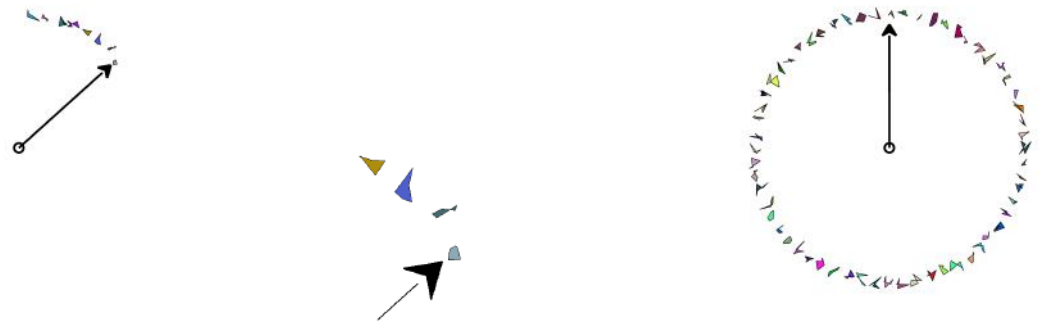
Solution. The animation is coded below.

```
figure,hold on
h1 = plot(0,0,'r.','markersize',100); % red marker
h2 = plot(0,0,'b.','markersize',100); % blue marker
h3 = plot(0,0,'g.','markersize',100); % green marker
plot([0 100 100 0 0],[0 0 100 100 0],'k-') % outline the square
axis([-3 103 -3 103]) % set the axes
```

```
axis square off
for i = 1:100 % loop 1 (half way)
    set(h1,'XData',i/2,'YData',i/2) % red marker goes twice more slowly
    set(h2,'YData',i), set(h3,'XData',i)
    pause(0.02)
end
for i = 1:100 % loop 2
    set(h1,'XData',i/2+50,'YData',i/2+50) % red marker starts from half up
    set(h2,'XData',i), set(h3,'YData',i)
    pause(0.02)
end
```

9.3.3 A Fancy Stopwatch

Create MATLAB code which will simulate a stopwatch. Initially, the figure should contain the clock arm (with a proper tip) at 12:00. The clock arm should move with 1 second offset to its new place. At the time it reaches the new spot, a 'fancy random tick' should appear near the tip, as shown in Figure 9.4 (a) (zoomed in sub-plot (b)). A very short 'beep' sound should be played at each move.



(a) (b) zoom of (a) (c) full circle

Figure 9.4: Fancy stopwatch

Solution. Recall the rotation matrix

$$R = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix},$$

where theta is the rotation angle in radians.[1]

```
% Fancy Stopwatch
bzz_freq = 1600; % needed for the short beep sound
fs = 8000; % the sampling frequency
```

[1] A full circle contains 2π radians.

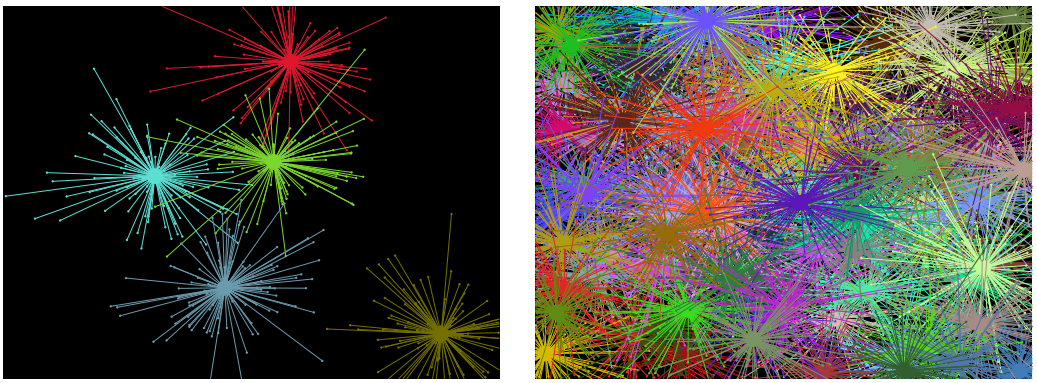
```
t = 0:1/fs:0.05; % the carrier variable
y = sin(bzz_freq*2*pi*t); % the short beep signal

figure
axis([-1.1 1.1 -1.1 1.1],'square','off') % format the axes
hold on

plot(0,0,'ko','markersize',8,'linewidth',2) % the pivot
% create the tip coordinates
tipCoord = [0 0 0.05 0 -0.05 0;[0.03 0.08 0 0.03 0 0.08]+0.8];
armCoord = [0 0; 0 0.8]; % arm coordinates
th = fill(tipCoord(1,:),tipCoord(2,:),'k','linewidth',2); % tip handle
ah = plot(armCoord(1,:),armCoord(2,:),'k','linewidth',2); % arm handle
theta = 2*pi/60; % angle corresponding to 1 second
R = [cos(theta) sin(theta);-sin(theta) cos(theta)]; % rotation matrix

pause % start the stopwatch by pressing a key

tic % measure time to determine the pause needed to make up 1s
for i = 1:60
    tipCoord = R*tipCoord; % rotate tip at angle for 1 second
    armCoord = R*armCoord; % rotate arm at angle for 1 second
    sound(y,fs) % short beep
    % update the tip and the arm handles
    set(th,'XData',tipCoord(1,:),'YData',tipCoord(2,:))
    set(ah,'XData',armCoord(1,:),'YData',armCoord(2,:))

    % create the fancy ticks at the tip of the hand
    fill((rand(1,5)-0.5)*0.1+armCoord(1,2)*1.2,...
        (rand(1,5)-0.5)*0.1+armCoord(2,2)*1.2,rand(1,3))

    pause(0.93) % pause corresponding to 1s => needs tuning
end
toc
sound(y,fs),sound(y,fs),sound(y,fs),sound(y,fs),sound(y,fs)
```

Listing 3: A fancy stopwatch.

9.4 Exercises

1. Fireworks. Use the mouse to create figures similar to the ones in Figure 9.5. On a mouse click, the centre of the star should be plotted at the position of the mouse. 100 random rays should be drawn from the centre. The next mouse click will generate a new star centred at the mouse position. The colours of the stars are random. The script should finish when a key, on the keyboard (rather than on the mouse), is pressed.

2. Highlighting



(a) 5 stars (b) 100 stars

Figure 9.5: Examples of 'fireworks'

Load and show an image of your choice. When the user clicks on the image, highlight the region around the position of the click. The size of the region should be a changeable constant in your code. An example is shown in Figure 9.6.

Figure 9.6: An example of a highlight

3. Create an animation whereby a filled square will grow progressively in 10 steps.

4. Create an animation so that 10 squares, nested as in Figure 9.7, evolve simultaneously. The outer square disappears at the next step, and all 9 inside squares grow by one size. At each step, a new smallest square of a random colour appears in the middle. Each square must keep its colour during the growing stages. An example of 4 consecutive steps is shown in Figure 9.7.

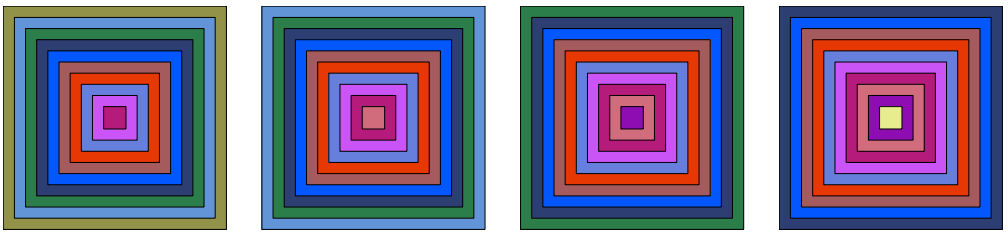
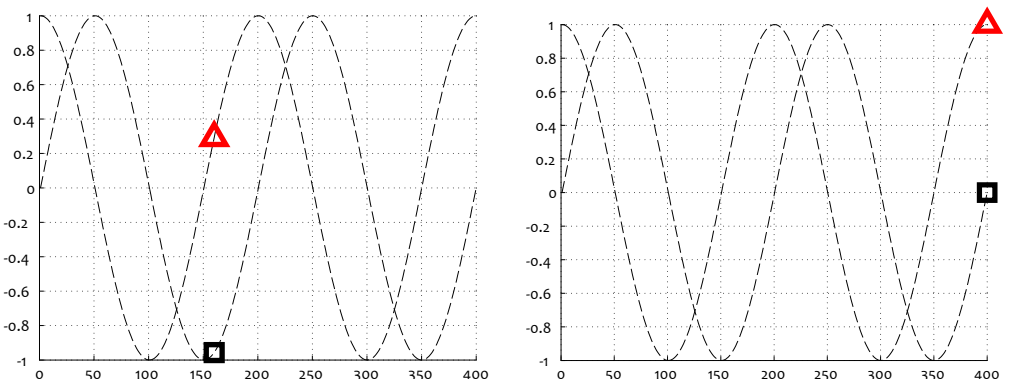


Figure 9.7: Four steps of an animated sequence of nested squares

5. The dashed lines in Figure 9.8 show two trajectories: $\sin(\theta)$ and $\cos(\theta)$, where θ varies from 0 to 4π. Create an animation where a black square marker and a red triangle marker move simultaneously in 400 steps, following the respective trajectories.



(a) During motion (b) End position

Figure 9.8: Animated triangle and square markers moving along a sine and a cosine trajectories

6. Open a figure with a yellow background. Place a text string 'Stopwatch' near the top left corner. Use large letters, and a font of your choice (not the default font). Position anywhere in the figure the number zero, with a larger size of the chosen font. Ask the user to input a number of seconds. Get your stop watch to count the seconds in nearly real time. An example of the clock face is shown in Figure 9.9.

   Take into account that there is a slight delay due to the printout, so the 'pause' command should not be exactly for 1 second but a little less. (Hint: Use `tic` and `toc` to time 10 seconds and tune the argument of the 'pause' function accordingly.)

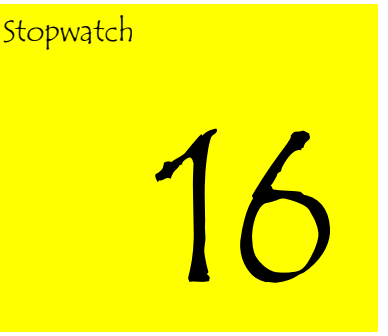
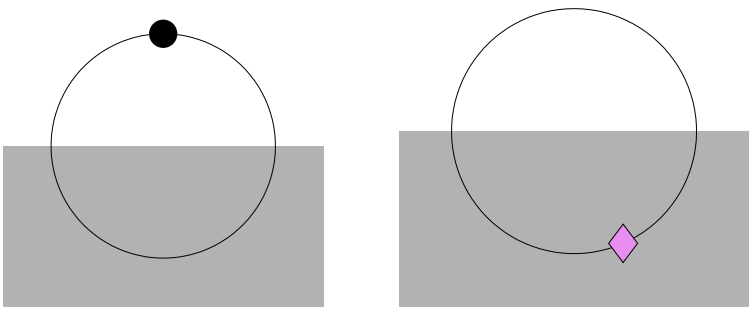


Figure 9.9: An example of the stopwatch face

7. Plot a circle trajectory as shown in Figure 9.10 (a). Plot a large black round marker on the zenith of circle. Shade the bottom half of the figure grey. Make the marker complete a full circle on the drawn trajectory in 100 steps. Change the dot into a diamond when it enters the shaded zone,

and revert it back to dot marker when it leaves the zone. In addition, make the diamond change its fill colour randomly at each step. An example is shown in Figure 9.10 (b).



(a) Starting position (b) Marker within the shaded zone

Figure 9.10: Animated circular motion

8. Planets

Write MATLAB code to do the following:-

(a) Create an animation of a solar system with one sun and two planets. Each planet orbits the sun in a circular orbit. The two orbits have different radii. One of the planets goes clockwise, and the other goes anti-clockwise. The outer planet takes twice longer to make one full circle than the inner planet. Plot in the animated figure the two orbits with dashed lines. Remove the axes and make sure that they stay square and fixed (don't float with the animation).

(b) Give the outer planet a moon. Plot the just the orbit of this moon (a circle around the planet) and make sure that the moon and its orbit move together with the planet. The moon itself does not have to follow its orbit for now.

(c) Make the moon orbit its planet at a speed that you choose.

An example of the required figure is shown in Figure 9.11. The text is NOT required. It is for your reference only

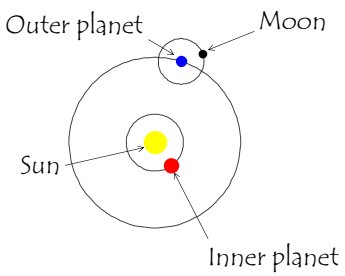
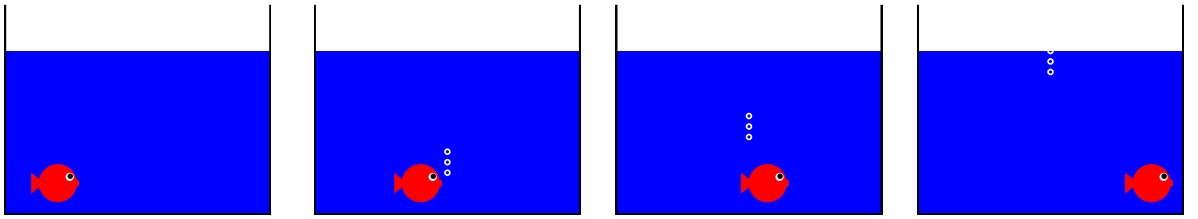


Figure 9.11: A solar system

9. Fish Tank

   Use MATLAB to draw a fish tank as shown in Figure 9.12. Place a fish near the left wall. You can draw the fish using markers and filled polygons. Make the fish move slowly across to the right wall of the tank. When the fish reaches the middle; it should breathe out three bubbles which float towards the surface. Figure 9.12 (a)–(d) show the beginning, middle and end of the animation.



(a) (b) (c) (d)

Figure 9.12: Stages in the fish tank animation

10. The Umbrella

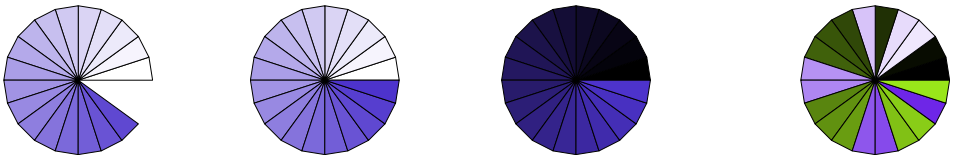
    (a) Create a function which that has two input arguments; a number of sectors N, and a colour v (a vector of red/green/blue, each component in the interval [0,1]). Open up a figure and plot consecutively N sectors (triangles) in a circle. Starting with a white sector, animate each subsequent sector appearing. The sector should have an interpolated colour between white and v. An example of the output is shown in Figure 9.13 (a). Make sure that the figure size does not change with each new sector that appears. Write a script to demonstrate the function and give examples of the output.

    (b) Expand the function written in (a) to include a third boolean (true/false) input parameter. If true, the starting colour is white; if false, the starting colour is black. An example with a black start sector is shown in Figure 9.13 (c).

    (c) Write MATLAB code which calls your function, and then waits for a mouse click. If the click is on a sector, the colour of this sector changes to the opposing colour. For example, if the current colour is [0.3 0.7 0.2], the opposing colour is $[1,1,1]-[0.3,0.7,0.2]=[0.7,0.3,0.8]$. If the click does not fall on any sector, close the figure. An example with several clicks is shown in Figure 9.13 (d).

11. Rotating Random Shapes

    (a) Write a MATLAB function that will take three input arguments: k, the number of vertices for a shape component, $r \in [0,1]$, a scaling factor, and m the axis limit (for formatting the axes using `axis([-m m -m m])`. The function must open up a figure, plot a symmetrical shape of a random colour, scale it by r, and rotate it about the middle to a full circle using

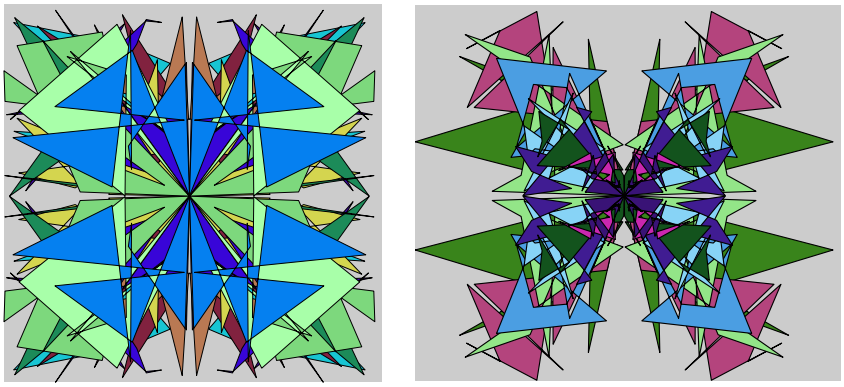


(a) During Animation (b) End State (c) Black Start Sector (d) Opposing Colours

Figure 9.13: Examples of the 'Umbrella' problem outputs.

100 steps. An example of the type of the required shape is shown in Figure 5.11 (see the problem about producing this shape).

(b) Write a script that calls the function from the previous problem 10 times, with the same scaling factor and axes limit. After the calls, the current figure should have all 10 forms in it as shown in Figure 9.14 (a). On a new figure, make 10 calls to the function with a progressively decreasing scaling factor. An example of the figure at the end of the 10 calls is shown in Figure 9.14 (b).



(a) Without scaling (b) With scaling

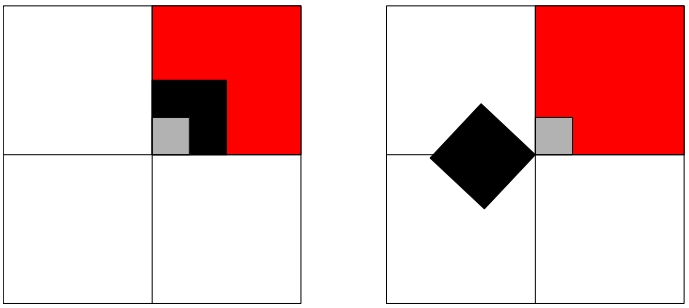
Figure 9.14: Examples of the 10-shapes output.

