Introduction to MATLAB Programming for Behavioral Scientists

Jan Hausfeld

21-10-2020

Contents

1 Overview and learning objectives

1.1 Summary

To answer a scientific question, we often use experiments. During such a scientific experiment, data is gathered, this data is then analysed and finally presented. Most courses focus on how to analyse and present data, or how to do the actual measurements. In contrast, this course will focus on how to acquire data. Acquiring data manually can be very cumbersome or even impossible (in case you want to measure EEG, eye-movements or reaction times for example). Programs can do this job for you. However, when you design new experiments, sometimes no suitable programs are available, or if you make changes in the design of your experiment, you might need to adopt the program to those changes. A commonly used software environment in science is MATLAB (MathWorks).

The main objectives for this course are to quickly learn the basics skills you need in MATLAB, practice the application of these skills by repeating the classical visual search task from Treisman & Gelade [1] and finally apply the knowledge by programming a Stroop task with your own twist [8] in the second half of the course.

1.2 Set up

At the end of this course, you/your group will have implemented a Stroop task to answer a research question. In order to do so, you will be introduced to all basic concepts in MATLAB (first two weeks) and will apply these concepts when implementing the classical visual search task [1] (weeks 3 to 4). In the last week, the knowledge of all 7 weeks will be tested in an exam.
Thus, alltogether this course consists of 8 weeks. **The theoretical background of the course will be presented on Canvas in an online fashion accompanied by the book** ([2]). It is your task to work through the content before the **obligatory online workgroups on Wednedays and Fridays**. There will be a **live Q&A session on Tuesdays**, in which the planning and other practical information of the course is clarified, specific feedback on handed in assignments is given, and there is time for questions.

In the tutorials on Wednesdays of weeks 2 and 3 at 1:00 p.m., you have to **individually** do a small test about week 1 and 2 respectively that will count for 10% (5% each, no resit possible) of your final grade (see section 1.4 for details). After week 4 you have to hand in your code and documentation for the visual search task (10% of your final grade). At the end of the course, you have to hand in your code and an expanded documentation, Stroop task experiment (20%). The course will end with an exam (45%). You must earn at least a 5.5 on both the last assignment and the exam to pass the course. Further details for all assignments and the exam will be made available during the course.

Finally, to enhance learning and create some extra feedback for you, **there are peer-feedback assignments where you have to give feedback on the code of another student between the two work-group**s. For this, you should upload the respective assignments by Thursday 3 p.m.

such that the other person can prepare.

Jan Hausfeld (j.hausfeld@uva.nl)) coordinates the course and gives most of the lectures. If there are any questions, problems or remarks feel free to ask or email! For the work-groups, three additional experienced programmers will be present to help you with all your questions.

1.3 Learning objectives

a) The student is able to describe and apply basic functions and code in Matlab such as logical operators, loops, calculations, creating random numbers, plotting figures, creating own functions and scripts
b) The student is able to write basic code, while applying programming practices of good style and documentation (commenting, working in sections)
c) The student is able to reproduce the classical visual search task, and is able to critically reflect on their own process
d) The student is able to understand, detect bugs, and improve another person's code
e) The student is able to program a short experiment, acquire, store and analyse data

The learning objective (a), which is mostly *Knowledge* based, can be broken into smaller parts, which also provides a good guideline throughout the earlier part the course:
(a1) The student is able use the MATLAB interface and can make use of the different windows
(a2) The student is able differentiate between variable types (double, string, array (vector and matrix), array or structs) and apply this knowledge
(a3) The student is able to make the distinction between the following MATLAB operators and can apply them: +, -, *, /, .*, ./, =, >, <, ==, &, ||, strcmp
(a4) The student is able to implement the following control flow statements: if, for, while
(a5) The student is able apply MATLAB functions to create figures: figure, plot and use figure handles
(a6) The student can make use of the following functions to create random numbers and uses them at the appropriate moment: rand, randn, randi, randperm
(a7) The student is able to distinguish between a script and a function, and makes use of the appropriate one
(a8) The student can make use of the save function to save data in a .mat file
(a9) The student can independently find, understand and implement built-in functions
(a10) The student can use vectors of structs to store and access data of different types
(a11) The student can use conditional indexing

1.4 Grading

Your final grade will be composed of the grade for:

- the weekly tests (weeks 1 and 2) - **10%** (no resit possible)
- the documentation and code of the visual search paradigm, see section 3.2.10 (week 3 and 4) - **10%**
- final assignment, (see section 3.3), - **20%**
- the attitude and skills you show during the practicals (for grading model see canvas) - **15%**
- and the exam - **45%.**

Note that both the grade for your own experiment AND the grade for your exam must be at least 5.5 to pass the course. If your final assignment is graded with less than a 5.5 you can rewrite the code and/or documentation + article for a maximum grade 6.

The grading of your code is based on what is done at Harvard's cs50 course https://cs50.harvard.edu/ :

> Your work [...] will be evaluated along four axes primarily.
> **Scope**: To what extent does your code implement the features required by our specification?
> **Correctness**: To what extent is your code consistent with our specifications and free of bugs?
> **Design**: To what extent is your code written well (i.e., clearly, efficiently, elegantly, and/or logically)?
> **Style**: To what extent is your code readable (i.e., commented and indented with variables aptly named)?
> *quote from cs50.harvard.edu*

Keep these four axes in mind while coding! You will find the models we use to grade the documentation, the article, your attitude and skills, the elevator pitch and the code on canvas.

2 Scientific Background (week 3 & 4)

During weeks 3 and 4 you will practice the new MATLAB knowledge by implementing a visual search task. Below you find background information about this task.
"Everyone knows what attention is. It is the taking possession by the mind in clear and vivid form, of one out of what seem several simultaneously possible objects or trains of thought...It implies withdrawal from some things in order to deal effectively with others." (William James, Principles of Psychology, 1890) [3]

There are different types of attention that can be investigated in multiple different ways. A classical experiment to investigate visual attention is the visual search task by Treisman & Gelade [1] where a subject has to find a target in a busy environment with other (similar) objects (distractors). There are two distinct conditions: 1) The target has only one property that differs from the distractors (e.g. colour or shape) and 2) the target differs in more than one property from the distractors (e.g. colour and shape). In the former the time it takes to identify

the object (e.g. a black circle in blue crosses and circle, figure 1 left) is independent of the number of distractors due to the so called pop-out effect. This situation is also called feature search or disjunctive search. In the latter condition (e.g. a black round within black crosses and blue rounds and crosses, figure 1 middle) the reaction time linearly increases with the number of distractors. This is called a conjunctive search. These results form the basis of the feature integration theory and the research on the binding problem (for a more modern review see [4] and for more easy to read information see [5] - the main information of both sources are exam material).

During this course we will make use of this classical experiment to apply the MATLAB knowledge gained during the first two weeks. During the second half of the course you will have programm the Stroop task with your own little research question and the scientific background that belongs to it.

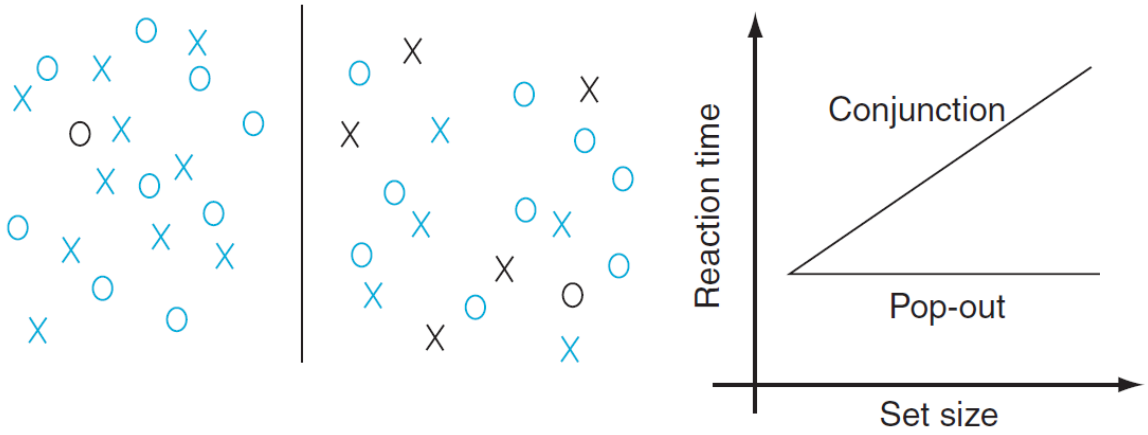

Figure 1: The visual search task by Treisman en Gelade: Example of a disjunctive search or colour pop-out (left), a conjunctive search (middle) and the differences in reaction time depending on the number of distractors (set size) in both situations (right). Figure from [6].