12. Rotating Square

Create an animation starting with three squares as shown in Figure 9.15 (a). The black square rotates clockwise and completes a full circle around the centre. Figure 9.15 (b) shows a position of the square during the animation.

13. Rotating Triangles

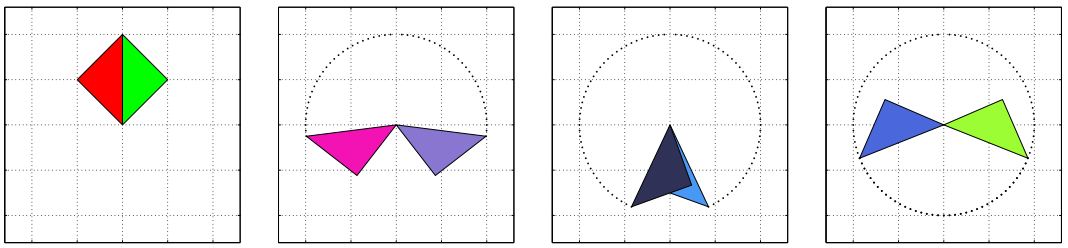
Write MATLAB code to produce the following animation. Plot two triangles as shown in Figure 9.16 (a). Each of the two triangles should rotate in a full circle about the centre. The two rotations should be in different directions as shown in Figure 9.16 (b) to (d). The final position should be the same as the starting position. The rotation should be done in 100 steps. At each



(a) Starting and ending position (b) During animation

Figure 9.15: Rotating-square animation

step, each triangle should assume a new random colour. The tips of both triangles should produce a dot trace as shown in Figure 9.16 (b) to (d).



(a) (b) (c) (d)

Figure 9.16: Rotating triangles

14. Write a script that will run 15 random jumps of a 'frog' within a 'pond'. The pond should be the unit square coloured in blue. The frog must pause for 0.5 seconds at each location. It should and leave a trajectory behind, plotted with a dashed green line. The initial position of the frog is the point (0.5,0.5). The frog should be presented as a green triangle. The expected output at the end of the animation is shown in Figure 9.17.

15. Pastel Folders

    Plot a collection of 9 folders of random *pastel* colours. Offset them as shown in Figure 9.18. (Hint: use function `rectangle` which allows for round corners.) Label the folders with the numbers from 1 to 9 as shown in the figure. After a key is pressed, make the folders shrink and disappear, one at a time, in a random order. The folder number should disappear before the shrinking starts.

16. Scrambled Eggs

    Recall the problem, from the Images Chapter, where you had to create a function that breaks up an image into blocks, and shuffles the blocks? Use your solution to help with this problem.

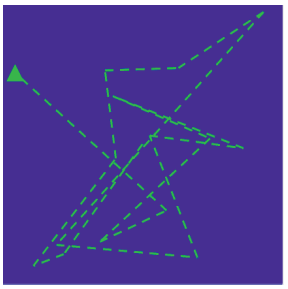


Figure 9.17: Expected output at the end of the animation for the jumping frog problem

Figure 9.18: Pastel-coloured folders

Load an image, called the 'Original' (you don't need to use a picture of eggs!). Split the image into 4-by-5 tiles and shuffle them, the 'Scrambled eggs' - as shown in Figure 9.19.

Here comes the twist: manipulate the scrambled image further, so that one random tile is missing, and another random tile is repeated in its place. Display the image as in Figure 9.19 'Repeated tile'. Hold for 3 seconds, and then display another figure where the whole image is darker apart from the two repeated tiles, which should stay of the same colour.

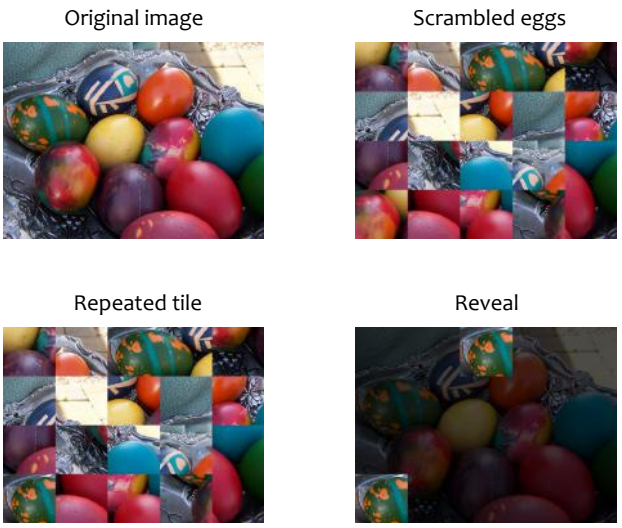


Figure 9.19: The 'scrambled eggs with a twist' problem

Challenge your friends to discover the repeated tile within the three second interval.

17. Load a JPEG image and make the four quadrants blink with different transparent colour: red, green or blue only, in a clockwise pattern. An example of a full rotation of four random colours is shown in Figure 9.20. (Hint: The transparent colour is obtained by setting the respective colour pane to the maximum value while keeping the other colour panes.)

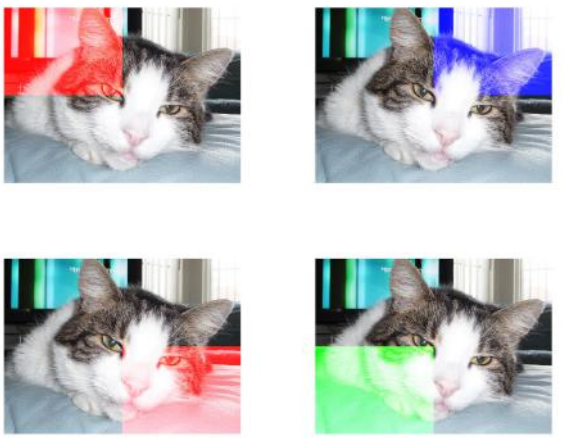


Figure 9.20: Four transparent colours

18. The Grazer

    Create a 10 by 10 matrix filled with ones which will be the grazing ground. Plot the ground using the `spy` command, as shown in Figure 9.21. Create a 'grazer' at a random position in the array.

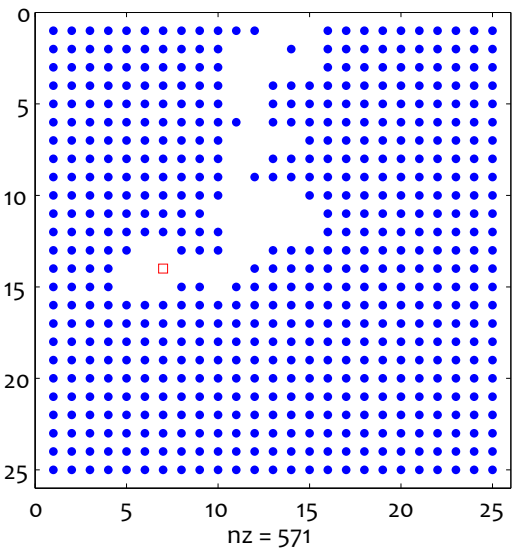


Figure 9.21: The Grazer problem with a 25-by-25 grazing ground

The grazer moves to a randomly chosen neighbouring cell at each time step. Neighbouring cells are only north, south, east and west (i.e. four-connected cells). The grazer is not allowed to move out of the borders of the grazing ground. It eats the provision in the cell it is in, which is marked as empty space in the Figure. The grazer itself is a red square marker.

Your animation should show the grazing ground and the grazer's position at each step. The code is run until there is no food left.

You can run a competition with your friends for the fastest grazer.

19. Load a JPEG image and, after a key is pressed, make a red horizontal red 'laser beam' line run down from top to bottom. The part above the line should turn grey and the part underneath should remain in colour. Make a very short beep sound with each step of the line movement. An example is shown in Figure 9.22.

Figure 9.22: Laser beam revealing the grey image

Chapter 10

Graphical User Interfaces - GUI

10.1 Programming GUIs

GUIs can be created interactively using the `guide` command. Alternatively, you can program the elements of the GUI and set up their parameters from within your code. Figure 10.1 illustrates the two approaches.

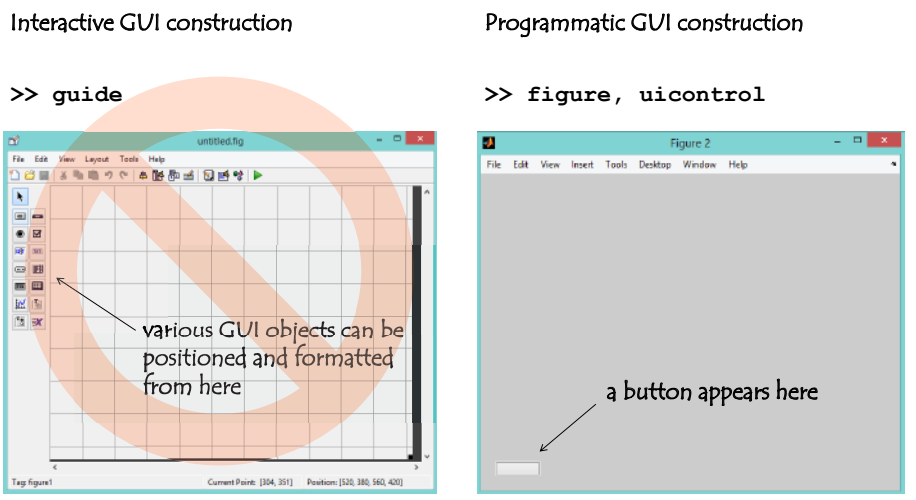


Figure 10.1: Two approaches to creating GUI in MATLAB

In this book, we take the second approach, and confine the examples and the exercises to using only the push button object. Like with any object in a MATLAB figure, the properties of the button are reachable using the `get` and `set` commands.

The most important property is the `Callback`, which determines what the button does when pressed and released. The Callback can be set as a string in the definition of the button or afterwards, using the `set` command. For example,

```
figure, uicontrol('Callback', 'beep')
```

will create a button at the bottom left corner of the figure, which will beep (with the unpleasant sound of a MATLAB error `:))` when pressed.

The callback can be given as a function handle instead of a string. Usually the whole GUI is contained within one function file. The Callback function must have two compulsory parameters – object and event. Any input parameter which you want to transmit to the function will be listed next. As an example, try the code below.

```
function my_first_gui
figure
for i = 0:9
    uicontrol('Units', 'Normalized', 'Position', [0,i,i+1,1]/9, ...
        'BackgroundColor', rand(1,3), 'Callback', @long_button)
end
function long_button(o,~)
p = get(o,'Position');
if p(1) == 0, p(1) = 1-p(3); else p(1) = 0; end
set(o, 'Units', 'Normalized', 'Position', p)
```

10.2 Examples

10.2.1 One Colour Button

Design a figure with one button in the middle as shown in Figure 10.2. When the button is pressed the background colour of the figure should change to a random colour. The three numbers that make up the colour should be displayed as the button string.

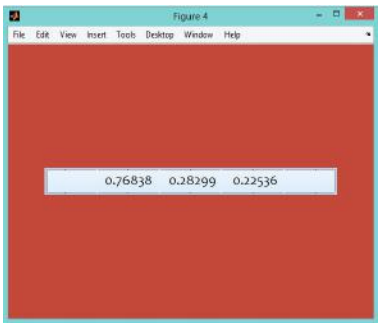


Figure 10.2: Random colour GUI

Solution.

The code is shown below. The default 'uicontrol' object is a push button, so we don't have to specify this explicitly. Normalised units are easier to use than pixel units, in order to position the button in the middle of the figure. The code below includes a choice of font and font size. The Callback consists of three actions: (1) Generate a new colour as three random values (array t); (2) Set the figure colour to t; and (3) Set the string if the button to the values in t. For the latter, the values must be converted from number to string, hence the 'num2str' command.

```
figure
uicontrol('Units','Normalized','Position',[0.10 0.45 0.80 0.1],...
    'FontName','Candara','FontSize',16,'Callback',...
    't = rand(1,3);set(gcf,''Color'',t);set(gco,''String'',num2str(t))');
```

Note that the code can be shortened by using only the beginning of the words for the properties and their values. The words can be shortened to the minimum number of letters which eliminates any ambiguity. For example, 'Units' can be shortened to 'Un', and 'Position' to 'Po'. The font set-up is optional, so the code can be as follows:

```
figure, uicontrol('Un','N','Po',[1 4.5 8 1]/10,'Ca',...
    't = rand(1,3);set(gcf,''Color'',t);set(gco,''Str'',num2str(t))');
```

When using string Callbacks, pay particular attention to apostrophes. As MATLAB uses these to delimit strings, you can end up with broken callbacks. This is why in the previous examples, the apostrophes are *escaped* by a second apostrophe.

10.2.2 Disappearing Shapes

Create a set of 5 random shapes, each one having 20 random vertices and filled with a random *light* colour. Plot the shapes in a row as shown in Figure 10.3. When left-clicked upon, the shape should change its colour to a darker colour. The 7th click on any of the shapes should delete it.

Figure 10.3: Light colour shapes which progressively darken and disappear with the 7th click. Shape #4 has been clicked on 4 times

Solution.

Some thoughts about the solution are given in Figure 10.4.

The code for the solution is shown below:

```
figure, hold on, axis([1 6 -0.3 1.3]), axis equal off
h = []; % initialise the array with the handles
for i = 1:5 % plot the figures
    h(i) = fill(rand(1,20)+i,rand(1,20),rand(1,3)*0.4+0.6,...
        'EdgeColor','none'); % store the handles in array 'h'
end
times = zeros(1,5); % initialise the array with 'times-clicked'
```

Hmm, how do I solve this problem?...

1. The shapes don't look like UI buttons.
2. I can use "fill", create handles to the shapes, and use the mouse coordinates to get the object.
3. I will need an array with 5 elements to keep the record of the clicks.
4. Use "waitforbuttonpress". If the object clicked upon is among the handles, darken it and check how many clicks it has received. If 7, delete.
5. How many times should I repeat these actions? There is no instruction about this so I can choose my own option. There is no point keeping the loop open if there are no shapes, so I can close the loop when all objects have been deleted. Alternatively, I can break the loop upon a right-click or a key from the keyboard. WHILE loop is needed.

Figure 10.4: Thoughts about the solution of the disappearing shapes problem.

```
while ~isempty(h) % run until there are no more shapes
    waitforbuttonpress
    j = find(gco==h); % identify the shape clicked upon
    if ~isempty(j) % if not clicked outside a shape
        set(h(j),'FaceColor',get(h(j),'FaceColor')*0.78) % darken
        times(j) = times(j) - 1; % record the click
        if times(j) == 7 % if clicked 7 times
            delete(h(j)) % remove from the figure
            h(j) = []; times(j) = []; % shrink 'h' and 'times
        end
    end
end
```

10.2.3 Catch-me-up Game

Create a timed game where the player has to click on 10 randomly drawn triangles. The triangles appear one after another. If the click is not on the triangle, play a 'beep' sound and continue with the next triangle. At the end of the game, display the time taken since the appearance of the first triangle.

Solution.

```
figure, hold on, axis([0 1 0 1]), axis square off
h = fill(rand(1,3),rand(1,3),rand(1,3)); % first triangle
tic % starting the timer
hits = 0;
while hits < 10
    set(h,'XData',rand(1,3),'Ydata',rand(1,3),'FaceColor',rand(1,3))
    w = waitforbuttonpress;
    if w == 0 && gco == h
        hits = hits + 1; % hit
    else
```

```
        beep
    end
end
time10hits = toc; % time taken for 10 hits
t = text(0.7,0.8,sprintf('Your time: %.2f s',time10hits));
set(t,'FontName','Candara','Fontsize',16)
```

10.3 Exercises

1. Spaceship, Moon and Stars

   (a) Use the mouse to draw a 'spaceship'. While the mouse button is being clicked, keep collecting points. When a key from the keyboard is pressed, fill the ship with grey colour and set the background to black. (You may need to close the shape manually as the user may not select the same first and last point).

   (b) Upon the next mouse click, plot a moon centred at the position of the mouse.

   (c) Plot four constellations centred at the positions of four subsequent mouse clicks. Each constellation should have 10 stars of different sizes.

   An example is shown in Figure 10.5.

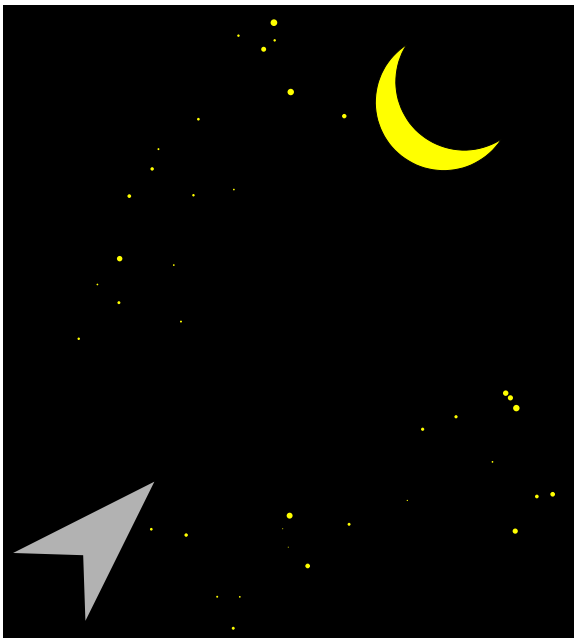


Figure 10.5: Spaceship, moon and stars

2. Create a figure with 26 buttons displaying the letters of the Latin alphabet as shown in Figure 10.6.

Figure 10.6: Button alphabet

When pressed, each button should turn its colour to black. Two randomly chosen buttons should hide 'bombs'. When pressed, a bomb button will delete all buttons and turn the figure background to black.

3. Write a script which displays a panel of 2 rows by 4 columns of push buttons with different random background colours. (This is possible with one loop!) When a button is pressed, the title of the panel is changed to the RGB values of the button's colour, and the background of the panel is changed to that colour. An example is shown in Figure 10.7.

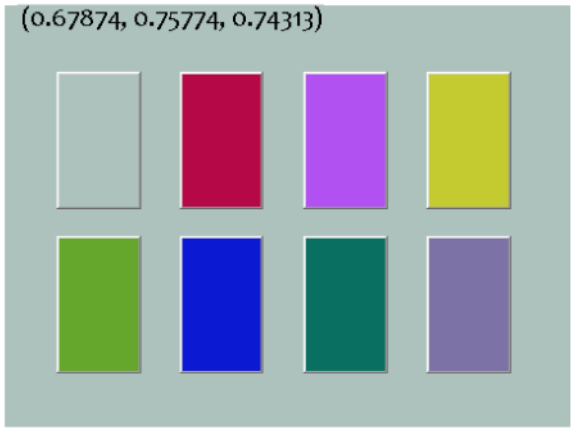


Figure 10.7: A panel of colour buttons

4. Open a figure with a black background. Create a green push button when the user left clicks with the mouse. The button should be centred at the point of the mouse click. The button string should be 'Press to disappear'. When the button is pressed, the string should disappear, and the button should shrink in 100 steps towards its centre. Finally, the button should disappear, and the figure should change its background to green.

5. Create the layout shown in Figure 10.8. When presses, a button should move the whole panel with 4 buttons in the said direction. The panel should not leave the figure space. When an edge or a corner is reached, pressing the button for continuing in the same direction should have no effect.

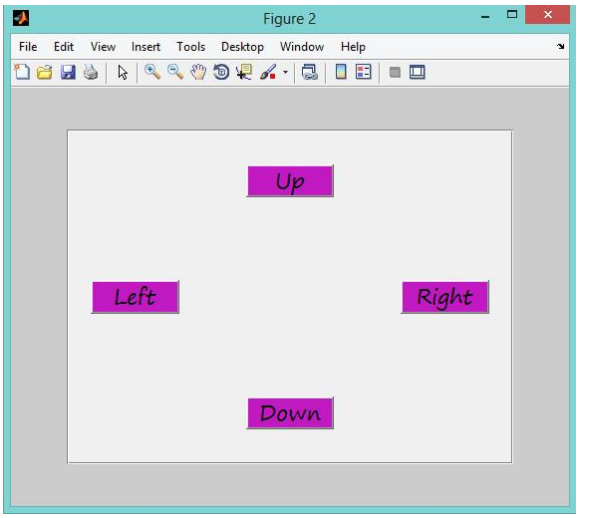


Figure 10.8: The layout for the moving panel problem

6. Scrabble Helper.

   In the game of Scrabble, it is important to have at hand a view of the remaining tiles. The total number of tiles is 100. There are 27 different tiles: 26 for the letters of the alphabet, and two empty tiles which can be placed as any letter. The number of tiles for each letter corresponds roughly to this letter's frequency in the English language. The letter set, including the 2 empty tiles at the end, and the corresponding frequencies can be introduced in MATLAB using the two lines below:

```
ltf = [9 2 2 4 12 2 3 2 9 1 1 4 2 6 8 2 1 6 4 6 4 2 2 1 2 1 2];
let = ['A':'Z',' '];
```

   The Scrabble helper should show you the available tiles for a given game. Create a function with no input arguments. The function should open up a figure with 10× 10 buttons corresponding to the Scrabble tiles, as shown in Figure 10.9.

   Upon pressing, a button should change its background colour. If the colour is light, it should become dark, and vice versa. In this way, the tiles that have been used in the game are masked with a dark colour in the figure.

7. Open up a (nearly) square figure. Create 16 square buttons with random colours, as shown in Figure 10.10 (a). A random ASCII character should be displayed on each button - see Table 7.1. The characters don't have to be unique. When pressed, a button should disappear.

   One randomly chosen button should have a different behaviour. It should wipe the figure clean with a black background, and plot 10 'fireworks' at random places and with random colours. Each 'explosion' should have 100 rays of different length and direction, as shown in Figure 10.10 (b).

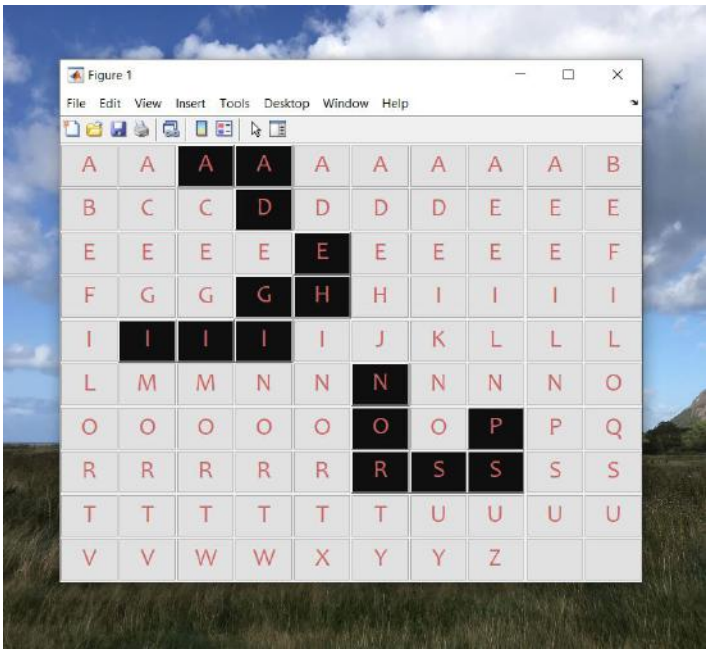
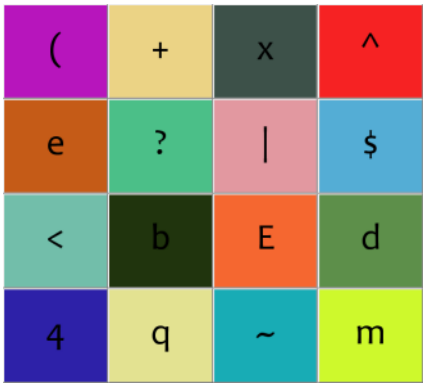
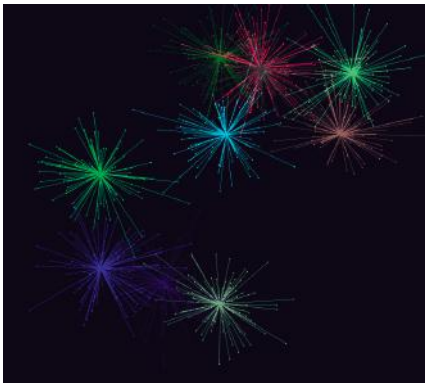


Figure 10.9: The Scrabble helper



(a) ASCII labelled buttons



(b) Fireworks

Figure 10.10: ASCII labelled buttons

(Hint: command 'randn' will be useful here.) The firework should be shown consecutively, at random time intervals, each interval not exceeding 1 second.

8. Write a script which will open a figure and position 4 push buttons in the corners as shown in Figure 10.11. Initially, fill the four squares with white colour, then square and remove the axes.

   When a button is pressed, the respective square is shown in random colour while the remaining squares are shown in white. (In the example in the Figure, the top right button is pressed.) Display underneath the figure the RGB values which make up the colour of the non-white square, as shown.

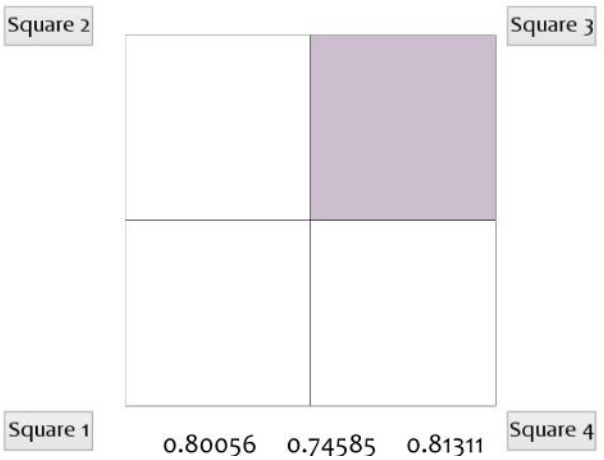


Figure 10.11: Four squares with random colours, one filled at a time

9. Write the shortest code to accomplish the following:

   (a) Create a figure with 20 buttons with colours gradually changing from black to yellow, as shown in Figure 10.12 (a).

   (b) When pressed, each button should change its colour to the colour's complement, that is, the colour that completes it to white. For example, a colour [0.2, 0.4, 0.1] has a complement [0.8, 0.6, 0.9]. An example is shown in Figure 10.12 (b).

   (c) Choose any three random buttons. In addition to changing their colour, when pressed, the three bottons should do the following. The first button should label the buttons with the numbers 1-20. The second button will check whether there are numbers on the buttons; if so, it will shuffle the buttons each time it is pressed. If there are no numbers, the button will do nothing. The third button will remove the labels from all of the buttons.

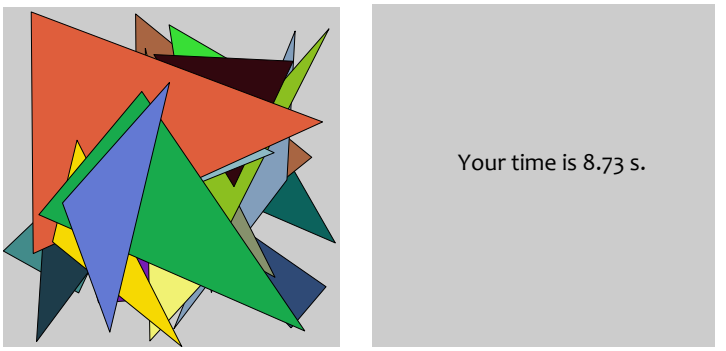
(a) (b)

Figure 10.12: Black to yellow row of buttons

10. Random Triangles

   (a) Generate a figure with 20 random triangles in the unit square, filled with random colours (Figure 10.13). Plot each triangle, only if its area is greater than 0.05. Hint: The area of triangle with vertices $A(a_1, a_2)$, $B(b_1, b_2)$ and $C(c_1, c_2)$ is;

$$S = \frac{1}{2} \times | a_1(b_2 - c_2) + b_1(c_2 - a_2) + c_1(a_2 - b_2) |$$



(a) Starting figure (b) Ending figure

Figure 10.13: Remove-the-triangles game

Note: The vertical bars are mathematical 'jargon' for the *absolute* value. This will produce a positive value even if the result of the expression is negative.

(b) Program the game so that, in order to remove a triangle, you need to click with the mouse over it. Remove the triangles in reverse order of the way they were generated - i.e. last triangle goes first or the user must click the 'top' triangle each time. If the user clicks on a different location, i.e. not the triangle whose turn it is to be moved, produce a short beep-beep sound. The game finishes when all 20 triangles have been removed. To time the user's performance, start the clock when the user clicks with the mouse over the figure for the first time (correct or incorrect click). When the game finishes, show the user's time in the centre of the figure as in Figure 10.13 (b).

11. Random Rectangles

(a) Plot k random-sized rectangles of random colours at positions indicated with the mouse. At each click, the respective rectangle should appear in the figure. An example of the output with $k = 15$ is shown in Figure 10.14 (a).

(b) Delete a rectangle if the user clicks on it with the mouse. Measure the time from the first click to the end of the game where all rectangles disappear. Display, in the centre of the figure, the time taken at the end of the game play.

12. Colour boxes game.

Begin by displaying a 4-by-5 matrix of black boxes. A random box changes its colour to a random colour. When the mouse is clicked on that box, the box disappears and a new box changes its colour. The game finishes when all the boxes disappear from the figure. At the end, the time taken to remove all the boxes is displayed. An example of the layout is shown in Figure 10.15.

Design a further feature of your choice for this game. This could be sound, text, new rule, new level, a text window showing the number of wrong clicks, changing background, etc.

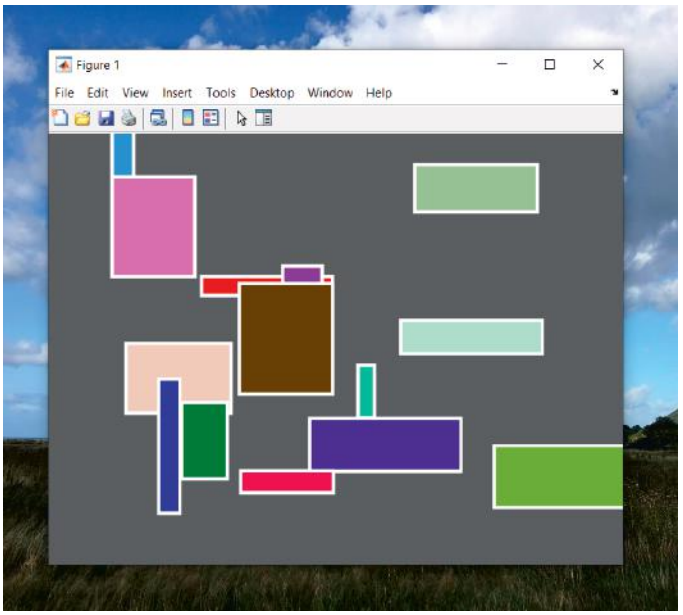


Figure 10.14: Random rectangles created at the points of mouse click

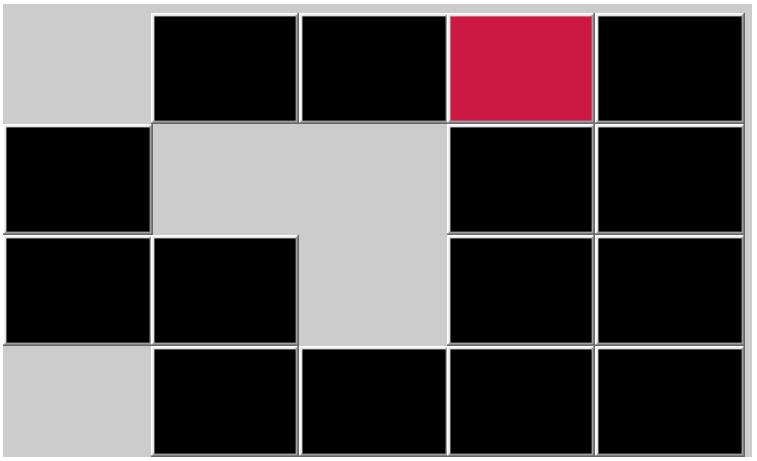


Figure 10.15: An example of the layout of the Colour boxes game during game play

13. Program a graphical user interface (GUI) as follows. The layout should be as shown in Figure 10.16. All blocks are push buttons, and start as all grey. When pressed, each button should change its colour to a random new colour, as shown in the figure. Plant a 'destroyer' at a random position in the matrix. Once the destroyer is pressed, instead of changing colour, the buttons in its row and in its column are deleted.

14. How fast can you find the numbers?

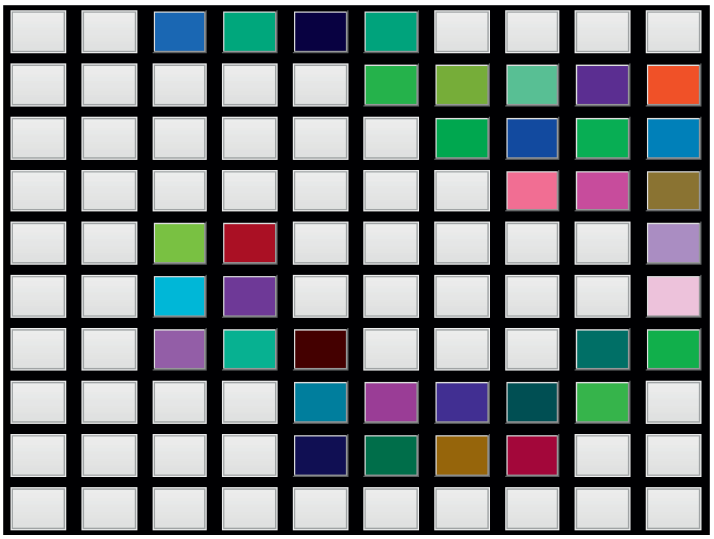


Figure 10.16: The 10 × 10 array with buttons

(a) Create a GUI figure with 100 push buttons, arranged in 10 rows and 10 columns. Each button should have a dark random background as shown in the Figure 10.17 (a). The numbers from 1 to 100 should be randomly assigned to the buttons and displayed in white.

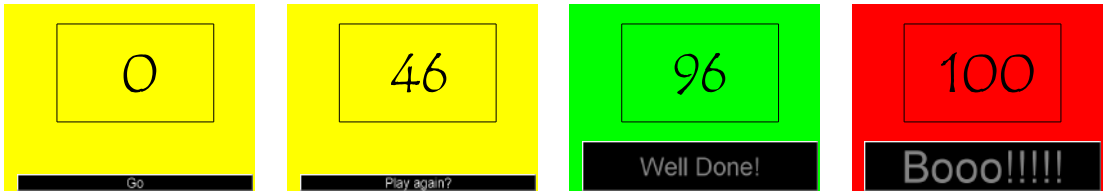
(b) Create a game which starts the clock, displays the buttons, and finishes when the player has pressed all 100 buttons in the correct order, starting with 1, 2, 3, and so on, up to 99, 100. If an incorrect button is pressed, a random two-note sound is made (two beeps with different random frequencies). If the correct button is pressed, this button is disabled, its background is turned to white, and a 'click' sound is played. A snapshot of the game is shown in Figure 10.17 (b). When the last button (100) is pressed, the game should wipe out the figure and show the time elapsed since the start of the game.

(a) Beginning of the game (b) During play

Figure 10.17: Examples of the 1 to 100 Game output.