3 Exercises

To become able to programm the Stroop task, you first have to learn the basics (first two weeks) and apply them to an example experiment (weeks 3 and 4).

NB: You need to prepare for the work-groups at home (below you find for each week a detailed description of what you have to prepare, see subsections 'Preparation'). In general you have to prepare the chapters introduced during the Tuesday lecture. **But be careful**: During the lecture only selected concepts will be addressed. However, the complete chapters will be part of the exercises, test, assignments and the exam. The time during the work-groups is not enough to finish all assignments. Thus plan time during the week to finish your assignments outside contact hours!

In the following sections you will find the exercises for the first two weeks (see section 3.1), a guide on how to implement the visual search experiment for weeks 3 and 4 (see section 3.2) and the assignment for the final three weeks (see section 3.3).

Everyone learns programming at a different speed. There might even be students that have programmed before. For those of you that are a quick study or already have some experience there are extra assignments: The hackers' edition! When you are done with your weekly assignments and have time left or you want to learn that extra bit go to subsection 3.2.4 and choose an extra topic of your choice. The topics are chosen to allow for a better version of the visual search task and the Stroop task.

Peer-feedback
To make sure you receive enough feedback and to give you practice reading other peoples' code, you will engage in a peer-review period every week (during week 1 to 4). Every Thursday by 3 p.m. you have to hand in a few exercises or parts of the visual search task code. You have to give feedback on another student's exercise and will receive feedback from another student at the beginning of Friday work-groups.

3.1 Week 1 and 2: The basics

3.1.1 Preparation

For the first two weeks you need to prepare chapters from the book [2]. You find a detailed description of what you have to prepare at the beginning of each week (see below). In general, we advice to not only read the examples in the book but to do them yourself. This makes learning how to program much easier (and it is more fun if something actually happens!). If you have questions about the content of the book or if you have problems following the examples please do not hesitate to ask during the lecture or the work-groups! NB: There are multiple editions of Attaway. All chapters and exercises mentioned here refer to the 5th edition [2]. If the chapters do not match any more, just look up the same topics in your edition. A detailed list of topics can be found at the beginning of each week.

Before the first lecture: The UvA offers MATLAB licenses for all students. Please make sure you install a recent version of Matlab **before** the first lecture (see instructions below). Note that installing Matlab is an activity that can take more than an hour depending on your internet connection and Mathworks servers' performance, so plan ahead. Finally, Matlab requires several GB of space on your hard drive, so make sure you have enough space before you start the installation.

Installing Matlab: Go to https://datanose.nl/#byod (you most likely have to log in) and follow the instructions under the heading 'Matlab'. For more information check https://student.uva.nl/en/content/az/software/matlab/matlab.html

3.1.2 Graded Test

On the Wednesdays of weeks 2 and 3 at 1:00 p.m. there will be a small test about the content of weeks 1 and 2 respectively. The aim of these tests is to evaluate your progress to help you to keep track of what you should know by then and to familiarize you with one of the question types we use during the exam. The feedback you will receive will help you to better understand the criteria we use to grade code. The test will be one exercise per week. You may use your book for those tests, but not for the final exam. The tests of both weeks are obligatory and each count for 5% of your grade. Thus, the first two weeks make up 10% of your final grade. NOTE: There is no resit for a better grade possible.

3.1.3 Week 1

First, we will get to know the MATLAB interface and the help-function. We will use MATLAB as a calculator and get to know different variable types. Understanding the differences between different variable types is essential because functions can often only receive input with a certain data type or store their output in a certain type. To become able to do specific analyses on a subset of the data we will learn about indexing specific elements within a variable.

As a researcher, you often want to repeat an experiment several times or do the same analysis more than once. Therefore it is useful to be able to *save commands* to repeat them later. In MATLAB you can use *scripts and functions* to store commands and reuse them later. Researcher often use and adopt other peoples' scripts for their own research or want to understand how you did an analysis. To make it possible for them (and for yourself) to understand already written code you will learn the basic rules of *commenting and formatting code*. *From now on you are supposed to use them at all times!* Additionally, to be able to visually inspect your acquired data you will learn how to present it in figures.

Sometimes, you only want to analyse the data of your female or right-handed participants. Or you only want to give feedback if the participant's answer was correct. Additionally, you sometimes want to selectively change variables based on some property. In programming languages pieces of code that allow for these cases are called *selection statements* or *conditional statements* and they are the final content you will learn about during week 1.

Preparation

During the first week we will practice the content of chapters 1, 2, 3 and 4 from the book[2]. Make sure you read the chapters well and run all examples in MATLAB yourself. Below you find a more detailed lists of all topics you are expected to know before the work-groups. If you have questions, first try to find the answer on the internet (stack-overflow is a great resource answering almost any question and you-tube films offer a great resource to have the concepts explained). If you cannot find the answer or something is still unclear, do not hesitate to ask during the lecture and the work-groups.

- Vectors and Matrices, and how to use indices
- The MATLAB interface: command window, command history, workspace

- MATLAB's help-functions
- MATLAB as a calculator: addition & subtraction, multiplication & division
- The variable types double, logical and string
- Random numbers and the functions to make them in MATLAB
- Use of sections %%: not in the book, look up 'Run Code Sections' in 'Search documentation'
- Use of the editor, and how to create and save programs
- The difference between a script and a function, how you create, use and run them
- Why and how you put comments in your code (important sections from the book: 3.2.1 and 3.7.1)
- How to plot data in a figure
- How to use 'Selection statements' (Focus on if statements! You don't need to know switch for this course)
- Meaning of $>$, $<$, &&, || and $\sim$ and how to use them
- Difference between = and == and the use of logicals

Peer-feedback

Create a separate script with the exercises and hand in **chapter 1, exercises 11 and 15, and chapter 2, exercises 27 and 30 before Thursday 3 p.m. via canvas**. Make sure you peer review the exercises assigned to you on Thursday 3 p.m. before Friday's work group. Since this is the first week exercises are quite short and giving feedback will not take much time. Still, try to give at least one tip and one compliment (top) on each exercise.

Exercises

Put all exercises in one script, as longs as they are not functions. Start each exercise as shown below for the first exercise. Make sure that each exercise is written in its own section.

```
%% chapter 1, exercise 2
myage = 38;
 ...
```

NB: To create a section there **has** to be a space after the %%. If you want to run an individual section (thus not the complete script) push ctrl+enter. The section where your cursor is placed will be run.

During the work-group, you have to do the following exercises from the 5th edition of the book [2]. Plan your time well so that you can finish all exercises during the lessons, but also don't forget to take breaks when necessary!

Chapter 1: 1, 3, 9, 10, 11, 15, 16^{+}, 17, 20, 21, 22, 25, 31 & 37. $^{+}$Note exercise 16: Start the exercise with exiting MATLAB
Chapter 2: 5, 6, 7, 12, 15, 19, 24, 27, 30 & 40.

Up to now, you have put all the exercises in one script separated by sections. However, we are

now going to work with functions. Functions (kind of a script but then with input and/or output) need their own file. You can create a function by clicking on 'New' and then 'Function' in MATLAB's 'Home' tab. Give each function its proper name and save it immediately. **NB: The name of the function and the name you give the file must be the same!** To keep one script with all your exercises (much easier to study for an exam) always call your function in the script: `myfunction(8)` or, if you want to allocate the output of the function to a variable `myoutput = myfunction(8);` .If you want to use a vector or matrix as an input you can first create it in a script `myvector = 1:3;` and then use it as input for your function `myfunction(myvector)`.

Chapter 3: 2, 15, 17, 18, 26,, 27, 28, 34, 35*, 38 & 31 using your knowledge from chapter 4 (thus first practice chapter 4 and finish this exercise last).

For exercise 38, the figure should look similar to the one below (depending on the user input). Thus, you have to create a vector from 0 to the limit of t the user put in in the script.