15. Create a game similar to the games in the TV show 'The Cube'.

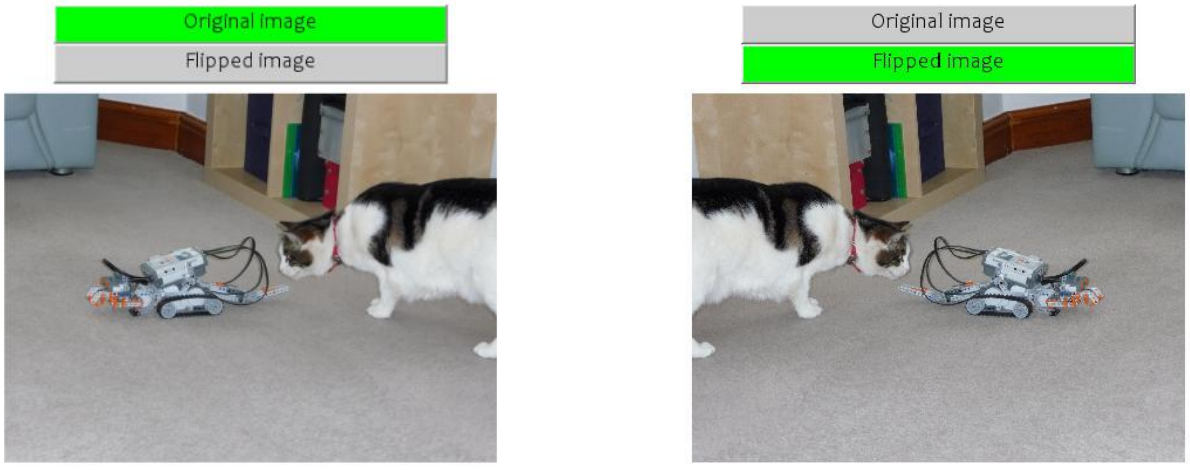
Prepare a figure with a button and a counter set to 0, as shown on in Figure 10.18 (a). Upon pressing the button 'Go', the numbers should start going up towards 100, fairly quickly. The aim is to stop the counter between 90 and 99 inclusive. The string on the button should change to 'Stop!'. Pressing the button again should stop the counter and disable the button. If the number is between 90 and 99 (inclusive), display 'Well Done!' as the button string and change the figure background to green (Figure 10.18 (c)). If the count reaches 100, display a suitable message and change the figure background to red (Figure 10.18 (d)). After 2 seconds, enable the button, change the background back to yellow and, change the button string to 'Play again?' If the button is pressed, start the game again.



(a) (b) (c) (d)

Figure 10.18: Examples of the set-up and the output of the reaction-gauging game featured in the TV show 'The Cube'.

16. Use `uigetfile` to select and load a JPEG image. Create GUI with the layout shown in Figure 10.19 (a). The top push button should display the original image. The bottom push button should display the flipped image as shown in Figure 10.19 (b). A green button should indicate which of the two images is currently shown.



(a) (b)

Figure 10.19: Layout for the mirror-image GUI.

17. The DrawMaster

    Load and display a JPEG image. Create handles for the figure and the axes. Create a loop for selecting a sequence of points on the figure. Plot and join the consecutive points with a yellow line and a dot marker as shown in Figure 10.20. The loop should run, until a key from the keyboard is pressed. Collect the points in an array.

Figure 10.20: Examples for the DrawMaster problem

18. Moving Car

    Create an animation where an object (for example, a car) moves horizontally from left to right and back. The figure should contain three buttons as shown in Figure 10.21. Buttons 'Froward' and 'Backward' should move the object a little, but visible, step in the respective direction. The middle button, 'Move', should cause the object to move from the current position to the right edge, *turn back*, move to the left edge, and *turn forward* again.

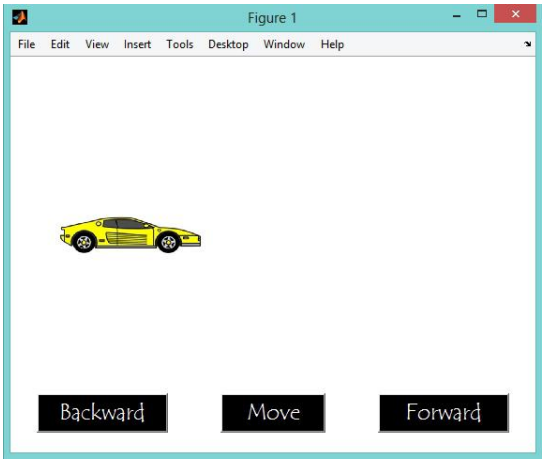
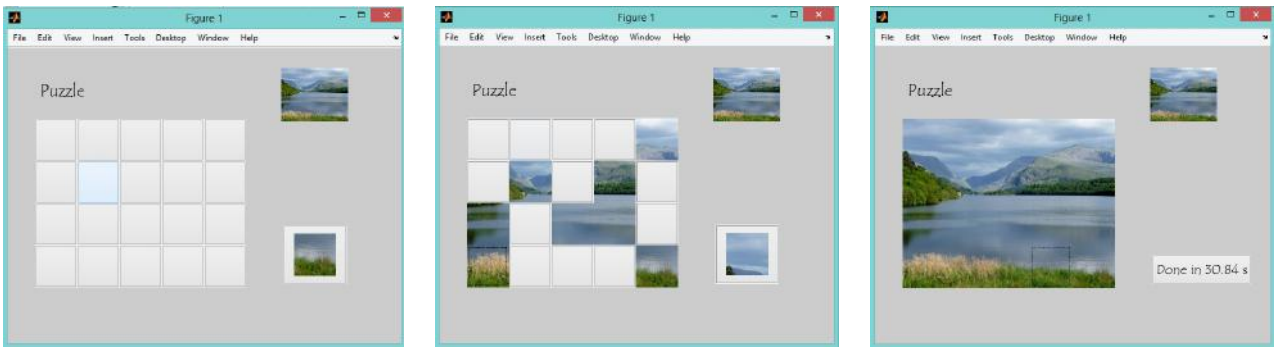


Figure 10.21: The GUI set-up for the moving car problem

19. Puzzle

    Construct a GUI as in Figure 10.22 (a).



(a) Start (b) Middle (c) End

Figure 10.22: The puzzle GUI.

Choose an image for the puzzle, upload it and resize it to [240, 300] pixels. Create a grid of 4×5 push buttons which will serve as the puzzle tiles.

Chop the image into 4 rows and 5 columns of 60×60 tiles. Chose a random sequence of showing the tiles, one at a time. Show the first tile in the window at the bottom right of the grid.

The player's task is to click on the grid button where the shown tile should be. Selecting a wrong button has no consequences. If the correct button is pressed, the tile should appear as the foreground of the button. An example of the mid-game play is shown in Figure 10.22 (b).

Upon placing the penultimate tile, complete the game by putting the last two tiles in place, and display the player's time taken to finish the puzzle. An example is shown in Figure 10.22 (c).

Chapter 11

Sounds

11.1 Sounds as Data

Sound is made up by waves which are often simplified to a sine function shown in Figure 11.1. The sine way is characterised by amplitude, frequency and phase. The phase is important when a sound contains more than one waves and they are offset by their phases. The amplitude determines how loud the sound is, and the frequency determines the pitch.

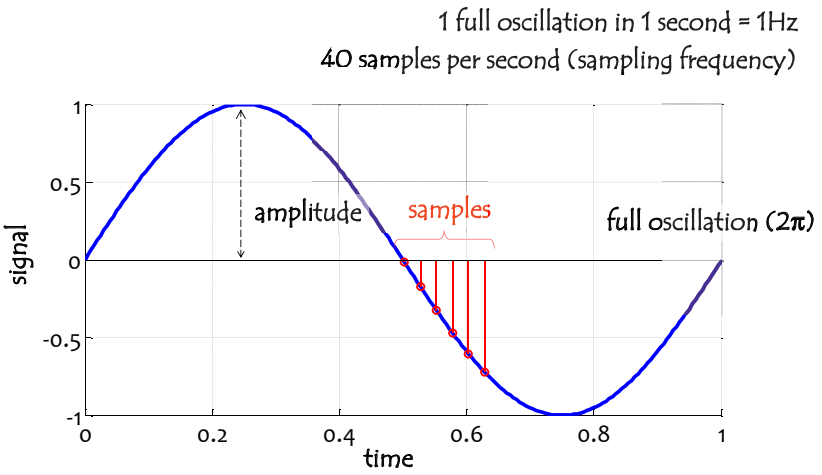


Figure 11.1: Sine wave

Usually sounds are much more complicated than a single sine wave, including many sine-like waves together. A sound can be reproduced if we find the sine waves it is made up from.

MATLAB command `wavread` reads Microsoft WAVE (".wav") sound file. Figure 11.2 plots two examples: a chain-saw sound and a 4-short-beeps sound.

Both examples in the figure are sampled with one of the standard frequencies, $fs = 8000$ (samples per second). *Note*: the difference between the *signal* frequency and the *sampling* frequency. The sampling frequency is the number of measurements of the signal that we take per second. The signal frequency is the number of full sine waves contained within one second (this is measured in a unit called Hertz, Figure 11.3).

A full piano keyboard is shown in Figure 11.4. The note frequencies are indicated as well.

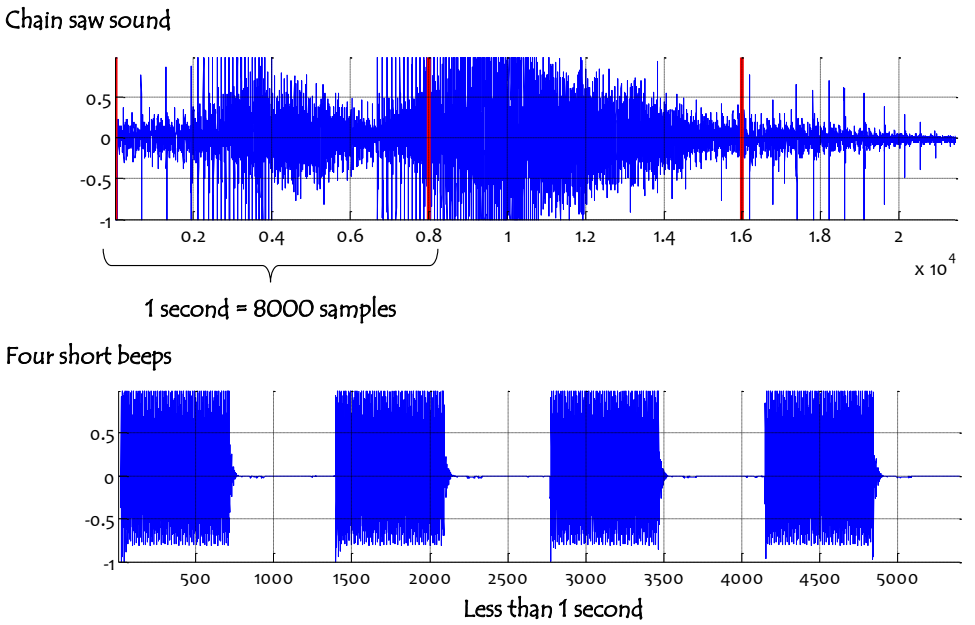


Figure 11.2: Two examples of sounds

Note frequencies can be calculated from a single frequency, which is usually A from first octave (A4), $f(A) = 440 = f_0$ Hz. The equation is:

$$f(n) = f_0 \times a^n$$

where n is the number of half steps away from A. For higher notes n is positive, and for lower notes, negative. $f(n)$ is the frequency of the note n half steps away. The constant is:

$$a = 2^{\frac{1}{12}} \approx 1.05946 \text{ Hz}$$

Sound can be created as a sine function and played in MATLAB using the `sound` command. The code below creates and plays middle A4 for 2 seconds.

```
fs = 8000; % the sampling frequency
T = 2;% length of the note in seconds
t = 0:1/fs:T;
F = 440; % frequency of A4
y = sin(F*2*pi*t); % the signal
sound(y,fs)
```

Consider another example, where C5 is played for 3 seconds while fading away. In this case the amplitude should gradually decrease to zero while the frequency will stay unaltered. The code is shown below. Note the element-wise multiplication where each value of the sine signal is multiplied by the respective amplitude value.

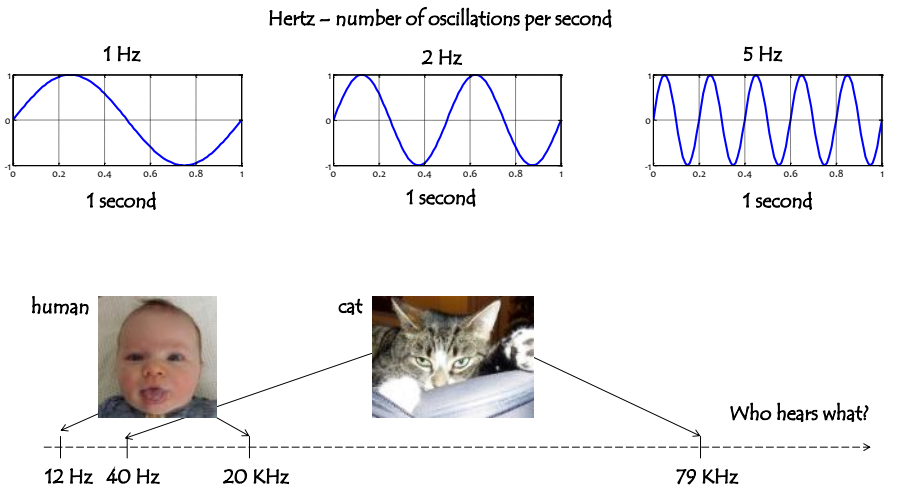


Figure 11.3: Illustration of Herz

```
fs = 8000; % the sampling frequency
T = 3;% length of the note in seconds
t = 0:1/fs:T;
F = 523.25; % frequency of C5
A = (T - t)/T;
y = A .* sin(F*2*pi*t); % the signal
sound(y,fs)
```

To produce a more natural tone, add overtones with smaller amplitudes. Add one overtone with twice the frequency, which will be an octave higher, and another overtone with half the frequency, which will be an octave lower. When running the sound command, MATLAB will clip all values of the signal outside the range $[-1, 1]$. Therefore, to include overtones, you should use amplitude 0.5 for the base signal and amplitude 0.25 for both overtones.

Look up the `play` command as an alternative to `sound`.

11.2 Exercises

1. Utter Noise

   Carry out the sequence of tasks below:

   (a) Create and play one second of 'utter noise' (random values in the interval $[-1, 1]$).

   (b) Insert a pause (zero values) in the interval [0.45, 0.55] of the second.

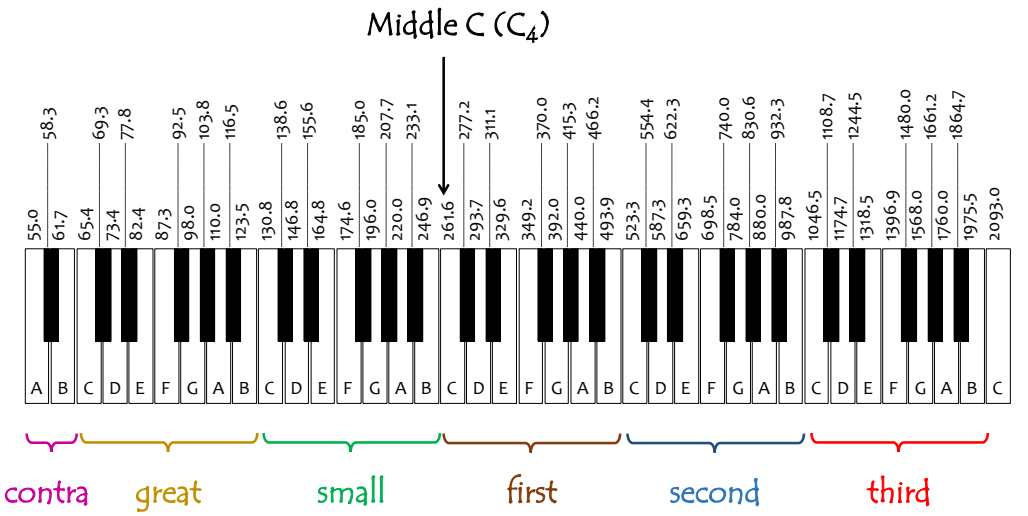


Figure 11.4: Note frequencies

(c) Alter the noise signal to quieten between 0s and 0.45s and progressively raise in amplitude between 0.55s and 1s.

2. Piano Keyboard

Create a working piano keyboard. All the keys from C4 to C5 should be shown (both the white and the black ones). Upon pressing a key with the mouse, the respective tone should be played for 1 second, with the volume fading away. An example of the graphical output is shown in Figure 11.5.

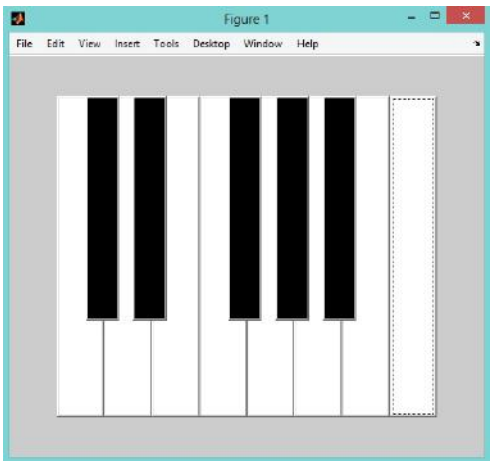


Figure 11.5: Piano keyboard C4 to C5

start CHALLENGE

Shortest piano code

Write the script for the Piano Keyboard problem using the minimum possible number of lines. The rules are: (1) Each row has a maximum of 75 symbols. (2) The number of characters does not matter. (3) The figure must be opened with the `figure` command. (The authors' current record is 5 lines.) Best of luck!

end CHALLENGE

3. Find and download a Wave file of your choice, for example a police siren. Read the signal into MATLAB using `wavread`. Modify the signal so that it starts from silence, amplifies to a maximum and then fades away to silence again.

4. Musical Scales

   (a) Write a script to play an ascending musical scale. Each note should be played for 0.8 seconds and should fade linearly. Include overtones.