Chapter 4: 1**, 2, 10, 16, 20 & 21.

Graded Test

A small test will take place at 1 p.m. on Wednesday of week 2 to test your knowledge gained during week 1. **It is obligatory and worth 5%**. You may use your book and this manual but no internet or other notes.

*In exercise 35 from chapter 3 you implement a function that will turn out to be useful for the visual search task. However, for the visual search task the function has to return a variable with the type char if the input was a cell. Adopt the function so that it complies with the requirements. Since we will need it again at a later point of the course check with a member of the staff that you came up with a good solution.

In the example in the book if `myInput == 'x'` is used to compare the actual input to the desired input. Comparing with == works letter by letter, meaning that `'aaa' == 'abc'` results in the vector [1 0 0] and not a simple 0. To make sure the complete string is compared, always use the function ***strcmp* to compare characters and strings, except if you explicitly want to do it letter by letter. Using == will be considered bad practice in the former cases.

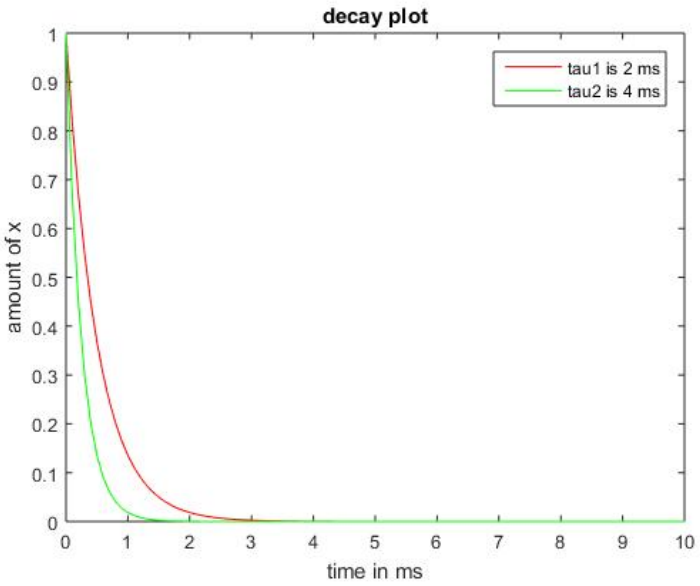

Figure 2 : Example of the figure that the script from exercise 38 can create.

3.1.4 Week 2

Next to being able to selectively run an action (or not), you sometimes want to run an action several times. To achieve this you can make use of *loops*. **This week we will practice using both, *conditional statements* and *loops*, together called *controlflow statements*.**

Then, we will mostly focus on functions and concepts we actually need to implement the visual search task. First, we will write a script to measure reaction times, then we will get to know how to order and save the data we will create in the task using a *vector of structs*.

Preparation

During the second week we will practice the content of chapter 5 and apply the knowledge gained in the first four chapters [2]. Make sure you read the chapters well and run all examples in MATLAB yourself. Additionally read in Attaway [2] about structures (mainly section 8.2.4, p. 289-297 but also the sections before that to understand the context). Below you find a more detailed lists of all topics you are expected to know before the work-groups. If you have questions first try to find the answer on the internet (stackoverflow is a great resource answering almost any question and youtube films offer a great resource to have the concepts explained). If you cannot find the answer or something is still unclear do not hesitate to ask during the lecture and the work-groups.

- Loop statements: both for and while, we save vectorizing for the hackers edition
- The built in functions strcmp, pause and fprintf (check the MATLAB documentation)
- How to correctly layout and comment 'controlflow statements' (see 'Programming Style Guidelines' in the book [2], p. 147 and p. 192)

- Understand the use of input checks (see *data validation* and p. 176 to 179 in the book [2])
- Use of error messages
- Use of tic-toc
- Get to know figure handles,
  understand how to use them to make figures (in-)visible,
  what hold on is, and
  how you can use different plotting symbols

Peer-feedback
Hand in chapter 5, exercise 16 and exercise Dice (Exercise 1—doesn't need to be final) before Thursday 3 p.m. via Canvas. Make sure you peer review the exercises assigned to you on Thursday 3 p.m. before Friday's work group. Give at least one tip and one compliment (top) for each exercise.

Exercises
During the practical you have to make the following exercises from the 5th edition of the book [2]. Plan your time well so that you can finish all exercises during the lessons but also don't forget to take breaks when necessary!

Chapter 5: 5, 6, 7, 13, 14, 16, 22 & 36.

Now, you know basically all concepts needed to program in MATLAB. From now on, to improve you have to get to know more built-in functions and practice, practice, practice... Therefore the next exercise (Dice) combines (almost) all knowledge you acquired up to now. After that, a couple of new built-in functions are introduced and the use of loops is practised (Dice 2.0).

Exercise 1: Dice
Write a script that simulates a game of rolling dice. Make use of...
- randi() ,
- *controlflow statements*,
- fprintf() .
- Don't forget to use % and %% for comments and sections where necessary and
- use informative variable names.

You are free in how you implement the simulation as long as you fulfil the specifications below:
- The player and the computer both roll two dice each
- If the two dice aren't the same than the sum of the dice is the score. If they are the same, then the score is one die times ten. Thus:

  Rolling a 4 and a 6 makes 10
  a 1 and a 1 makes 10
  a 6 and a 5 makes 11
  and a 6 and a 6 makes 60

- After the dice are rolled show in the command line who rolled what, what the

individual scores were and who won.

Example:

You: 3 and 6, score: 9
Computer: 1 and 1, score: 10
The computer wins!

To create this kind of output use the fprintf function.

Hackers edition: Make use of pause and *flowstatements* to add *flavour* to the game. You can also let the player choose to play or to quit using questdlg and/or you can make the game *best-in-3*.

Exercise 2: Dice 2.0

This game expands from the exercise before. We will rewrite the game to become a function that takes as an input the number of players you want to play with. The function has to create a vector with the automatically calculated scores of each player. To make the game more fun, make sure that the scores are plotted for each player one by one.

We've broken the assignment down into several steps. Test the program after each step and make sure it still works before you continue with the next step.

1. Create a function called 'dice.m' that receives the number of players as an input and a vector with the final scores of all players as an output.
2. This game works for 2 to 4 players. Make sure the function controls that the input is valid. In other words make sure the input is a integer, is smaller than 5 but bigger than zero and let an error message pop up if this isn't the case.

This step is called *input checks* of a function. Make sure that whenever you create a function that receives input, that the input satisfies the requirements. If it doesn't, make sure an informative error message pops up.

3. Plot the scores of each player one by one (Tip: hold on). Plot the scores on the y-axis against the player-numbers on the x-axis (Tip: Use xlim([0, n+2]) to make sure your x-axis doesn't keep changing). Tip: If your symbol is very small use the marker option to change it.
4. Now make sure that the function waits until a button press occurs before the next player is plotted (pause).
5. Hide the score of the former player whenever a new player is plotted. You have to use handles for this. Tip: set(nameHandle, 'Visible', 'off') .
6. At the end make all scores of all players visible again in the figure.

Exercise 3: Tic-Toc

During a behavioural experiment, it is often useful to be able to measure a participant's reaction time to a stimulus. This is what we are going to practice during this exercise. If everything went well, you know that the time between two things can be measured using tic - toc in MATLAB. If you don't know what it does look it up! If you combine tic - toc with pause (for pause see exercise 2) you can show a figure with a stimulus and measure the reaction time of the participant.

Create a script **reactiontime.m**. Make sure the script waits after it is initialized until you press a key before the experiment starts. Plot a figure with a random number between 0 and 100 on the y-axis. Use an 'X' as a symbol to plot the number (see figure 3 for how this could look like). As soon as the symbol appears, start measuring the time. If the plotted number is greater than 50 the participant has to press 'a', else 'b'. Store how long it took the participant to press the button and which button was pushed. To store the last button press use the following code: buttonPress = get(gcf, 'CurrentCharacter')*.*

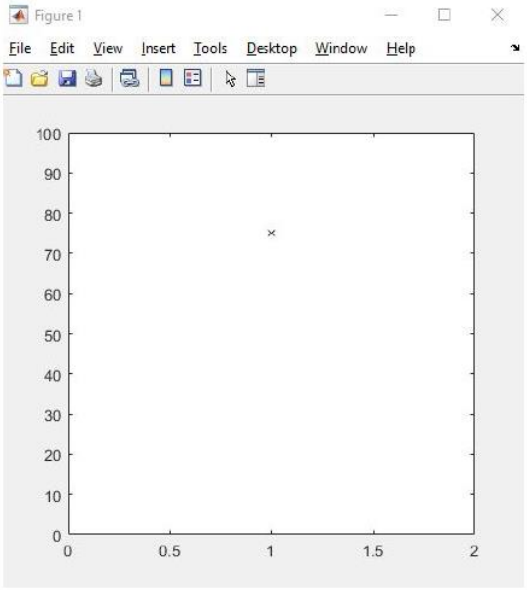

Figure 3 : Example of how the tic-toc exercise could look like

The script that you have just created seems trivial and possibly even unnecessary, however you will see that a comparable piece of code is part of the visual search task that you will implement during the coming two weeks. Therefore save the script properly and don't forget to comment it clearly.

Exercise 4: Vector of structs

This exercise is about saving data using a 'vector of structs'. Since this is quite a complex topic you will find some explanations and an example before the exercises. Don't forget to use Attaway [2] to understand the concept better.

The biggest advantage of structures is that they can hold different data types (for example strings and doubles). There are other data types that can do this as well. If you have time left check the cell arrays in the hackers edition (section 3.2.4). You have prepared structures at

home. Try to understand the following example of a vector of structs. If you don't understand what is happening read up in the book, check the internet and/or ask a member of the staff.

Example: Vector of Structs
Assume we have data about our students' names, IQs and grades. This data would look something like table 1.

Table 1: Overview student data, exam grades and IQ

Student	Student number	IQ	Grade
Henry	1	86	5.5
Harry	2	120	8.1
Sally	3	113	3.3

To save data using column names you can use structs in MATLAB. To save the data from table 1 you can use the following code (as with the book, do the example yourself in MATLAB):

```
% Simply insert data, fields are generated automatically.
% data(1).Student => this stands for row 1, column 'Student'

data(1).Student = 'Henry';
data(1).StudentNumber = 1;
data(1).IQ = 86;
data(1).Grade = 5.5;

data(2).Student = 'Harry';
data(2).StudentNumber = 2;
data(2).IQ = 120;
data(2).Grade = 8.1;

data(3).Student = 'Sally';
data(3).StudentNumber = 3;
data(3).IQ = 113;
data(3).Grade = 3.3;
```

Double click in your workspace on your struct 'data'. A new variable window will appear with a table that should look similar to table 1. A vector of structs therefore is a useful way to save table data in MATLAB!
But how can we now make use of the data in the table? To get a specific datapoint, index your struct with the rownumber and add the variable of choice after the dot:
IQHarry = data(2).IQ gives IQHarry = 120.
If you want to access the whole dataset of a student (in other words access a complete row) index the struct with the row number:
DatasetHarry = data(2);
Finally, if you want to access one variable with all the numbers in there (thus access a column)

use Cijfers = [data(:).Cijfer].
NB: The [ ] are necessary to access a complete row, if you forget adding them you will access only the first value. For names use { }, if you use [ ] you will receive one word with all names fused together. Try this, you will need it later!

In the dataset above, we used a table with data from different students. You can also use a vector of structs to save data from one person doing multiple trials. In the example below the data is saved per trial in its respective row, the row number being nTrial, a counter from 1 to the number of trials. Keep in mind that even though the code is written down independently here, it is normally used in a loop (for or while). Whenever the loop is executed the values of the variables are determined and the nTrial counter is increased by one.

```
% savce data for each trial in the different columns
data(nTrial).ppn = ppn;
data(nTrial).condition = condition;
data(nTrial).nObject = nObject;
data(nTrial).target = hadTarget;
data(nTrial).rt = reactietijd;
data(nTrial).correct = correct;
data(nTrial).trialNumber = nTrial;
```

If you save the data of four trials in this way you will end up with a table like table 2.

Table 2: Example data of four trials from a visual search task

participant number (ppn)	condition	nObject	target	rt	correct	trialNumber
	dsymbol	12	1	11.506	0	1
	dcolor	56	0	18.755	1	2
	dcolor	4	1	17.751	1	3
	dsymbol	24	0	10.757	1	4

Check whether you understand the code and try to make this vector of structs in MATLAB yourself. If this works, create your own vector of structs using a for-loop. Make sure the vector of structs contains 5 columns and 20 rows with data. Which data you enter is your own choice but make sure it's different in the different rows. One way to fill it is using random integers like in table 3.

By now you are able to create your own vector of structs. To practice this, make the exercises below. These exercises are from the 5th edition of the book [2].

Chapter 8: exercise 18 (just use three elements as an example, not the whole table) & 19.

Table 3: Random data table

Column1	Column 2	Column 3	Column 4	Column 5
	91	49	88	80
	90	26	87	34
	68	88	16	14
…	…	…	…	…

Exercise 5: Tic-toc 2.0
Go back to your tic-toc script. Change the code so that the participant has to do the experiment ten times and not only once. Store in a vector of structs the number of the participant[1] (make sure the participant can enter the number or hardcode it into the script), the button pushed, the random number, the reaction time and the correctness. At the end of the script save this data using the function save. Use the number of the participant as the name of the file.

Graded Test
A small test will take place at 3 p.m. on Wednesday of week 3 to test your knowledge gained during week 1 and 2. It is obligatory and worth 5%. You may use your book and this manual but no internet or other notes.

3.1.5 Hackers edition (until 3.2)

If you already have experience with programming, if you are a quick study or if you just want to learn that extra bit you can have a look at the topics below. We have gathered topics and exercises that are useful to make the visual search task you will implement during week 3 and 4 and that will add to your understanding of MATLAB and programming in general. There is no specific order in the topics, just pick the topics that sound interesting to you. **NB: These topics are not obligatory! However, to get the most out of the course, don't hesitate and have a look.**

Cells
While doing an experiment you usually gather lots of data. To be able to easily analyse the data later on, it is useful to save it per trial. You have learned how to use vectors of structs for that. MATLAB also has another variable type for that: cells. Structs and cells are quite similar to each other with the biggest difference being how the data can be read out. A structure of a cell resembles an excel sheet: It has rows and columns and you can index them using cellStructure{x-index, y-index}. Cells use {}. You can also use the () but then you will receive the data as a new cell and not a double or string (depending on what was in the cell you indexed).

```
>> cellStructure(2)
ans =
[2] ⇐ Cell
>> cellStructure{2}
ans =
 2 ⇐ double
```

[1] NB: To protect privacy always use anonymous data, thus never use the name of the participant

If you want to use a cell create it beforehand. It is much quicker to fill a cell in a loop that already exists since MATLAB already assigned a big enough chunk of memory to the cell and does not have to reassign memory every time you add something. It is considered bad design if you don't. The same holds true for a vector and structs. You can create a cell using:

```
%% By the {} notation, meaning empty cell. just like [] is a empty vector.
cellStructure = {};

%% By a cell notation filled with integers (; for columns)
cellStructure = {1,2,3,4,5};

%% By a cell notation with strings
cellStructure = {'welcome','to','matlab'};

%% By the cell constructor for n by n (5 by 5)
cellStructure = cell(5)

%% By the cell constructor for x-by-y (by-z etc) (2 by 2)
cellStructure = cell(2,2);
```

You can fill an empty cell by assigning a value to a certain cell:

```
%% This is how you assign an 'x' to the 4th column, 1st row (all other cells remain empty):
cellStructure{1,4} = 'x';
```

Now make exercise 4 from week 2 but remodel the table 1 into a cell. Make use of column number instead of the names you usually use in vectors of structs: c{1,1}, c{1,2} etc... .

```
%% This is how you assign an 'x' to the 4th column, 1st row (all other cells remain empty):
cellStructure{1,4} = 'x';
```

Then recreate Tic-toc 2.0 using a cell structure instead of a vector of structs.