   (b) Add a second melody playing the scale backwards (descending), four times quieter than the leading melody. The harmony should be played together.

   (c) Modify the melody you created in (b) so that it 'speeds up'. For example, if the first note lasts 1 second, the last one should last 0.1 seconds.

   (d) Plot the first 30 milliseconds of the sound signal in (b). Label the axes and add a title. The plot should look like the one in Figure 11.6.

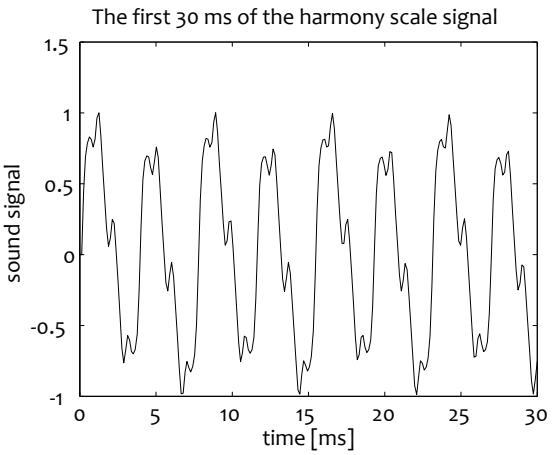


Figure 11.6: Expected signal shape for problem 4

5. Horror Movie Music

   Write a script that will compose a piece of music for a horror movie. Here are the rules:

(i) The music should be created with a significant random element in it. This means that you can re-use parts of the 'melody' but there should be random sampling as well.

(ii) All sounds should have proper frequencies, that is frequencies corresponding to existing notes. The array below contains the allowed frequencies:

```
Note = [246.94 261.63 277.18 293.66 311.13 329.63 349.23 369.99 392.00 415.30...
     440.00 466.16 493.88 523.25];
```

(iii) The length of the notes should be related to the 'beat'. If a beat has length h seconds, your notes may be of size $0.25h$, $0.5h$, h, $1.5h$ $2h$ and $3h$. The total length of your piece should be about 30 seconds. Choose randomly from the lengths and the frequencies until the total length reaches 30 seconds. Use beat $h = 0.5$s.

(iv) Use pauses (zeros) in the signal. They may have any of the lengths that other notes may have.

(v) Use fading sounds.

(vi) Save the signal. For example, you can use:

```
save('Horror_Music','y','fs')
```

where y is the variable containing your signal and fs is the sampling frequency. This line will save variables y and fs in a `mat` file. If you want to play the piece again later, load the file and then use `sound(y,fs)`.

6. What Does Music Look Like?

   Write the function `see_music` to visualise a music piece. The function should take one input parameter, the signal y. Split the signal into T equal intervals (for example, $T = 2000$). For each interval, calculate an approximation of the pitch (main frequency of the sound) by finding how many times the signal crosses the x-axis. Plot the T pitches in a complete circle, as shown in Figure 11.7 for the simple scales. Each interval should be a spoke with length proportional to the approximated pitch. The pitch does not have to be one of the note frequencies; it will only serve to trace the pattern of the music piece. The earliest spokes should be dark. The colour should lighten progressively to a colour of your choice.

   Figure 11.8 shows examples of visualisation of five music pieces. (Note that the visualisation is not required to be animated.)

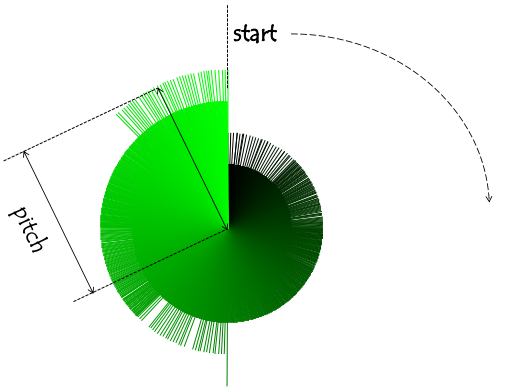
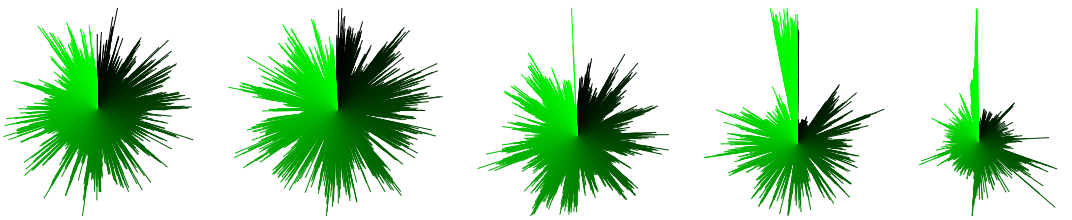


Figure 11.7: Visualisation of the scale (one octave)



(a) (b) (c) (d) (e)

(a) Wolfgang Amadeus Mozart, Piano Sonata No.16 in C Major K.545 (Sonata Facile)
(b) Wolfgang Amadeus Mozart, Serenade no.13 in G major K.525 (Eine Kleine Nachtmusik)
(c) Johann Sebastian Bach, Air on the G String
(d) Johann Sebastian Bach, Ave Maria, harp and violin.
(e) Dubstep, 'I am waiting for you last summer - Medley season'

Figure 11.8: Music pictures.

Chapter 12

Solutions

These are the solutions of problems with even numbers.

Chapter 1: Getting Started

• 1.4.2

Type in the command window `help imagesc`. MATLAB will display:

```
imagesc Display image with scaled colors
   imagesc(...) is the same as IMAGE(...) except the data is scaled
   to use the full colormap.
```

• 1.4.4

Type in the command window:

```
sqrt((4.172+9.131844)^3-18)/(-3.5+(11.2 - 4.6)*(7-2.91683)^-0.4)
```

The MATLAB output is: 186.1859

• 1.4.6

Type in the command window `log(exp(10))` and then `exp(log(10))`. Both expressions should return the value 10.

• 1.4.8

Denote the left-hand side of the equation by $f(x)$. Starting with the middle of the interval ($x = 3$) find out in which half the solution lie. For example, check $x = 2$ next. As both $f(3)$ and $f(2)$ are negative, the solution must be in $[3, 4]$. Then keep dividing (and guessing, if you like) to shorten the interval of the solution until the interval length is 0.1. Return the x such that $f(x)$ and $f(x \pm 0.1)$ have different signs.

```
>> x=3;
>> 0.5*(x-2)^3 - 40*sin(x)
ans =
-5.1448
>> x=2;
>> 0.5*(x-2)^3 - 40*sin(x)
ans =
```

```
-36.3719
>> x=3.5;
>> 0.5*(x-2)^3 - 40*sin(x)
ans =
15.7188
>> x=3.25;
>> 0.5*(x-2)^3 - 40*sin(x)
ans =
5.3044
>> x = 3.12;
>> 0.5*(x-2)^3 - 40*sin(x)
ans =
-0.1612
>> x = 3.18;
>> 0.5*(x-2)^3 - 40*sin(x)
ans =
2.3574
>> x = 3.14;
>> 0.5*(x-2)^3 - 40*sin(x)
ans =
0.6771
>> x = 3.13;
>> 0.5*(x-2)^3 - 40*sin(x)
ans =
0.2578
```

Since $f(3.12) < 0$ and $f(3.13) > 0$, the solution lies between the two values. Therefore we can return either of them, say, $x = 0.12$.

Chapter 2: MATLAB: The Matrix Laboratory

• 2.7.2

We have not studied loops thus far, therefore use:

```
E = {eye(1),eye(2),eye(3),eye(4),eye(5),eye(6),eye(7),eye(8)};
```

• 2.7.4

First, create the vector with all integers from 1 to 100 by the colon operator and then apply the command 'sum'.

```
sum(1:100)
```

• 2.7.6

Create arbitrary matrices that can be multiplied as required (ABC) and calculate the two expressions. The results should match.

```
A = [2 3 1;-3 2 5]; % 2-by-3
```

```
B = rand(3); % random 3-by-3
C = [3 1;-2 4;5 -6]; % 3-by-2
disp((A*B*C)')
disp(C'*B'*A')
```

- 2.7.8

```
v = 20:25;
w = 5*v;
```

- 2.7.10

```
a = linspace(1,2*pi,100);
```

- 2.7.12

First, create a zero matrix A of size 100-by-100 and then replace all values in the even numbered columns with value 2.

```
A = zeros(100);
A(:,2:2:100) = 2;
```

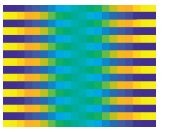
- 2.7.14

```
A = ones(10)*8; % 10-by-10 matrix of 8s
A(3:8,3:8) = 0; % inset a 6-by-6 matrix of 0s
A(5:6,5:6) = 3; % inset a 2-by-2 matrix of 3s
figure, imagesc(A), axis equal off
```

- 2.7.16

There are many ways to construct these matrices. For part (a), the matrix is constructed by repeating a tile of 2 rows. The top row are the numbers from 1 to N, and the second row are the numbers from N down to 1.

```
N = input('Number of columns: ');
tile = [1:N;N:-1:1];
A = repmat(tile,8,1); % 8 tiles = 16 rows
figure, imagesc(A), axis equal off
```



For part (b), the matrix is concatenated from four parts. Then the central pixels are assigned the same value, different from the values of the four parts.

```
A = [ones(5), ones(5)*2;ones(5)*3, ones(5)*4];
A(3,3) = 0; A(8,3) = 0; A(3,8) = 0; A(8,8) = 0;
figure, imagesc(A), axis equal off
```

In part (c), we need to address the edges suitably so that the corners hold the respective colour.

```
M = input('Matrix size: ');
A = zeros(M);
A(1,1:M-1) = 1;
A(1:M-1,M) = 2;
A(M,2:M) = 3;
A(2:M,1) = 4;
figure, imagesc(A), axis equal off
```

• 2.7.18

To find the solution using matrix equation, we should recall that we need the matrix with the coefficients in frond of the variables (A) and the vector with the right-hand-side values ($\mathbf{b}$):

$$A = \begin{bmatrix} 7 & -12 \\ 12 & -45 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} -4 \\ -26 \end{bmatrix}.$$

The solution is $\mathbf{x} = A^{-1}\mathbf{b}$.

```
A = [7 -12;12 -45]; % coefficients
b = [-4;-26]; % right-hand side vector
x = inv(A)*b; % solution [x;y]
disp(x)
```

For this problem, the solution is $x = 0.7719$, $y = 0.7836$.

• 2.7.20

Suppose $m = 4$ and $n = 3$. Using 'meshgrid', we can create all row and column indices i and j:

```
[cols,rows] = meshgrid(1:n,1:m);
```

The results are

```
rows =

1     1     1
2     2     2
3     3     3
4     4     4

>> cols

cols =

1     2     3
1     2     3
```

```
1     2     3
1     2     3
```

Notice that the columns are the x-coordinate, therefore they are the first output arguments of 'meshgrid', and the rows are the second output argument. With these two arrays in place, the code is:

```
m = 4; n = 3;
[cols,rows] = meshgrid(1:n,1:m);
A = (cols - 4).^2 .* (rows + 1).^-3 + rows.*cols;
```

The resultant matrix A is

```
A =

2.1250    2.5000    3.1250
2.3333    4.1481    6.0370
3.1406    6.0625    9.0156
4.0720    8.0320   12.0080
```

Just to be sure, let's check with one of the values, $A(2,3)$

$$A(2,3) = (3-4)^2(2+1)^{-3} + 2 \times 3 = 6.0370.$$

• 2.7.22

To calculate the x and y for part (a), we can use

```
x = [0:10 10:-1:0]; % bottom x's followed by top x's (reverse)
y1 = [repmat([1 0],1,5) 1]; % bottom y's
y2 = y1 + 3; % top y's (no need to be reversed)
y = [y1 y2]; % put the y's together
%
% Bonus (we haven't studied plotting at this point)
figure, plot(x,y,'b.-'), grid on
```

For part (b), y goes from 1 to 40 while x goes forth and back. We can create one of the upward lines of x's, and shift it by 10 to obtain the other. Then we need to merge them like the teeth of two cog wheels:

```
x1 = 1:20; % left
x2 = x1 + 10; % right
x(1:2:40) = x1; % insert the left x's (odd)
x(2:2:40) = x2; % insert the right x's (even)
y = 1:40;
figure, plot(x,y,'b.-'), grid on
```

Part (c) is quite similar to part (b). This time y oscillates between 0 and 1. One possible solution is:

```
x1 = 1:20; % left
x2 = x1 + 10; % right
x(2:2:40) = x1; % insert the left x's (odd)
x(1:2:40) = x2; % insert the right x's (even)
y = repmat([0 1],1,20);
figure, plot(x,y,'b.-'), grid on
```

Chapter 3: Logical Expressions and Loops

- 3.6.2

```
disp(37:37:1000) % first way

find(~mod(1:1000,37)) % second way

% third way
x = 1:1000;
disp(x(floor(x/37)==ceil(x/37)))

% fourth way
i = 2;
z = 37;
while z < 1000
    disp(z)
    z = 37*i;
    i = i + 1;
end
```

- 3.6.4

```
z = rgb2gray(imread('peppers.png')); figure, imshow(z)
z(z<=100) = 0; % black
z(z>100&z<200) = 150; % light grey
z(z>=200) = 255; % white
figure, imshow(z)
```

- 3.6.6

To check your solution, plug the logical expression you created in the code below, as indicated. For this wrapper script to work, you need to make sure that your expression allows for vector variables x and y.

```
figure, hold on, grid on, axis square
x = rand(10000,1)*10; y = rand(10000,1)*10;
ind = <YOUR EXPRESSION>;
plot(x(ind),y(ind),'k.','markersize',10)
plot(x(~ind),y(~ind),'g.','markersize',10)
```

The solutions are:

```
ind = x>3 & x<8 & y<4 & y>1; % (a)
ind = (x-3).^2 + (y-8).^2 < 4; % (b)
ind = x < y; % (c)
ind = 2*x + 3*y  -18< 0; % (d)
ind = xor(((x-6).^2 + (y-6).^2 < 4) , x > y); % (e)
ind = (8*x-3*y-13 > 0) & (-8*x-3*y+67>0) & (y > 1); % (f)
ind = xor(((8*x-3*y-13 > 0) &(-8*x-3*y+67>0)&(y > 1)),...
     ((x-5).^2 + (y-4).^2 < 1)); % (g)
ind = ~((x>4.5&x<5.5&y>1&y<9)|(x>1&x<9&y>4.5&y<5.5)); % (h)
```

- 3.6.8

```
NumberToGuess = randi(10);
UserGuess = input('Please enter your guess -> ');
if UserGuess == NumberToGuess
    disp('Congratulations! You won!')
else
    disp('You lost! Better luck next time! The number was')
    disp(NumberToGuess)
end
```

- 3.6.10

For problem (a), pad array A with one cell on each edge so that each cell of A has 8 neighbours. Then construct a double loop to go through the rows and the columns of A.

```
m = 20; n = 30;
A = rand(m,n) < 0.1; % sparse
% A = rand(m,n) < 0.5; % medium
% A = rand(m,n) < 0.7; % dense

B = zeros(m+2,n+2); % padding
B(2:end-1,2:end-1) = A; % inset A

S = 0; % sum of neighbours
for i = 2:m+1
    for j = 2:n+1
        if B(i,j) % bug
            S = S + sum(sum(B(i-1:i+1,j-1:j+1))) - 1; % only neighbours
        end
    end
end
disp('Averge number of neighbours per bug:')
disp(S/sum(B(:)))
```

Uncomment the other versions of creating A to see how the average number of neighbour bugs changes.

Part (b) can be programmed by checking the neighbourhood for each cell and applying the rule that fits. The important trick here is not to destroy the current grid A while calculating the 'tomorrow's grid. This is why we use G to store the tomorrow's bugs calculated from the neighbourhood in A.

```
A = rand(m,n) < 0.7;
B = zeros(m+2,n+2); % padding
G = B; % the new generation
B(2:end-1,2:end-1) = A; % inset A
for i = 2:m+1
    for j = 2:n+1
        on = sum(sum(B(i-1:i+1,j-1:j+1))) - B(i,j); % only neighbours
        if (B(i,j) && (on == 2 || on == 3)) || (~B(i,j) && on == 3)
            G(i,j) = 1;
        end
    end
end
A = G(2:m-1,2:n-1); % the new generation
```

Part (c) requires only to include the code from part (b) in loop.

```
A = rand(m,n) < 0.3;
B = zeros(m+2,n+2); % padding
B(2:end-1,2:end-1) = A; % inset A
figure, spy(B), axis off
figure
for k = 1:50
    G = zeros(m+2,n+2); % the clean grid for the new generation
    for i = 2:m+1
        for j = 2:n+1
            on = sum(sum(B(i-1:i+1,j-1:j+1))) - B(i,j); % only neighbours
            if (B(i,j) && (on == 2 || on == 3)) || (~B(i,j) && on == 3)
                G(i,j) = 1;
            end
        end
    end
    B = zeros(m+2,n+2); % padding
    B(2:end-1,2:end-1) = G(2:m+1,2:n+1); % new generation
    spy(B), axis off
    pause(0.2)
end
```

The glider gun in part (d) is the code in part (c) initialised with the pattern of bugs shown in Figure 3.9.

```
m = 25; n = 40;
A = zeros(m,n);
gb = [6,2;6,3;7,2;7,3;6,12;7,12;8,12;5,13;9,13;4,14;10,14;4,15;10,15;...
7,16;5,17;9,17;6,18;7,18;8,18;7,19;4,22;5,22;6,22;4,23;5,23;6,23;...
```

```
3,24;7,24;2,26;3,26;7,26;8,26;4,36;5,36;4,37;5,37]; % the gun bugs
A(sub2ind([m,n],gb(:,1),gb(:,2))) = 1; % position the gun bugs
B = zeros(m+2,n+2); % padding
B(2:end-1,2:end-1) = A; % inset A
figure, spy(B), axis off
figure
for k = 1:250
    G = zeros(m+2,n+2); % the clean grid for the new generation
    for i = 2:m+1
        for j = 2:n+1
            on = sum(sum(B(i-1:i+1,j-1:j+1))) - B(i,j); % only neighbours
            if (B(i,j) && (on == 2 || on == 3)) || (~B(i,j) && on == 3)
                G(i,j) = 1;
            end
        end
    end
    B = zeros(m+2,n+2); % padding
    B(2:end-1,2:end-1) = G(2:m+1,2:n+1); % new generation
    spy(B), axis off
    pause(0.02)
end
```

Chapter 4: Functions

- 4.6.2

```
function D = euclidean_distance_arrays(A,B)
for i = 1:size(A,1)
    x = A(i,:); % ith row of A
    for j = 1:size(B,1)
        y = B(j,:); % jth row of B
        D(i,j) = sqrt(sum((x-y).^2));
    end
end
```

Here is a solution without loops:

```
function D = euclidean_distance_arrays2(A,B)
N = size(A,1); M = size(B,1);
AA = repmat(A,M,1); BB = repmat(B',1,N)';
D = reshape(sqrt(sum((AA-BB).^2,2)),M,N);
```

Check with:

```
P = [3 4;1 2]; Q = [-2 5;3 -1; 7 4];
disp(euclidean_distance_arrays(P,Q))
```

- 4.6.4

```
function is_in = point_in_a_square(x,y,p,q,s)
is_in = x >= p & x <= p + s & y >= q & y <= q + s;
```

Here we assume that if the point is on the edge or corner, it is in the square. Check with this example:

```
point_in_a_square(0.3,0.8,0,0,1)
point_in_a_square(0.3,1.8,0,0,1)
point_in_a_square(1,0.8,0,0,1)
```

The first and the third points are in, and the second is out of the unit square.