Vectorisation

Vectorisation is a MATLAB property that allows for parallelisation of certain processes. This saves a lot of time and is one of the big advantages of MATLAB. First check out vectorisation in the book [2] (p.171 ff). After reading the book follow the example below on how to vectorise a for-loop:

```
for n = 1:10000
    V(n) = 1/12*pi*(D(n)^2)*H(n));
end
```

The for-loop above can also be written in the vectorised form below:

```
% Vectorized Calculation
V = 1/12*pi*(D.^2).*H;
```

In the vectorised example above MATLAB will square all elements of D because of the .^: D is a vector (or matrix) and MATLAB will calculate the correct value from each element in D and put it in V. V will become a vector (or matrix) as well. The same holds true for H: For each element in D the corresponding element with the same index will be used in H (just like in a loop). This means of course that D and H have to have the same dimensions. The dot signifies that the following operations (thus squaring and multiplying) have to be performed for each element separately. In MATLAB this is much faster than using a loop. So whenever possible use vectorization, this is good design!

Practice vectorisation using the following exercises from the book [2]. Chapter 5, exercises 26, 27, 28, 29 & 30.

Below you find a number of functions that can be written in a vectorised way. Rewrite the code below into vectorised functions and show that the new code runs quicker than the original (use tic-toc to do so). When you are ready check with an assistant if everything works out.

```
%% exercise 1
x = 1:pi/20:2*pi;
y = zeros(1,length(x));
for i=1:length(x)
    y(i) = sin(x(i) + 0.5*pi) + 5;
end
plot(x,y);

%% exercise 2
a = rand(1,800);
b = 0;
c = 0;
for i=a
    if i < 0.5
        b = b + 1;
    else
        c = c + 1;
    end
end
if b > c
    disp('More than 50% was greater than 0.5');
end
```

Example pseudo-random controlled randomness

Reading a paper, you often see that groups are pseudorandomized. This means that random trials with a predefined proportion of conditions are presented, so that you can't predict which

condition will be next but all conditions will end up with equally many trials. To create random trials you can use rand in MATLAB. The disadvantage of this function is that the distribution will only be uniform (equal distribution over all values) when you run it more than 1000 times. There are multiple different ways to ensure equal group sizes for trial numbers below 1000.

Do it yourself

First try to develop a pseudorandom generator yourself. This generator should print the numbers from 1 to 10 in a random order without repeating any of the numbers. Try to implement it in MATLAB.

A simple controlled randomness generator

A simple way of solving the problem above is to create a vector that you fill with all possible values (e.g. the numbers from 1 to 10). Then, you draw one number after the other from the vector without putting them back until the vector is empty. At this point during an experiment you can decide you are done (enough trials) or you can refill the vector and start again. The latter option comes in handy if you know the proportions of the conditions you want (e.g. 1:1:1) but don't know the number of trials beforehand (because you want to only count correct trials). In the example below, we show a pseudorandom script that uses refilling of the list. The script randomly chooses one of 5 conditions. Each trial has to do each condition at least once before the next trial begins. The next trial again has to be 5 conditions in a random order etc. Run through the script and check whether you understand the use of every line.

```
% a vector with the conditions
conditions = ['a', 'b', 'c', 'd', 'x'];
nTrials = 20;
% produces a random seed based on the time when you run the function
rng('shuffle');
% RNG stands for random number generator, it sets the number where functions like randi
% and rand create their numbers from
% if you use 'shuffle' a new seed is generated every time the functions is run
% if you always use the seed 1234 (a seed is a number to start with for the formula
% that produces the random numbers), you will always receive the same random numbers.

% create a copy of the conditions
reload = conditions;

% Run nTrials
for t=1:nTrials
    fprintf('Trial %i\n',t);
    % Run for all possible conditions
    for i=1:length(condities)

        % random integer between 1 and the number of conditions (length(conditions))
        random = randi(length(conditions));
        % pick the random condition
        condition = conditions(random);
```

```
        % This line removes the condition used from the vector.
        conditions(random) = [];

        % Check if conditions is empty, if it is refill
        if length(conditions) < 1
            conditions = reload;   %reload: our copy of conditions
        end
        %% Run your experiment with the chosen condition
        fprintf('experiment condition %s is run\n',condition);
    end
end
```

This example for pseudorandomisation is only one of the possible implementation. Keep in mind that other ways to control your randomisation might be equally valid to use or might work even better later on.
Have you gained new inspiration from this code? Then adopt your own pseudorandom generator script!

Graphical User Interfaces
Graphical User Interfaces, usually called GUIs, allow the user to enter data in a user friendly way without having to access functions or scripts. You can make use of very simple GUIs in MATLAB (the function inputdlg for example shows an input box, try it yourself! See figure 4) but you can also create more complex GUIs. Below you see the GUI of a simulation of the action potential of the squid giant axon by Touretzky [7] (figure 5).

Figure 4: Simple GUI. A simple way of letting the participant input his student ID (studentnummer) is the use of the function inputdlg.

Such a more complicated interface can be relatively easily programmed in MATLAB using the Graphical User Interface Development Environment (GUIDE). Read as an introduction chapter 13.3 (p. 475 ff) from the book [2]. You can then practice with exercises 13 to 24 (same chapter). The exercises become more and more complex and in the end you can almost always enhance your GUI. So don't do all the exercises and keep in mind which parts of a GUI can be useful for the visual search task and your own experiment.

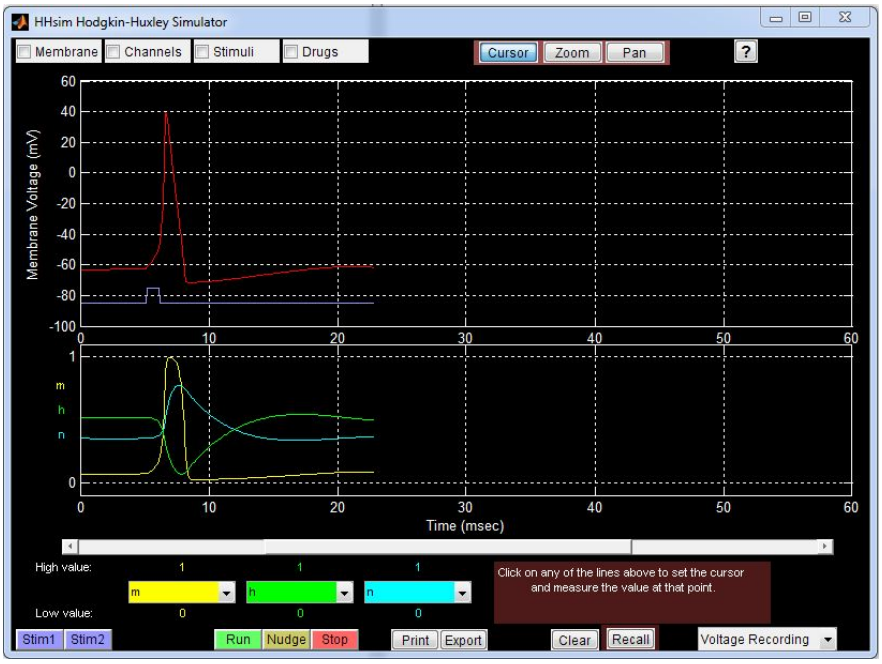

Figure 5: **GUI of the Hodgkin & Huxley simulation program.** Screenshot of the simulation script HHsim [7] that can be used to study the currents and potentials during an axon potential in a simulated squid giant axon. Such an extensive GUI can relatively easily be made using GUIDE.

3.2 Week 3 and 4: The visual search task

3.2.1 Preparation

During weeks 3 and 4, we will implement a visual search task. Make sure that you know what a visual search task is before the work-group. The original article is by Treisman and Gelade [1], a more recent review of the topic is by Wolfe and Horowitz [4] that also summarized information about visual search in a scholarpedia article [5]. You can use your own resources as well but make sure you can explain the rational of a visual search task before the work-group. Lastly, before you get started with the visual search task carefully read the information about the documentation (section 3.2.10) so that you keep notes during the process (half of the work will be done before you start with the assignment if you keep good notes!).

3.2.2 Peer-feedback

Hand in the Treisman function before week 3, Thursday 3 p.m. via canvas. If it's not complete yet, you can ask the reviewer for specific help by adding a comment. Make sure you peer review the code assigned to you on Thursday 3 p.m. before Friday's work group. You don't have to comment on every line of the code but make sure you give a couple of compliments and tips.

Hand in the script that runs the visual search task before week 4, Thursday 3 p.m. via canvas. If it's not complete yet you can ask the reviewer for specific help by adding a comment. Make sure you peer review the code assigned to you on Thursday 3 p.m. before Friday's work group. You don't have to comment on every line of the code but make sure you give a couple of

compliments and tips.

3.2.3 Instructions week 3 and 4

At this point you know all the basic principles of MATLAB and it is time to practice them by creating your own experiment. You will program the visual search task in couples, so find yourself a partner. Obviously, it is a lot more complex to implement an experiment than doing one of the exercises. If you do and report an experiment in science, you usually follow the empirical cycle (figure 6). However, implementing an experiment in MATLAB is not science but engineering. Engineers follow a similar cycle as scientists: The 'Development cycle' (figure 7). The first step in the 'Development cycle' is to inventory the requirements for the product. Then the requirements are translated to a concrete design. The design is then implemented and the results are evaluated in the testing phase. In the last phase, the feedback phase, the recommendations based on the results are given. Then, the next cycle with adjusted requirements begins. During weeks 3 and 4 we will guide you through one development cycle and during the final assignment (weeks 5 to 7) you have to do it yourself. Both times you have to hand in a written documentation as (part of) the assignment.

Since this is probably the first time you work as a developer, you can follow a step-by-step protocol (see below) on how to define the requirements and create a design. During the lecture you have learned about flowcharts as design tools. We will use them here to create a design, too. During the last three weeks, you (and your partner) have to make your own design for your Stroop task with a twist (and implement it etc.).

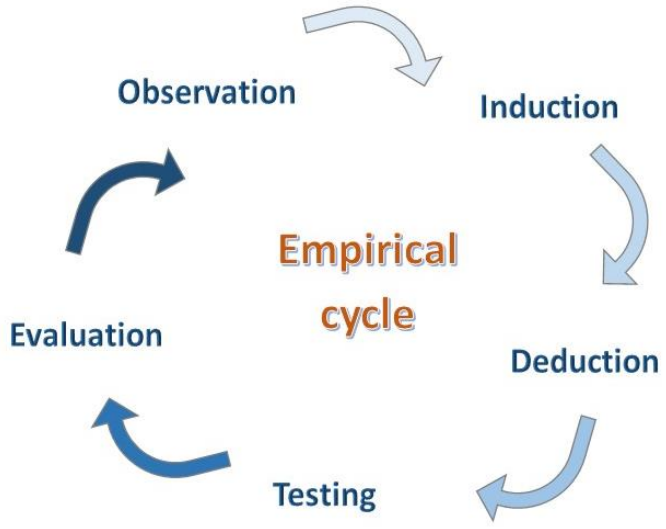

Figure 6: The empirical cycle. The empirical cycle describes the different steps a scientist does during an experiment. It starts with an observation of a phenomenon and resulting inquiry. In the induction phase an hypothesis is formed that is translated into experiments during the deduction phase. The experiments to test the hypothesis are performed in the testing phase. The results are evaluated in the last phase. The evaluation of the results can result in new observations and another round in the cycle.

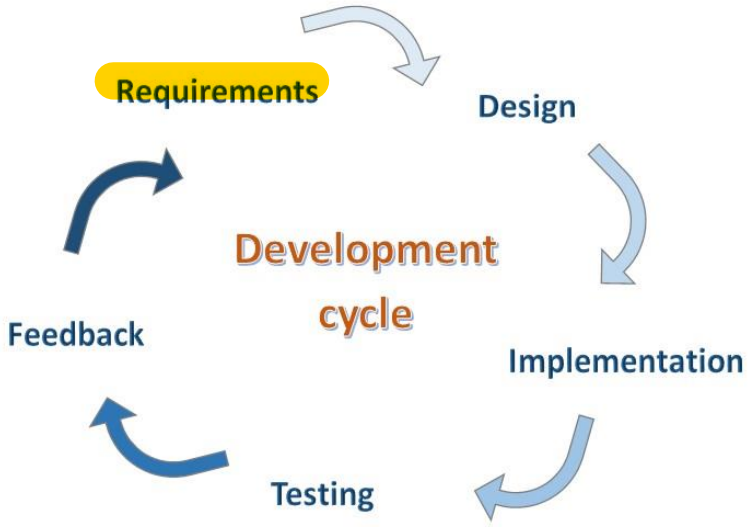

Figure 7: The development cycle. In the development cycle the different steps to develop a product (e.g. a software) are described. First, the requirements have to be defined. After translating them to a design, the software will be implemented. During the testing phase data is acquired to be able to evaluate the software in the last phase. The evaluation can give rise to adjusted requirements and the beginning of a new cycle.

Step 1: Requirements

The aim of the first step is to generate an overview on what needs to be done. In your requirement list you have to write down all the things the program has to be able to do. To write a requirement list, you first have to get to know everything about the program you want to write. Therefore, you already read the background literature [5], [4] and [1] at home. If you have not read it yet, do it now. Then write down all the requirements that your program has to fulfil in order to get data comparable to figure 1. Break it down into small steps that are possible to program for example 'the program measures the reaction time'. Discuss your requirements with a member of staff before you proceed.

Step 2: Design

In the design step, the more abstract requirements are translated into a concrete design. It is a two-step process. First, you have to decide on how you want to design your experiment in general. How many functions and scripts are you going to use and how will you nest them into each other. Use a new function each time you have a complex process that has to be repeated multiple times. Don't forget you can also nest a function into a function. The outer shell of a MATLAB program usually is a script that controls how often a certain condition is performed. Discuss the global design of your program with a member of staff.

The second design step is the design of each individual function and script. You will implement every function one-by-one starting with the innermost function. We will use flowcharts of each function as a design tool. Those flowcharts are very specific ('There are two types of symbols, X and Y'). Before you start implementing make sure a member of staff approves your design.

Often during the programming process the way you program differs from the original plan. Whenever you deviate from the original flowchart create an updated version. You will have to hand in the original flowchart and the flowchart that represents your actual code for the assignment of this week (see 3.2.10) and comment on the changes you have done.

To sum up: First make your requirements list. Then make a global design of how your scripts and functions will be nested, start with the innermost function, make a flowchart for it and then implement it. Then do the same with the next function/script etc.

At the end of week 4 you have to hand in the code of your complete experiment, two datasets you acquired with your program and a documentation of your program (see section 3.2.10). You can distribute your time at your own liking. As a rule of thumb you should have both functions implemented after week 3 and write the script, acquire data and write a brief evaluation during week 4. There is no extra preparation for week 4 but, since time is tight, we advice you to prepare your flowchart for the script at home.

3.2.4 Hackers edition

For those of you that already know a bit more about programming, are a quick study or just want to do that extra bit again there are some extra things you can implement at the end of all functions and scripts. We are curious to see what you can manage to implement!

3.2.5 Function 1: Create stimuli

This is the innermost function. Since this is the first function you will implement, we made a couple of exercises that will help you to make a design and implement the function. So, if you cannot figure out how to make a design follow the exercises on this page to get some ideas. Before you start with the following page you have to have your flowchart ready! The function will create the stimuli. Stimuli are defined as the individual figures the participant gets to see. As discussed in section 2, there are two different types of stimuli: *Conjunctive* and *disjunctive* (figure 1). In both cases a stimulus is made up from n objects, whereof one is the *target* (that differs from all the others), if a target is present. In short, a given stimulus has the following properties:

1. *n* (double: total number of objects in the figure)
2. *cond* (string: 'dcol' for a disjunctive search with colour pop-out, 'dsym' for a disjunctive search with symbol pop-out or 'c' for conjunctive search)
3. *target* (double: 0 if absent or 1 if present)

We define that 'dcol' corresponds to a disjunctive search with colour pop-out, but obviously other names would be possible as well. However, to make trouble-shooting easier please stick to the above mentioned variable names.