• 4.6.6

```
function o = fibo_recursive(k)
if k < 2
    o = k;
else
    o = fibo_recursive(k-1) + fibo_recursive(k-2);
end
```

Check with:

```
for i = 1:10
    disp(fibo_recursive(i))
end
end
```

• 4.6.8

Bubble sort operates by swapping neighbouring elements in the array if they are not in the right order. It finishes when no swaps are made passing through the whole array.

```
function A = bubble_sort(A)
SWAP = true;
i = 1;
N = numel(A);
while SWAP
    SWAP = false;
    for j = 1 : N - i
        if A(j) > A(j+1)
            A([j,j+1]) = A([j+1,j]);
            SWAP = true;
        end
    end
    i = i + 1;
end
```

Example of applying the algorithm to an array:

```
>> a = randi(500,1,10) - 250
a =
92   131   -37  -126  -154   -90   -50   -82  -122    44
>> bubble_sort(a)
ans =
-154  -126  -122   -90   -82   -50   -37    44    92   131
```

Chapter 5: Plotting

• 5.3.2

The function for plotting a circle:

```
function plot_circle(x,y,r,c)
theta = linspace(0,2*pi,100);
fill(x+sin(theta)*r,y+cos(theta)*r,c)
```

The script:

```
figure, hold on, axis equal off
for i = 1:30
    plot_circle(rand,rand,rand*0.2,rand(1,3))
end
```

• 5.3.4

```
figure, hold on
for i = 1:20
    ver = randi([3,6]);
    fill(rand(ver,1),rand(ver,1),rand(1,3))
end
axis([0.2 0.8 0.2 0.8])
axis square off
```

• 5.3.6

```
function nested_squares(k)
hold on
for i = 1:k
    w = k-i+1;
    fill([w,w,-w,-w],[-w,w,w,-w],rand(1,3))
end
axis equal off
```

• 5.3.8

```
figure, hold on
k = 10; x = [0; rand(k-1,1)]; y = [0; rand(k-1,1)];
% vertex (0,0) is needed for touching in the centre
```

```
co = rand(1,3); % fill colour
fill(x,y,co); fill(-x,y,co);fill(x,-y,co);fill(-x,-y,co);
axis equal off
```

- 5.3.10

(a)

```
figure, hold on
ind = 1;
h = zeros(1,25);
for i = 1:5
    for j = 1:5
        h(ind) = fill(rand(1,6)+0.9*i,rand(1,6)+0.9*j,rand(1,3));
        ind = ind + 1;
    end
end
axis equal off
```

(b)

```
figure
subplot(1,2,1), hold on
ind = 1;
X = rand(25,6); Y = rand(25,6); C = rand(25,3); % the forms
for i = 1:5
    for j = 1:5
        fill(X(ind,:)+0.9*i,Y(ind,:)+0.9*j,C(ind,:));
        ind = ind + 1;
    end
end
axis equal off
title('Original')

subplot(1,2,2), hold on
% permute the figures
rp = randperm(25); X = X(rp,:); Y = Y(rp,:); C = C(rp,:);
ind = 1;
for i = 1:5
    for j = 1:5
        fill(X(ind,:)+0.9*i,Y(ind,:)+0.9*j,C(ind,:));
    ind = ind + 1;
    end
end
axis equal off
title('Shuffled')
```

- 5.3.12

```
% (a)
figure, hold on
T = 5;
for i = 1:T
    plot([0 i 0 -i 0],[-i 0 i 0 -i],'k-','color',[i 0 0]/T,'linewidth',5)
end
axis equal off

% (b)
figure, hold on
T = 100;
for i = 1:T
    if i <= T/2
        plot([0 i 0 -i 0],[-i 0 i 0 -i],'k-',...
            'color',2*[0 i 0]/T,'linewidth',5)
    else
        plot([0 i 0 -i 0],[-i 0 i 0 -i],'k-',...
            'color',[0 0 i-T/2]/(T/2),'linewidth',5)
    end
end
axis equal off
```

• 5.3.14

(a)

```
function draw_balloon(x,y,r,c,l)
t = linspace(0,2*pi,100);
fill(sin(t)*r + x, cos(t)*r + y, c) % draw balloon

b = rand*2*pi; % random phase of the string
zy = linspace(0,l,50);
zx = linspace(0,3*rand*pi+2*pi,50);
plot(x+sin(zx+b)*0.15*r-sin(b)*0.15*r,...
y-r-zy,'k-') % draw string, amplitude 0.15*r

p = 0.08*r; % the blower triangle offset
fill([x-p x+p x],[y-r-p y-r-p y-r],c) % draw blower
plot(x,y-r,'k.')

% draw light reflection
l1 = 80; l2 = 92;
fill([sin(t(l1:l2))*0.8*r + x, sin(t(l2:-1:l1))*0.68*r + x], ...
[cos(t(l1:l2))*0.8*r + y,cos(t(l2:-1:l1))*0.6*r + y],...
'w','EdgeColor','none')
```

(b)

```
figure, hold on, axis equal off
for i = 1:20
```

```
    draw_balloon(rand, rand, rand*0.1+0.05, rand(1,3), rand*0.3+0.3)
end
```

(c)

```
figure, hold on, axis equal off
for i = 1:20
    ra = rand*0.1+0.05; % radius
    draw_balloon(rand, 1-ra, ra, rand(1,3), rand*0.3+0.3)
end
v = axis;
fill([v(1) v(2) v(2) v(1)],[v(4) v(4) v(4)+0.1 v(4)+0.1],[0.7 0.7 0.7])
```

Chapter 6: Data

- 6.4.2

```
a = randn(30,1)*20 + 100;
```

- 6.4.4

(a) Generate a random number k between -30.4 and 12.6.

```
k = rand * (12.6 - (-30.4)) - 30.4;
```

Note that the expression in the parentheses can be calculated as a single constant. The expression was left in this form here for readability. The random number should be multiplied by the (max - min) and then the minimum should be added.

(b) Generate an array A of size 20-by-20 of random integers in the interval $[-40, 10]$. Subsequently, replace by 0 all elements of A which are smaller than k.

```
A = randi([-40, 10],20);
A(A < k) = 0;
```

(c) Find the mean of all non-zero elements of A.

```
mnzA = mean(A(A~=0));
```

(d) Pick a random element from A.

```
rndA = A(randi(numel(A)));
```

(e) Visualise A using a random colour map containing exactly as many colours as there are different elements of A.

```
figure, imagesc(A), axis equal off
c = numel(unique(A)); % how many colours are needed
colormap(rand(c,3))
```

(f) Extract 4 different random rows from A and save them in a new array B.

```
r = randperm(size(A,1),4); % rows # to extract
B = A(r,:);
```

(g) Find the proportion of non-zero elements of B.

```
propnzB = mean(B(:)~=0); % B is reshaped into a vector
```

(h) Display in the Command Window the answers of (a), (c), (d) and (g) with a proper description of each one.

```
disp('A random number between -30.4 and 12.6:'), disp(k)
disp('Mean of all non-zero elements of A:'), disp(mnzA)
disp('A random element of A:'), disp(rndA)
disp('Proportion of non-zero elements of B:'), disp(propnzB)
```

- 6.4.6

1. Generate an array of 10,000 random outcomes of the three slots of the machine.

```
outcomes = randi(6,10000,3);
```

2. Find the total number of winning combinations among the 10,000 outcomes.

```
wins = sum(outcomes(:,1) == outcomes(:,2) & ...
    outcomes(:,1) == outcomes(:,3));
```

3. Assume that the entry fee for each run is 1 unit of some imaginary currency. Each winning combination is awarded a prize of 10 units except for the combination of three 1s, which is awarded a prize of 50. Assuming you are the owner of the slot machine, calculate your profit after the 10,000 runs of the game.

```
win_index = outcomes(:,1) == outcomes(:,2) & ...
     outcomes(:,1) == outcomes(:,3);
profit = 10000 - sum((win_index & outcomes(:,1) ~= 1) * 10) - ...
     sum((win_index & outcomes(:,1) == 1) * 50);
disp('Profit = '), disp(profit)
```

- 6.4.8

```
a = randn(5000,2);
ind = sqrt(a(:,1).^2+a(:,2).^2) < 0.7 | sqrt(a(:,1).^2+a(:,2).^2) > 1.5;
figure
plot(a(ind,1),a(ind,2),'g.')
axis equal, grid on
```

- 6.4.10

```
a = randn(1,1000).*(1:1000)*0.2; % random signal with increasing amplitude
y_offset = linspace(-400,200,1000); % increasing slope to add
s = a + y_offset; % the signal
[mins,ind_mins] = min(s);
[maxs,ind_maxs] = max(s);

figure, hold on, grid on
xlabel('time'), ylabel('amplitude')
plot([ind_mins, ind_maxs],[mins maxs],'ys','markersize',20,...
'MarkerFaceColor','y')
% plot the min and the max first so that the signal plot goes over
plot(s,'k-')
plot([0 1000],[-400, 200],'r-','linewidth',1.5)
axis([0 1000 -600 800])
```

- 6.4.12

```
figure, grid on
fill([0 1 1 7 7 5 5 0],[1 1 8 8 9 9 5 5],[0.7 0.7 0.7])
x = rand(1000,1) * 7;
y = rand(1000,1) * 8 + 1;
index = (x > 0) & (x < 1) & (y > 1) & (y < 5) | ...
    (x > 1) & (x < 5) & (y > 5) & (y < 8) | ...
    (x > 5) & (x < 7) & (y > 8) & (y < 9);
hold on
plot(x(index),y(index),'k.')
```

- 6.4.14

```
x = rand(2000,1) * 200 - 35;
y = rand(2000,1) * 100 - 20;
index1 = ((x - 30).^2 + (y - 40).^2 < 900) | ...
    ((x + 10).^2 + y.^2 < 1600);
index2 = ((x - 30).^2 + (y - 40).^2 < 64) | ...
    ((x + 10).^2 + y.^2 < 64);
figure, hold on
plot(x(~index1&~index2),y(~index1&~index2),'k.','markersize',15,...
    'color',[0.7 0.7 0.7])
plot(x(index1&~index2),y(index1&~index2),'rx','linewidth',2.5,...
    'markersize',10)
plot(x(index2),y(index2),'g+','linewidth',2.5,'markersize',10)
```

```
    grid on, axis equal tight
```

• 6.4.16

(a)

```
k = 10; % range of the matrix entry
T = 10000; % number of iterations
count = 0;
for i = 1:T
    m = randi([-k k],2);
    if det(m) == 0 % singularity check
        count = count + 1;
    end
end
fprintf(['Proportion of singular 2x2 integer-valued matrices in ',...
    '[-%i,%i]: %.4f\n'],k,k,count/T)
```

(b)

```
c = zeros(1,50);
for k = 1:50
    count = 0;
    for j = 1:T
        m = randi([-k k],2);
        if det(m) == 0
            count = count + 1;
        end
    end
    c(k) = count;
end
figure
plot(c/T,'k-')
grid on
title('Proportion of singular 2x2 matrices with integer entries in [-k,k]')
xlabel('k')
```

The output is shown in Figure 12.1. The proportion of singular 2-by-2 matrices with integers between $-k$ and k decreases exponentially with increasing k.

• 6.4.18

(a)

```
% checking for three-of-a-kind
Values = {'1','2','3','4','5','6','7','8','9','10','J','Q','K'};
Suits = {'C','D','H','S'};
CardValue = zeros(1,5);
k = 0; % counter
while sum(CardValue==max(CardValue)) ~= 3 || numel(unique(CardValue)) ~=3
    k = k + 1;
```

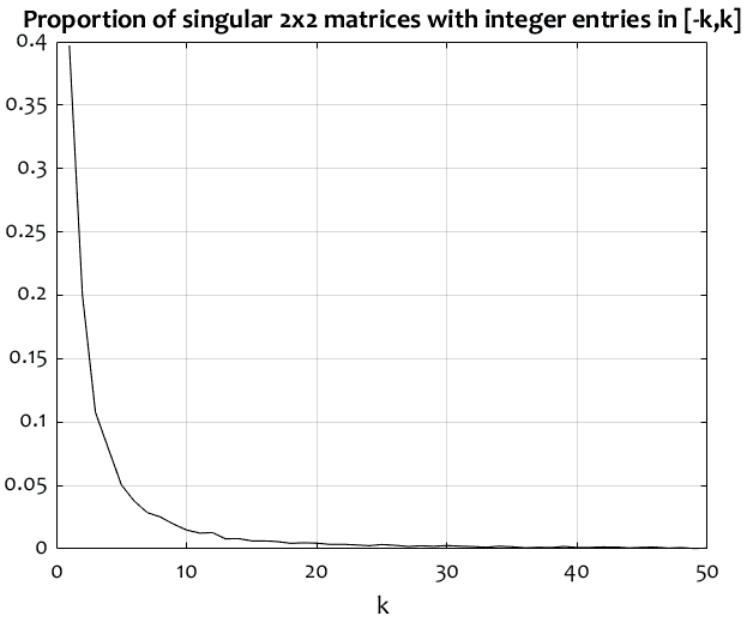


Figure 12.1: Output for problem 6.4.16 (b)

```
    rp = randperm(52); % generate a new hand
    hand = rp(1:5);
    CardSuit = ceil(hand/13); % suit index
    FrequenciesOfCards = zeros(1,13);
    CardValue = mod(hand,13) + 1;
end
for i = 1:5 % display the hand
    fprintf('%s%s ',Values{CardValue(i)},Suits{CardSuit(i)})
end
% print the number of trials
fprintf('\n\nNumber of trials before 3-of-a-kind = %d\n\n',k)
```

(b)

```
% evaluating a hand
Values = {'A','2','3','4','5','6','7','8','9','10','J','Q','K'};
Suits = {'C','D','H','S'};
ph = {'high card','one pair','two pairs','three of a kind','straight',...
     'flush','full house','four of a kind','straight flush'};

%    1 high card (none of the following)
%    2 one pair
%    3 two pair
%    4 three of a kind
%    5 straight (consecutive cards, mixed suits)
%    6 flush (same suit, any value)
%    7 full house (three of a kind and a pair)
%    8 four of a kind
```

```
%    9 straight flush (consecutive cards, same suit)

rp = randperm(52); % generate a new hand
hand = rp(1:5);
mask = zeros(13,4);
mask(hand) = 1;
ranking = zeros(9,1);
M = sum(mask,2); % how many of each value
mM = max(M); % largest number of equal values
if mM == 4
    ranking(8) = 1; % four
elseif (mM == 3)
    if isempty(find(M == 2)) %#ok<*EFIND>
        ranking(4) = 1; % three
    else
        ranking(7) = 1; % full house
    end
else
    if sum(M==2) == 2
        ranking(3) = 1; % two pair
    elseif sum(M==2) == 1
        ranking(2) = 1; % one pair
    end
end
if max(sum(mask)) == 5 % all of the same suit
    if max(find(M)) - min(find(M)) == 4 %#ok<MXFND> % consecutive
        ranking(9) = 1; % straight flush
    else
        ranking(6) = 1; % flush
    end
else
    if mM == 1 % all different and different suits
        if (max(find(M)) - min(find(M)) == 4) ... % consecutive
            || all(find(M') == [2 3 4 5 13]) % A 2 3 4 5
            ranking(9) = 1; % straight
        end
    end
end
% After all ranking 2:9 have been explored
if sum(ranking) == 0 %(none of the fancy hands)
    ranking(1) = 1; % high card hand
end
CardSuit = ceil(hand/13); % suit index
CardValue = mod(hand,13) + 1;
for i = 1:5 % display the hand
    fprintf('%s%s ',Values{CardValue(i)},Suits{CardSuit(i)})
end
fprintf('\n')
r = find(ranking);
```

```
disp(ph{r})
```

- 6.4.20

(a)

```
cx = randi(10); cy = randi(10); r = randi(10);
bx = randi(10); by = randi(10); w = randi(10); h = randi(10);
```

(b)

```
figure, hold on, axis equal, grid on
theta = linspace(0,2*pi,100);
plot(sin(theta)*r+cx,cos(theta)*r+cy,'b-','linewidth',2) % plot circle
Rx = [bx,bx+w,bx+w,bx,bx]; Ry = [by,by,by+h,by+h,by];
plot(Rx,Ry,'b-','linewidth',2) % plot rectangle
```