The function that creates the stimuli is called 'Treisman_exp'. It will receive the above named properties as its input. To help you, we have already created the function 'put_symbol_inFigure' that places a symbol in a figure (see canvas for the m-file). Run through the function, can you explain what it does?

```
function    put_symbol_inFigure(loc, s, k)

    % This function places a letter (symbol) s, with colour k
    % at location loc in a figure
    % loc = array of 1 x 2, with numbers between 0 and 1 (relative in figure),
    % first is x-coordinate, second is y-coordinate
    % s = string, e.g. 'X' or 'O'
    % k = string that gives the colour, e.g. 'g' or 'r'

    g=text(loc(1), loc(2), s);
    set(g, 'color', k);
end
```

Use put_symbol_inFigure.m to make the following exercises that will guide you towards an implementation of the Treisman_exp function:

1. Put one cross at a random spot in a figure. Use put_symbol_inFigure.m to do so.
2. Put one red cross, 11 red circles and 12 blue circles into a new figure. Which type of search is that? Why do we use 12 blue circles and not 11?
3. Put one red cross, 11 blue crosses and 12 blue circles into a new figure. Which type of search is this?
4. There is one search type missing. Make a new example of this search type. Explain why you chose the numbers you did.
5. Make MATLAB choose a random target (both properties, symbol and colour). Choose the symbols and colours of the distractors based on the random target and the condition (type of search). Do this for all three conditions and put the results in three different figures.
6. Create the function Treisman_exp.m

Before you get started with programming the real function make sure a member of staff approved your design!

You can (but don't have to) use the following beginning for your function 'Treisman_exp', but don't forget that you need approval of your flowchart before you begin.

```
function Treisman_exp(n, cond, target)

    % This function creates a figure with n objects, with or without a
    % target for a disjunctive or conjunctive visual search
    % INPUT:
```

```
    % n = number of objects (target + distractors)
    % cond = 'c' of 'dcol' of 'dsym'
    % (cond search: conjunctive, disjunctive with colour or with symbol)
    % target = 0 or 1 (target is absent or present)

    % input checks

    %% values

    %% choose colour and symbol of target (random)

    %% make colour and symbol of distractors

    %% make figures and add symbols
    %% first add the target (if target is present) at a random location
    % then add the distractors (how many of which type --> balance) at
    % random locations
end
```

Test the colours, and symbols on your own screen. Are they well visible, and easy to distinguish? If they are not adapt them. If you or your partner have colour blindness change the symbols so that both of you can distinguish them easily.

Before you get started with programming make sure you have thought about how many distractors of which type are visible in type of search (hint: make use of all possible different types of distractors in each search, including the conjunctive) and how many distractors of each type you have to show to have a good balance (including the target as many Xs as Os and as many red as green objects. NB: The balance will never perfectly work out for the conjunctive search). When you have implemented the search, check all the balances in all types of searches with the target absent and present. You will need this function later on so if you are not sure whether it works correctly, check with an assistant.

Hackersedition: Treisman_exp.m

You have just made the function Treisman_exp(n, cond, target). As an extension you can also add the requirements below:

1. As discussed before you need to check whether the inputs of a function are valid, and generate an informative error message if they don't. Next to the checks already in the function make sure that the 'target' can only be a boolean (logical) and 0 or 1. Everything else should generate an informative error.

2. To open a figure or close a figure outside the function you need global figure handles. This is useful when using a GUI. Store the handle of your stimulus figure in a variable.
   (a) Dont use 'close all'. This will also close GUIs.
   (b) For now (this exercise will proceed in the next function) make sure the figure will be

closed at the end of the function.

(c) Additionally make sure that if the figure is not properly closed at the beginning of the next function call it will

i. be used again...

ii. ...or be removed.

This is a preparation for the hackers edition of the next function.

3. Make sure controlled randomness is used when necessary (see section).

4. Extra requirements for the stimulus figure:

(a) Make sure X and O do not overlap with each other.

(b) Make sure X and O are not plotted on the axis when they are visible.

3.2.6 Function 2: Measure the reaction time and store it in a variable

Now you can create the stimuli! Nice! The next step is to create one trial: Show a stimulus, let the participant give a response (is the target present or absent?), measure the reaction time and check whether the response was correct. To achieve this you will create the function 'do_experiment'. It will receive the same input as 'Treisman_exp' but gives the reaction time and the correctness of the response as its output. Again, you may or may not use the beginning below and make sure you first get your flowchart approved!

```
function [reactionTime, correct] = do_experiment(n, cond, target)
    % description of the function

    %% Create figure
    % Make use of the function you wrote before.

    %% Measure reaction time. Make use of tic-toc and pause. Remember that
    % 'aw = get(gcf, 'CurrentCharacter');' gives you the last button pressed

    %% Check whether response is correct or not

end
```

Tip: Make use of tic-toc and pause when implementing this function. If you forgot how they work, check the help function and look back at the exercises you did before. Keep in mind that you can let a figure vanish and appear, this can come in handy as well.
When you are done implementing this function, check it. Do the different n, cond and target combinations turn out as expected? Do the reaction times look reasonable and does the function determine correctly whether you hit the correct or the wrong button? Is the output correct? Again, you need this function later so if you are unsure, check with an assistant.

Hackersedition: do_experiment.m

1. Make use of the handle you made in the previous hackers edition (see section) to keep the figure invisible until tic is run so that measuring the reaction time becomes more accurate.

2. Make sure only the keyboard is used for input and no mouse clicks.

3. Make a simple GUI using GUIDE (see section) that allows to
(a) fill in the student number by the participant
(b) fill in the needed variables (number of trials, percentage targets etc.)
(c) push a button to start the experiment...

3.2.7 The script: Control the experiment

Up to now, you have implemented a function that shows a visual search stimulus and then measures reaction time and correctness of the response. The last missing step is to set up the whole experiment. This is done in a script rather than a function. Can you explain why? Before you start programming prepare a flowchart. Keep the following requirements and questions in mind:

- To investigate whether the conjunctive search is linearly dependent on the number of objects we need to test different numbers of objects (different *n*s). Which requirements do those *n*s have to meet to work with the function you have written before? How big must n become to see an effect? Choose four *n*s you want to investigate.

- For the data to be meaningful you need at least 20 correct trials with the target present per search. We call this a hit-trial. How are you going to make sure that for every condition and every set size there will be at least 20 correct trials with the target present?

- Keep in mind that you want to randomly assign a target or not. For this course we ask you to keep the chance of the target to be present at 80%. Why do we need trials without a target and why isn't it useful to use a 50-50 chance? Don't forget to save whether the target was absent of present for each trial.

- Remember to save the reaction times and whether each trial was correct or not in a variable. You could use a vector of structs for that. Save your data in a *.mat* file so that you can analyse it later. Make sure that you don't overwrite your files with the data of the next participant (for example by saving it with the same name, so use different ones).

- There will be quite some trials a participant must undergo, therefore it is useful that you make blocks of trials after which the participants can take a break.

Before you start, in between blocks and at the end you can show a figure with instructions. You can use and adopt the following code for that:

```
figure;
text(0.2, 0.7, 'Press a button to start the block');
text(0.05, 0.5, 'Press "t" if you see a target and "n" if the target is absent');
pause
```

As for the functions it is again useful to get the script working in steps and keep adding extra features while testing every step. Below you find a step by step plan that you can follow:

1. First get the script working for *n = 24* and *cond = 'dcol'*
2. Add the different setsizes *n = [..., 24, ...]*
3. Does the script work for the four different setsizes in the *dcol* condition? Then first add the *dsym* and then the conjunctive search

When you are done with your script first test it on yourself (you might want to decrease the number of hit-trials). When you are sure it works, proceed, else ask a member of staff for help.

Hackersedition: script

1. Count how many correct trials a participant has done during a block. If there are less than 20 correct trials, add extra trials to the block.
2. Make the *start-experiment-button* that you made in section 3.2.7 to run your script.
3. At the beginning of the experiment check whether...
    (a) ... the data you gathered before might be overwritten
        Also give a solution of the problem (e.g. adding a suffix to the file).
    (b) ... there is still a window open from another experiment. Close it!
    (c) ... the data from the experiment before is still in the workspace, clear it!

    (d) ... the GUI works correctly.

4. Make sure the user gets to know what is going wrong whenever an error appears.
    (a) You can use msgbox or dlgbox for that but there are more boxes.

5. Instead of saving the data at the end of the experiment also save it after each trial. A useful function to do so is matfile. Matfile does not load the m-file into working memory but directly manipulates it on the hard-drive Thus makes it resistant against sudden errors, power breakdowns etc.
6. Ensure that the number of different kinds of trials remains uniform (use controlled randomisation)
7. At the end of the experiment clean the workspace after you have saved all data necessary. Thank the participant for his participation.