(c)

```
% check whether to calculate or simulate
R = [Rx' Ry'];
T = [];
if all(sum((R - repmat([cx,cy],5,1)).^2,2) < r^2) % rectangle within
    ar = w * h; % area
elseif cx-r>=bx && cx+r<=bx+w && cy-r>=by && cy+r<=by+h % circle within
    ar = pi * r^2;
else
    T = 30000; % number of points for the Monte Carlo simulation
    % generate points within the tight square around the circle
    x = rand(T,1)*2*r+cx-r; y = rand(T,1)*2*r+cy-r;
    % find the number within the intersection
    in_rectangle = x>bx & x<bx+w & y>by & y<by+h;
    in_circle = (x-cx).^2 + (y-cy).^2 - r^2 < 0;
    ind = in_rectangle & in_circle;
    number_in = sum(ind);
    ar = number_in/T * (2*r)^2;
end
```

(d)

```
figure, hold on, axis equal, grid on
if T % area has been estimated (not calculated)
   plot(x,y,'k.','markersize',2)
   plot(x(ind),y(ind),'r.','markersize',8)
end
title(['Area = ',num2str(ar)])
plot(sin(theta)*r+cx,cos(theta)*r+cy,'b-','linewidth',2) % plot circle
plot(Rx,Ry,'b-','linewidth',2) % plot rectangle
```

- 6.4.22

```
% The chromosome is a binary vector of length 25x25 matrix = 625.
% The fitness function = number of 1s in the chromosome; large is better.
% Start with a random population with 10 chromosomes.
% Use only mutation; set the mutation probability to 0.15.
% Run your algorithm for 20 generations.
gs = 25; % grid size
ps = 10; % population size
P = rand(ps,gs^2) > 0.5; % population
Pm = 0.15; % mutation probability
F = sum(P,2); % evaluation of P
figure
for i = 1:400 % up to 20 generations
    O = P; % offspring
    M = rand(size(O)) < Pm; % mutation mask
    O(M) = 1 - O(M); % mutate offspring
    FO = sum(O,2); % evaluation of offspring
    FA = [F;FO]; % concatenate the fitness
    A = [P;O]; % all chromosomes available
    [~,ind] = sort(FA,'descend');
    P = A(ind(1:ps),:); % selected population
    F = FA(ind(1:ps)); % corresponding fitness

    % At each new generation, plot the best chromosome in the current
    % population using the 'spy' command. Format the chromosome as a
    % 25x25 matrix. An ideal chromosome will have all spaces filled.
    % The worst chromosome will be an empty square in the figure.
    spy(reshape(P(1,:),gs,gs))
    title(['Best chromosome at iteration ' num2str(i)])
    drawnow
end
% At the end, print out the fitness value of the best chromosome,
% and show the chromosome as explained above.
figure, hold on
spy(reshape(P(1,:),gs,gs))
title(['Best chromosome''s fitness value: ' num2str(F(1))])
```

- 6.4.24

```
figure
P = rand(25) > 0.4;
directions = [1 0;0 1;-1 0;0 -1]; % for possible moves
k = 0; % counter of the steps
while sum(sum(P)) > 0 % bugs left
    W = zeros(25); % new canvas
    for i = 1:25
        for j = 1:25
            if P(i,j) % bug
                move = ceil(rand*4);
```

```
                if (i + directions(move,1) > 0) &&...
                        (i + directions(move,1) < 26) ...
                        && (j + directions(move,2) > 0) &&...
                        (j + directions(move,2) < 26)
                    % inside the grid
                    W(i + directions(move,1),j + directions(move,2)) = ...
                        W(i + directions(move,1),j + ...
                        directions(move,2)) + 1;
                end
            end
        end
    end
    W(W>1) = 0; % kill the multiple bugs in a cell
    P = W;
    spy(P)
    k = k + 1; % steps
    pause(0.05)
end
close
fprintf('Number of steps = %d\n',k)
```

- 6.4.26

The function:

```
function [p,d] = greedy_tsp(cities)

n = size(cities,1); di = zeros(n);
for i = 1:n-1
    for j = i+1:n
        di(i,j) = sum((cities(i,:) -  cities(j,:)).^2);
        di(j,i) = di(i,j);
    end
end
di = sqrt(di);

d = 0;
p = 1; % cities visited
cn = 2:n; % cities not visited yet
for i = 1:n-1
    [sd,next_city] = min(di(cn,p(end)));
    d = d + sd; % add the smallest distance sd
    p = [p, cn(next_city)]; % add the new city to the list
    cn(next_city) = []; % remove the new city from non-visited
end
d = d + di(p(end),1);
```

The calling script:

```
ci = rand(10,2); % cities' positions
```

```
[g,dg] = greedy_tsp(ci);
figure, hold on
plot([ci(g,1);ci(g(1),1)],[ci(g,2);ci(g(1),2)],...
    'go-','linewidth',2,'markersize',12);
plot([ci(g,1);ci(g(1),1)],[ci(g,2);ci(g(1),2)],...
    'ro','linewidth',2,'markersize',8);
axis square, grid on
title(['GREEDY Minimum d = ',num2str(dg)])
```

Chapter 7: Strings

• 7.4.2

```
s = ['Try to distinguish between e-mail addresses ending with "uk", '...
    'and those ending with something else. For example, check with ',...
    'n.o.body@fiction.co.uk and print out the result.'];
[first_part,second_part] = strtok(s,'@');
blanks_first_part = strfind(first_part,' ');
address1 = first_part(blanks_first_part(end)+1:end);
blanks_second_part = strfind(second_part,' ');
address2 = second_part(1:blanks_second_part(1)-1);
dots = strfind(address2,'.');
network_extension = address2(dots(end)+1:end);
address = [address1 address2];
if strcmp(network_extension,'uk')
    fprintf('UK address: %s\n',address)
else
    fprintf('non-UK address: %s\n',address)
end
% Change "n.o.body@fiction.co.uk" to "n.o.body@fiction.co.net"
% and run this part of the code again
```

• 7.4.4

```
s = input('And you were saying?... ','s');
% Explain how this next line works ... :)
fprintf('Really, %s?\n',fliplr(strtok(fliplr(s))))
```

• 7.4.6

(a)

```
s = ['Once upon a time, a very long time ago now, about last Friday, ',...
'Winnie-the-Pooh lived in a forest all by himself under the ',...
'name of Sanders. "What does ''under the name'' mean?" asked ',...
'Christopher Robin. "It means he had the name over the door in ',...
'gold letters, and lived under it."'];
```

(b)

```
s = strrep(s,'Winnie-the-Pooh','Dawn French');
s = strrep(s,'Sanders','Stephen King');
s = strrep(s,'Christopher Robin','Bahama Mama');
```

(c)

```
N1 = numel(s);
N2 = sum(s~=' ');
fprintf('The string contains %i characters if counting the spaces\n',N1)
fprintf('and %i characters without the spaces.\n\n',N2)
```

(d)

```
fprintf('The total number of words is %i.\n\n',N1-N2+1)
```

• 7.4.8

```
C = {'Albania';'Andorra';'Austria';'Belarus';'Belgium';...
     'Bosnia and Herzegovina';'Bulgaria';'Croatia';'Cyprus';...
     'Czech Republic';'Denmark';'Estonia';'Finland';...
     'France';'Germany';'Greece';'Hungary';'Iceland';...
     'Ireland';'Italy';'Kosovo';'Latvia';'Liechtenstein';...
     'Lithuania';'Luxembourg';'Malta';'Moldova';'Monaco';...
     'Montenegro';'Netherlands';'Norway';'Poland';...
     'Portugal';'Republic of Macedonia';'Romania';...
     'Russia';'San Marino';'Serbia';'Slovakia';...
     'Slovenia';'Spain';'Sweden';'Switzerland';...
     'Turkey';'Ukraine';'United Kingdom';'Vatican City'};
S = upper(C{randi(numel(C))});
SS = S(randperm(length(S)));
InputString = sprintf(['Anagram of a European country --%s--',...
    '.\nYour guess? --> '],SS);
trial = 1;
while trial < 4
    UserC = input(InputString,'s');
    if strcmpi(UserC,S) % compare the string ignoring the case !!!
        fprintf('\n\nCongratulations! %s is correct!\n\n',S)
        break
    else
        trial = trial + 1;
        if trial == 4 % exit with no success
            disp(['GAME OVER! The country was ' S '.'])
        else
            fprintf('Not correct. Try again!\n')
        end
    end
end
```

- 7.4.10

```
function M = LaTeX_matrix(A)
% outputs a string array with the LaTeX syntax for matrix A
[m,n] = size(A);
s = '\\left[\\begin{array}\n{';
for i = 1:n, s = [s,'r']; end
s = [s,'}'];
for i = 1:m
    s1 = '';
    for j = 1:n
        s1 = [s1,'%d&'];
    end
    s1 = [s1(1:end-1), '\\\\\n'];
    s = [s, s1];
end
s = [s,'\\end{array}\\right]'];
A = A';
M = sprintf(s,A(:)');
```

Chapter 8: Images

- 8.4.2

```
House =[
0   0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    1    1    0    0    0    0    0    0    0
0    0    0    0    0    1    1    1    1    0    0    0    0    0    0
0    0    0    0    1    1    1    1    1    1    0    0    0    0    0
0    0    0    1    1    1    1    1    1    1    1    0    0    0    0
0    0    0    0    3    3    3    3    3    3    0    0    0    0    0
2    2    2    2    3    4    3    3    3    3    2    2    2    2    2
2    2    2    2    3    4    3    4    4    3    2    2    2    2    2
2    2    2    2    3    3    3    4    4    3    2    2    2    2    2
2    2    2    2    3    3    3    4    4    3    2    2    2    2    2
];
house = [
    0 0 1 % blue
    1 0 0 % red
    0 1 0 % green
    0.8 0.8 0.8 % grey
    0 0 0 % black
    ];
f = figure;
```

```
imagesc(House);
colormap(house)
axis equal off
```

- 8.4.2

```
A = imread('BabyC.jpg');
ti = {'Red','Green','Blue'};
figure
for i = 1:3
    subplot(3,1,i)
    p = A(:,:,i); hist(double(p(:)),50);
    title(ti{i})
    grid on
    if i == 1, axis tight, v = axis; else axis(v), end
end
```

- 8.4.6

```
A = imread('bangor.jpg');
B = A*0.3;
figure, imshow(255*0.7+B)
```

- 8.4.8 The function

```
function A = randomise_image_panel(A,s)
A(:,:,strfind('RGB',s)) = uint8(rand(size(A,1),size(A,2))*255);
```

The script

```
A = imread('JazzieSleepTail.jpg');
B = [randomise_image_panel(A,'R'),randomise_image_panel(A,'G'),...
    randomise_image_panel(A,'B'),];
figure('Position',[100,100,800,220]),imshow(B)
```

- 8.4.10

```
A = rgb2gray(imread('Joey.jpg')); % read image and convert to grey
for i = 1:6
    B = imresize(A,.5);
    A(1:size(B,1),1:size(B,2)) = B; % resize and inset
end
figure, imshow(A)
```

- 8.4.12

```
A = imread('Joey.jpg');
s = size(A);
midR = round(s(1)/2); midC = round(s(2)/2);
A(1:midR,1:midC,1) = 255; % red top left
A(1:midR,midC+1:end,3) = 255; % blue top right
```

```
A(midR+1:end,1:midC,2) = 255; % green bottom left
A(midR+1:end,midC+1:end,[1 3]) = 255; % purple bottom right
figure, imshow(A)
```

- 8.4.14

```
A = imread('joey_and_robot.jpg');
figure, imshow(A)
s = size(A);
lw = ceil(s(1)/200);
tenrows = round(linspace(1,s(1)-lw,11));
tencolumns = round(linspace(1,s(2)-lw,11));
B = A;
for i = 1:lw
    B(tenrows+i-1,:,:) = 0;
    B(tenrows+i-1,:,2) = 255;
    B(:,tencolumns+i-1,:) = 0;
    B(:,tencolumns+i-1,2) = 255;
end
figure, imshow(B)
```

- 8.4.16

The function:

```
function Im = shuffle_image(A,M,N)

if ndims(A)==2 %#ok<*ISMAT>
    A = cat(3,A,A,A); % make a grey image into rgb
end

Rows = floor(size(A,1)/M);
Columns = floor(size(A,2)/N);

Im = uint8(zeros(size(A)));

% Shuffle index
RP = reshape(randperm(M*N),M,N);
k = 1;
for i = 1:M
    for j = 1:N
        T = A((i-1)*Rows + 1 : i*Rows,...
            (j-1)*Columns + 1 : j*Columns,:); % take current block
        [new_r,new_c] = find(RP == k);
        Im((new_r-1)*Rows + 1 : new_r*Rows,...
            (new_c-1)*Columns + 1 : new_c*Columns,:) = T; % position
        % in the new row/column
        k = k + 1;
    end
end
```

The script:

```
A = imread('winter_mountain.jpg');
B = shuffle_image(A,4,5);
figure,imshow(A)
figure,imshow(B)
```

- 8.4.18

The function:

```
function [p,q,r] = words_around_shape(C,mode)
N = numel(C);
if mode == 1
    C = sort(C);
end
hold on
axis([-2 2 -2 2])
axis square off
t = 2 * pi / N;
Co = [0; 1];
R = [cos(t) sin(t);-sin(t) cos(t)];
for i = 1:N
    r(i) = text(Co(1),Co(2),C{i},'Rotation',90 - t*(i-1)/pi*180);
    q(i) = plot([0 Co(1)],[0 Co(2)]);
    Co1 = R * Co;
    p(i) = plot([Co(1) Co1(1)],[Co(2) Co1(2)]);
    Co = Co1;
end
```

- 8.4.20

```
A = double(imresize(imread('winter_mountain.jpg'),0.05));
r = A(:,:,1); g = A(:,:,2); b = A(:,:,3);
figure, hold on
set(gca,'FontName','Candara','FontSize',12)
grid on
scatter3(r(:),g(:),b(:),6,[r(:),g(:),b(:)]/255,'filled')
rotate3d
```

Chapter 9: Animation

- 9.4.2 Highlight

```
A = imread('ConnorsCars2.jpg');
r = A(:,:,1); g = A(:,:,2); b = A(:,:,3);
```

```
figure, imshow(A)
waitforbuttonpress
t = get(gca,'CurrentPoint');
% create mask
[x,y] = meshgrid(1:size(r,2),1:size(r,1));
ra = 600; % radius of the region of interest
mask = (x-t(1,1)).^2 + (y-t(1,2)).^2 < ra^2;
r(~mask) = r(~mask)*0.5;
g(~mask) = g(~mask)*0.5;
b(~mask) = b(~mask)*0.5;
B = uint8(cat(3,r,g,b));
figure,imshow(B)
```

• 9.4.4 Squares in a loop

```
k = 10;
co = rand(k,3);
figure, hold on, axis equal off
for j = 1:30
    for i = 1:k
        w = k-i+1;
        fill([w,w,-w,-w],[-w,w,w,-w],co(i,:))
    end
    co(1,:) = [];
    co = [co;rand(1,3)];
    pause(0.1)
end
```

• 9.4.6 Stopwatch

```
S = input('Number of seconds? ---> ');
figure('color','y')
hold on
text(-0.10,0.95,'Stopwatch','FontName','Tempus Sans ITC','Fontsize',20);
t = text(0.2,0.5,'0','FontName','Tempus Sans ITC','Fontsize',140);
axis off
for i = 1:S
    set(t,'String',num2str(i))
    pause(0.98)
end
```

• 9.4.8 Planets (a), (b) and (c)

```
figure, hold on
plot(0,0,'y.','markersize',80) % the sun
theta = linspace(0,2*pi,100);
r1 = 1; % radius of orbit1
r2 = 3; % radius of orbit2
% Orbit trajectories
plot(sin(theta)*r1,cos(theta)*r1,'k--')
```

```
plot(sin(theta)*r2,cos(theta)*r2,'k--')
theta1 = linspace(0,4*pi,250);
theta2 = linspace(0,4*pi,500);
h1 = plot(0,1,'r.','markersize',50); % planet 1 (inner)
h2 = plot(0,3,'b.','markersize',40); % planet 2 (outer)
h3 = plot(sin(theta)*0.8,cos(theta)*0.8+r2,'k--'); % orbit of moon
h4 = plot(sin(theta(30))*0.8,cos(theta(30))*0.8+r2,'k.',...
    'markersize',30); % moon of planet 2
axis([-4 4 -4 4])
axis square off
for i = 1:500
    set(h1,'Xdata',r1*sin(theta1(mod(i,250)+1)),...
        'Ydata',r1*cos(theta1(mod(i,250)+1)))
    set(h2,'Ydata',r2*sin(theta2(mod(i,500)+1)),...
        'Xdata',r2*cos(theta2(mod(i,500)+1)))
    set(h3,'Ydata',r2*sin(theta2(mod(i,500)+1))+cos(theta)*0.8,...
        'Xdata',r2*cos(theta2(mod(i,500)+1))+sin(theta)*0.8)
    % fixed moon
    %     set(h4,'Ydata',r2*sin(theta2(mod(i,500)+1))+cos(theta(30))*0.8,...
    %         'Xdata',r2*cos(theta2(mod(i,500)+1))+sin(theta(30))*0.8)
    % orbiting moon
    set(h4,'Ydata',r2*sin(theta2(mod(i,500)+1))+cos(theta2(i))*0.8,...
        'Xdata',r2*cos(theta2(mod(i,500)+1))+sin(theta2(i))*0.8)
    pause(0.02)
end
```

• 9.4.10 The umbrella

(a) The function

```
function draw_sectors(N,Co)
figure
hold on
axis([-1 1 -1 1])
axis square off
theta = linspace(0,2*pi,N+1);
r = linspace(1,Co(1),N);
g = linspace(1,Co(2),N);
b = linspace(1,Co(3),N);
for i = 1:N
    fill([0 cos(theta(i)) cos(theta(i+1)) 0],...
        [0 sin(theta(i)) sin(theta(i+1)) 0], [r(i),g(i),b(i)])
    pause(0.1)
end
end
```

(b) The extended function

```
function draw_sectors_BW(N,Co,bw)
figure
```

```
hold on
axis([-1 1 -1 1])
axis square off
theta = linspace(0,2*pi,N+1);
r = linspace(bw,Co(1),N);
g = linspace(bw,Co(2),N);
b = linspace(bw,Co(3),N);
for i = 1:N
    fill([0 cos(theta(i)) cos(theta(i+1)) 0],...
        [0 sin(theta(i)) sin(theta(i+1)) 0], [r(i),g(i),b(i)])
    pause(0.1)
end
end
```

(c) The script

```
draw_sectors(20,[0.3 0.2 0.8])
draw_sectors_BW(20,[0.3 0.2 0.8],0)
draw_sectors_BW(20,[0.6 0.9 0.1],0)
while true
    waitforbuttonpress
    if gco == gcf
        delete(gco)
        break
    else
        coo = get(gco,'FaceColor');
        set(gco,'FaceColor',1-coo)
    end
end
```

• 9.4.12 Rotating square

```
figure, hold on
fill([-1 1 1 -1],[-1 -1 1 1],'w'); % white square
fill([0 1 1 0],[0 0 1 1],'r'); % red square
h = fill([0 0.5 0.5 0],[0 0 0.5 0.5],'k'); % black square
fill([0 0.25 0.25 0],[0 0 0.25 0.25],[0.7 0.7 0.7]); % grey square
axis square off
plot([-1 1],[0 0],'k-')
plot([0 0],[-1 1],'k-')
theta = 2*pi/100;
R = [cos(theta) sin(theta); -sin(theta) cos(theta)]; % rotation matrix
for i = 1:100 % rotate the black square
    X1 = get(h,'XData');
    Y1 = get(h,'YData');
    NewPoints = R * [X1(:)';Y1(:)']; % incremental rotation
    set(h,'XData',NewPoints(1,:),'YData',NewPoints(2,:))
    pause (0.05)
end
```

- 9.4.16 Jumping frog

```
figure
hold on
axis([0 1 0 1])
fill([0 1 1 0],[0 0 1 1],'b','edgecolor','b') % pond
axis square off
f = fill([0.47 0.53 0.5],[0.47 0.47 0.53],'g','edgecolor','g'); % frog
oldx = 0.5; oldy = 0.5;
for i = 1:15 % 15 jumps
    x = rand; y = rand; % new position
    X = [x-0.03,x+0.03,x]; % frog coordinates
    Y = [y-0.03,y-0.03,y+0.03];
    set(f,'XData',X,'YData',Y); % update frog position
    plot([oldx,x],[oldy,y],'g--') % plot the trace
    oldx = x; oldy = y; % save the current point as "old"
    pause(0.5)
end
```

- 9.4.16 Scrambled eggs

```
A = imread('eggs.jpg'); % the original image
M = 4; % rows
N = 5; % columns
Rows = floor(size(A,1)/M); % Tile size - rows
Columns = floor(size(A,2)/N); % Tile size - columns
A = A(1:M*Rows,1:N*Columns,:); % reduce the image to match
RP = randperm(M*N); % shuffle index
Tiles = mat2cell(A,ones(1,M)*Rows, ones(1,N)*Columns,3);
Tiles = reshape(Tiles(RP),M,N);
B = cell2mat(Tiles); % shuffled image
RepeatedRP = randperm(M*N);
Tiles(RepeatedRP(2)) = Tiles(RepeatedRP(1));
C = cell2mat(Tiles);
% calculate the dark image
DarkTiles = mat2cell(A*0.3,ones(1,M)*Rows, ones(1,N)*Columns,3);
DarkTiles(RepeatedRP(2)) = Tiles(RepeatedRP(2));
DarkTiles(RepeatedRP(1)) = Tiles(RepeatedRP(2));
D = cell2mat(DarkTiles);
figure
subplot(2,2,1), imshow(A)
set(gca,'FontName','Candara','FontSize',12)
title('Original image')
subplot(2,2,2), imshow(B)
set(gca,'FontName','Candara','FontSize',12)
title('Scrambled eggs')
subplot(2,2,3), imshow(C)
set(gca,'FontName','Candara','FontSize',12)
title('Repeated tile')
pause(3)
```

```
subplot(2,2,4), imshow(D)
set(gca,'FontName','Candara','FontSize',12)
title('Reveal')
```

Chapter 10: GUI

• 10.3.2 Button alphabet

```
figure('Units','Normalized','Position',[0.1 0.4 0.8 0.2])
le = 'A':'Z';
for i = 1:26
    b(i) = uicontrol('Units','Normalized','Position',...
        [(i-1)*(1/26)+0.0003,0.1,(1/26)-0.001,0.8],...
        'BackgroundColor',rand(1,3),'String',le(i),...
        'FontName','candara','FontSize',20);
    set(b,'Callback','set(gco,''BackgroundColor'',''k'')')
end
rp = randperm(26);
set(b(rp(1)),'Callback','delete(b),set(gcf,''color'',''k'')')
set(b(rp(2)),'Callback','delete(b),set(gcf,''color'',''k'')')
```

• 10.3.4 Disappearing green button

```
figure('color','k','Units','Normalized','Position',[0.1 0.1 0.8 0.8])
a = 0.2; b = 0.1;
stepsx = linspace(a,0.001,100);
stepsy = linspace(b,0.001,100);
waitforbuttonpress
p  = get(gcf,'CurrentPoint');
uicontrol('Units','Normalized','Position',[p(1)-a,p(2)-b,2*a,2*b],...
    'BackgroundColor','g',...
    'String','Press to disappear','Fontname','Candara','FontSize',16,...
    'Callback',['set(gco,''String'','''')',for i = 1:100,',...
    'set(gco,''Position'',[p(1)-stepsx(i),p(2)-stepsy(i),',...
    'stepsx(i)*2,stepsy(i)*2]),pause(0.01),end,',...
    'delete(gco),set(gcf,''color'',''g'')'])
```

• 10.3.6 Scrabble helper

```
function scrabble_helper
figure
% letter tile frequencies (100 tiles in total)
ltf = [9 2 2 4 12 2 3 2 9 1 1 4 2 6 8 2 1 6 4 6 4 2 2 1 2 1 2];
let = ['A':'Z',' '];
to_show = ''; % the sequence of 100 tile
for i = 1:27
    to_show = [to_show, repmat(let(i),1,ltf(i))];
end
k = 1; % index in array to_show
```

```
for j = 10:-1:1
    for i = 1:10
        uicontrol('Un','N','Pos',[i-1,j-1,1,1]/10,'Str',to_show(k),...
            'FontName','Candara','FontSize',18,...
            'ForegroundColor',[0.8 0.4 0.4],'Callback',...
            'co = get(gco,''Ba'');set(gco,''Ba'',1 - co)');
        k = k + 1;
    end
end
```

• 10.3.8 Four coloured squares

```
figure, hold on, axis equal off
x = [0 7 7 0]'; y = [0 0 7 7]';
sq = fill([x -x -x x],[y y -y -y],'w');
str = text(-3,-8,'');
q ='3412';f='1432';
for i = 1:4
uicontrol('str',['Square ' f(i)],'Un','N','Pos',[x(i) y(i) 1 1]/8,...
        'Ca',['c=rand(1,3);set(sq,''Facec'',''w''),set(sq(',q(i),'),',...
        ''''Facec'',c),set(str,''St'',num2str(c))']);
end
```

• 10.3.10 Remove-the-triangle game

```
f = figure; hold on, axis([0 1 0 1]), axis square off
triangle_count = 0;
h = zeros(1,20); % array with handles
while triangle_count < 20
    S = 0;
    while S < 0.05
        X = rand(1,3); % x-coordinates of the vertices
        Y = rand(1,3); % y-coordinates of the vertices
        S = abs((X(2)*Y(1)-X(1)*Y(2))+(X(3)*Y(2)-X(2)*Y(3))+ ...
            (X(1)*Y(3)-X(3)*Y(1)))/2; % area
    end
    triangle_count = triangle_count + 1;
    h(triangle_count) = fill(X,Y,rand(1,3));
end

% Create a sound effect
fs = 8000; % the sampling frequency
T = 0.1;% length of the note in seconds
t = 0:1/fs:T;
C = 800; % frequency
y = [sin(C*2*pi*t) zeros(1,20) sin(C*2*pi*t)]; % the signal

j = 20;
while j > 0
    k = waitforbuttonpress;
```

```
    hh = gco;
    if k == 0
        if j == 20
            tic
        end
        if hh == h(j)
            set(hh,'visible','off')
            j = j - 1;
        else
            sound(y,fs)
        end
    end
end
t = toc;
fill([0 1 1 0],[0 0 1 1],[0.8 0.8 0.8],'edgecolor','none')
tt = text(0.25,0.5,sprintf('Your time is %.2f s.\n',t));
set(tt,'FontName','Candara','FontSize',18)
```

• 10.3.12 Colour boxes game

```
figure
k = 1;
for j = 1:4
    for i = 1:5
        h(k) = uicontrol('units','normalized','backgroundcolor','k',...
            'position',[0.1+(i-1)*0.16, 0.18+(4-j)*0.16, 0.16, 0.16]);
        k = k + 1;
    end
end
order_of_squares = randperm(20);
tic
for i = 1:20
    set(h(order_of_squares(i)),'Backgroundcolor',rand(1,3))
    k = 1;
    while (k ~=0)||(gco ~= h(order_of_squares(i)))
        k = waitforbuttonpress;
    end
    delete(h(order_of_squares(i)))
end
st = sprintf('Done in %.2f s',toc);
end_text = uicontrol('style','text','units','normalized',...
    'position',[0.2, 0.5, 0.55, 0.1],'backgroundcolor',get(gcf,'color'),...
    'FontName','Trebuchet MS','FontSize',16,'string',st);
```

• 10.3.14 How fast can you find the numbers?

```
k = 1; % button counter
rp = randperm(100); % number distribution
tic % start the clock
topress = 1; % next number to press
```

```
fs = 8000; cli = sin(2*pi*800*(0:1/fs:0.1)); % click sound
t = (0:1/fs:0.05)*15000; % prepare the frequencies for sound "wrong"
fi = 20; % <--- final number to count to
figure
for i = 1:10 % rows of buttons
    for j = 1:10 % columns of buttons
        uicontrol('Un','N','Po',[(i-1)/10,(j-1)/10,0.1,0.1],...
            'Ba',rand(1,3)*0.4,'Str',num2str(rp(k)),...
            'For','w','FontN','Candara','FontS',16,...
            'Callback',['nu = str2num(get(gco,''Str''));',...
            'if nu == topress, topress = topress + 1;',...
            'set(gco,''Enable'',''off'',''Ba'',''w''),',...
            'sound([cli,cli],fs);else,',...
            'w = [sin((0.5+rand*0.5)*t),sin((0.5+rand*0.5)*t)];',...
            'sound(w,fs),end, if topress == fi+1, clf,',...
            'annotation(''textbox'',''position'',[0.2 0.2 0.6 0.6],',...
            '''Horiz'', ''center'',''Vert'', ''middle'',''String'',',...
            '[''Your time: '' num2str(toc) '' s''],''FontN'',',...
            '''Segoe Print'',''FontS'',14,''EdgeColor'',''none'');end']);
        k = k + 1;
    end
end
```

- 10.3.16 Mirror image

```
[filename, pathname] = uigetfile('*.jpg');
a = imread([pathname filename]);
flippedIm(:,:,1) = fliplr(a(:,:,1));
flippedIm(:,:,2) = fliplr(a(:,:,2));
flippedIm(:,:,3) = fliplr(a(:,:,3));
gr = [0.8 0.8 0.8]; % grey colour for the button background
f = figure;
axes('position',[0.05 0.05 0.9 0.75])
axis off
imshow(a)
b(1) = uicontrol('Parent',f,'Style','push',...
    'units','normalized','position',[0.2 0.9 0.6 0.08], ...
    'string','Original image','BackgroundColor','g',...
    'Callback',['imshow(a),set(b(1),''BackgroundColor'',''g''),',...
    'set(b(2),''BackgroundColor'',gr)']);
b(2) = uicontrol('Parent',f,'Style','push',...
    'units','normalized','position',[0.2 0.82 0.6 0.08], ...
    'string','Flipped image','BackgroundColor',gr,...
    'Callback',['imshow(flippedIm),set(b(1),''BackgroundColor'',gr),',...
    'set(b(2),''BackgroundColor'',''g'')']);
set(b,'FontName','Candara','FontSize',14)
```

- 10.3.18 Moving car

```
figure('Color','w')
```

```
A = imread('YellowCar.jpg');
h = axes('position',[0.02,0.5,0.3,0.1]);
imshow(A)
b(1) = uicontrol('Un','N','Pos',[0.05,0.05,0.25 0.1],'Str','Backward',...
    'Callback',['p = get(h,''Pos'');,',...
    'set(h,''Pos'',[max(p(1)-0.01,0),p(2),p(3),p(4)])']);
b(2) = uicontrol('Un','N','Pos',[0.7,0.05,0.25 0.1],...
    'Str','Forward',...
    'Callback',['p = get(h,''Position'');,',...
    'set(h,''Position'',[min(1-p(3),p(1)+0.01),p(2),p(3),p(4)])']);
b(3) = uicontrol('Un','N','Position',[0.4,0.05,0.2 0.1],...
    'Str','Move','Callback','move_the_car');
set(b,'FontN','Tempus Sans ITC','FontS',18,'Ba',[0 0 0],'For',[1 1 1])
```

This code requires a file with the Callback for the Move button with name `move_the_car.m`

```
p = get(h,'Position');
while p(1)+p(3) < 1
   set(h,'Position',[p(1)+0.005,p(2),p(3),p(4)])
   p = get(h,'Position');
   pause(0.005);
end
CarFlip(:,:,1) = fliplr(A(:,:,1));
CarFlip(:,:,2) = fliplr(A(:,:,2));
CarFlip(:,:,3) = fliplr(A(:,:,3));
imshow(CarFlip)
p = get(h,'Position');
while p(1)-0.01 > 0
   set(h,'Position',[p(1)-0.005,p(2),p(3),p(4)])
   p = get(h,'Position');
   pause(0.005);
end
imshow(A)
```

Chapter 11: Sound

- 11.2.2

```
function piano_keyboard

figure('Units','Normalized','Position',[1 1 4 6]*.1)
f = 440*2.^((-9:3)/12); % note frequencies

q = [1:2:5 6:2:12 13];
for i = 1:8 % white keys
    uicontrol('Un','N','Pos',[i,1,1,8]/10,'Ba','w',...
        'Callback',{@n,f(q(i))});
end
```

```
r = [2 4 0 7:2:11];
for i = [1 2 4 5 6] % black keys
    uicontrol('Un','N','Pos',[.6+i,3.4,.7,5.6]/10,'Ba','k',...
        'Callback',{@n,(f(r(i)))});
end
    function n(~,~,F)
        fs = 6^5;
        T = 1;
        t = 0:1/fs:T;
        sound((T-t)/T.* sin(2*pi*t*F),fs)
    end
end
```

• 11.2.4 Music scales

A piece of code needed for all sub-problems

```
noteFrequency = [261.63 293.66 329.63 349.23 392.00 440.00 493.88 523.25];
fs = 8000; % sampling frequency
```

(a)

```
y = [];
T = 1; % time in seconds
t = 0:1/fs:T;
A = (T - t) / T; % fading amplitude
for i = 1:8
    no = noteFrequency(i);
    s = A .* (sin(2*pi*t*no) + 0.2*(sin(pi*t*no) + sin(4*pi*t*no)));
    y = [y, 0 s];
end
sound(y,fs)
```

(b)

```
y = [];
T = 1;
t = 0:1/fs:T;
A = (T - t) / T;
for i = 1:8
    no = noteFrequency(i);
    noback = noteFrequency(9-i);
    s1 = A .* (sin(2*pi*t*no) + 0.2*(sin(pi*t*no) + sin(4*pi*t*no)));
    s2 = A .* (sin(4*pi*t*noback) + 0.2*(sin(2*pi*t*noback) + ...
        sin(8*pi*t*noback)));
    y = [y, 0 0.8 * s1 + 0.2 * s2];
end
sound(y,fs)
%wavwrite(y,fs,'scales_harmony.wav')
yb = y; % save the signal for the plot in 2(d)
```

(c)

```
y = [];
T = linspace(1,0.1,8);
fs = 8000;
for i = 1:8
    t = 0:1/fs:T(i);
    no = noteFrequency(i);
    noback = noteFrequency(9-i);
    A = (T(i) - t) / T(i); % amplitude is specific for duration T(i)
    no = noteFrequency(i);
    noback = noteFrequency(9-i);
    s1 = A .* (sin(2*pi*t*no) + 0.2*(sin(pi*t*no) + sin(4*pi*t*no)));
    s2 = A .* (sin(4*pi*t*noback) + 0.2*(sin(2*pi*t*noback) + ...
        sin(8*pi*t*noback)));
    y = [y, 0 0.8 * s1 + 0.2 * s2];
end
sound(y,fs)
```

(d)

```
% fs per second, fs/1000 per milisecond, round(30*fs/1000) per 30 ms
N = round(30*fs/1000); % number of samples corresponding to 30 ms
X = linspace(0,30,N); % prepare x-axis
figure, plot(X,yb(1:N),'k-')
set(gca,'FontName','Candara','FontSize',12)
title('The first 30 ms of the harmony scale signal')
xlabel('time [ms]')
ylabel('sound signal')
```

• 11.2.6 What Does Music Look Like?

The function:

```
function see_music(y)
N = numel(y);
T = 2000;
hos = floor(N/T); % split into T pieces
ints = floor(N/hos);
y = y(1:ints*hos); % truncate to a multiple of hos
r = reshape(y,hos,[]); % arrange consecutive intervals of length hos
pf = mean(abs(diff(r>0))); % find a proxy for the frequencies
Q = linspace(0,2*pi,T);
figure, hold on
for i = 1:min(ints,T)
    plot([0 pf(i)*sin(Q(i))],[0 pf(i)*cos(Q(i))],'k-','color',[0 i/ints 0])
end
axis equal off
```

Call the function with:

```
load handel
see_music(y)
```

Index