3.2.8 Doing the experiment

You have made a design and implemented the software. Now it is time for the next step: put it to the test. Time to get some results. Part of your assignment will be to hand in two datasets per couple (as a .mat file) that you acquired with your own software. Do not forget the participants' privacy when naming the data files.

Instruct the participant that it is important that his/her answer is both quick and correct. It will

be difficult to interpret your data when there is a speed/accuracy trade-off next to the reaction time increase in the conjunctive search. To prevent a bias make sure every participant receives the same instructions. Therefore it is useful to write the instructions down.

Attention!!! You have to end all programs other than MATLAB when doing your experiment otherwise the computer might be too slow to accurately measure the reaction times.

3.2.9 Analyse experiment

To really replicate the results of Treisman and Gelade [1] you need more than two datasets. In view of time you will only plot the two datasets you acquired and visually inspect and describe the results (see 3.2.10 for more details on what we expect you to do) and use them for your documentation.

First, calculate the average reaction time for each condition (each search and each n) and per participant. Plot for each participant the means and connect them with lines, use different colours for the different searches. Then plot the individual trials as individual circles. Do not forget to name the axes and add a legend. You should create two figures similar to figure 8. Thus plotting the averages is not enough - we also want you to plot all individual datapoints (hit trials). Don't forget to save all your analysis steps in a script or function. You have to hand this is with the assignment.

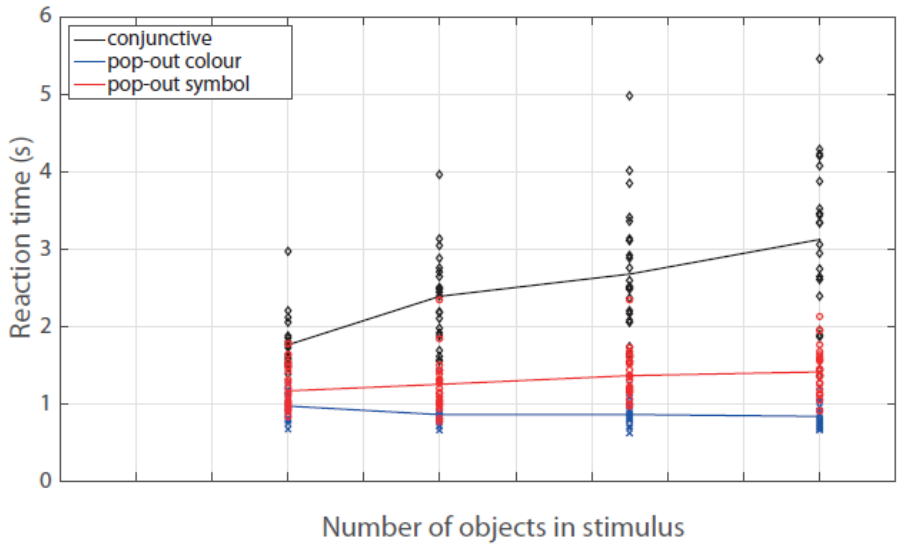

Figure 8: Example of the results of one participant of the visual search task. For both pop-out conditions the reaction time barely changes with the number of objects. For the conjunctive search the reaction time seems to increase with the number of objects. X-axis labels are missing to not influence the students' choice of n in the earlier exercises (but should be present in your figure of course).

3.2.10 Graded assignment

At the end of the two weeks you have to hand in the following products in couples:

- Your MATLAB code including the code for analysing and plotting your data and all non built-in functions you used
- The two datasets you acquired
- The complete documentation of your program (see below)

Documentation

When a client buys a product from an engineer, she usually also expects a documentation of the agreed requirements of the code, the thoughts behind the design, a description of what the code does, results of the testing phase, a summary with know limitations and bugs, and a list of features that still should or could be implemented to improve the product. Maybe even more relevant for you, if you or your colleagues come back to code you wrote years ago, it is absolutely necessary that there is a step-by-step of documentation on what the program should do, how this was implemented and how the program turned out including known limitations, bugs and a to-do list with features your were planning to implement - we tend to say we don't need to write it down we will remember, but that is seldom true for code. With the assignment of this part of the course we therefore practice the documentation of your work.

The documentation follows the development cycle and therefore should consist of five parts (see canvas for a template document). Since we also followed the developmental cycle you will see that some of the documentation parts have already been done during the process. For each part a brief description is given below of what we expect you to deliver.

- Requirements

The requirements of a product originate from the goal of the product and the original problem. So start this section with a brief overview of the research question (problem) and the task that the code should be able to do (goal) (max 300 words). Then give an extensive list of requirements at the level of what can be implemented in MATLAB (see 3.2.3). You should have made this list at the beginning of week 3.

- Design
  - Global (max 300 words)

An explanation of how the program is separated into scripts and functions and why this design was chosen (give reasons from a programming perspective - "because it was in the syllabus and the staff said so" is not a reason). This can be illustrated graphically and should include a description of the role of each individual script/function.

  - Detailed

*The flowcharts that describe your function/script step by step (one per function/script). You should make a flowchart before you start programming the function/script.**

- Implementation

The code including the analysis script/function and all non built-in functions. This will be graded separately.

- Testing (max 300 words)

Figures of both datasets (in article format) and a brief evaluation whether the program does what it should do and how the data compares to the original data. If there are differences

between the datasets and/or with the original data those should be discussed and possible explanations should be given.

- Feedback

An extensive list of features/changes in the code that would improve the program (on design & implementation level!). Note all bugs and limitations in the code.

Additionally we ask you to do a reflection exercise to help you improving your coding skills for the last part of the course (max 400 words): *What was difficult to implement? What were possible solutions to this implementation problem? Which one did you chose and why? Did you deviate from your original design? If you did, discuss why.* But don' forget: The aim of this assignment was not that everything was flawless! You are learning how to program, it is normal to make errors. This is no problem at all. What is important is to learn to identify errors and learn from them. Therefore, try to describe the process you went through while programming: When something went wrong, how did you notice? How can you prevent it from happening again?

Note: The documentation can also be seen as a form of scientific communication. So the known rules for referring to sources and citing apply.

* Often during the programming process the way you program deviates from the original plan. Whenever you deviate from the original flowchart create an updated version. Hand in both flowcharts (original & actually implemented). Discuss deviations in the reflection question.

3.3 Week 5 to 8: Stroop Task plus own twist

3.3.1 Instructions week 5 to 8

During the first four weeks of this course you have learned and practised all the basics. Now, it is time to apply them in practice. During the second half of the course you will have to recreate the Stroop Task [8] and add you own little twist to it. These could be, for example, 3 short questions such as the Cognitive Reflection Test, or testing different pieces of music. The process will be quite similar to the process for developing the visual search task: we will follow the design cycle. The assignment will be/has been extensively introduced during the lecture, here you find a step-by-step overview of how to create an experiment and a general planning (see table 4). If you are unsure, do not hesitate to ask! For this part of the course you will have a member of staff assigned to you, for all questions specifically about your assignment, talk to your assigned member of staff. All other questions will be answered by all members of staff.

Note that if you follow the step-by-step planning, you will gather most of the information necessary for your documentation on the fly.

1. Read up on the Stroop task, and think about what short and easy add-on is interesting as a **research question**. Base it on literature. Make sure it is feasible in the time frame. Before you proceed check it with your tutor. Note, that you do not need to come up with a

mind-bending idea. You should choose a short twist which explores the Stroop task's essence.

2. Think about how you want to implement the Stroop task and the twist.
3. Write an extensive list of **requirements**.
4. Make a **design** (don't forget that the design step is comprised of two steps, first decide on what the different functions and scripts will be, then make a flowchart for each function/script) - you will need both parts for the documentation.
5. Implement the functions and scripts one by one and keep testing them to check whether they work and fulfil the requirements. **The deadline for your experimental code is the Monday of week 7, before the end of the day.**
6. Acquire data. The work-group of the Wednesday of week 7 is reserved to acquire data and take part in your colleagues experiments.
7. Analyse your data akin to section 3.2.9 and test your question statistically if n-numbers allow.
8. Complete your documentation (see section 3.3.4).

Make sure that a member of staff approves of each step before you proceed. Keep all the literature, requirements and designs, you will need them later for the documentation and the article.

Table 4: Overview planning own experiment

	Tuesday Lecture	Wednesday Work-group 2 hrs	Friday Workgroup 4 hrs
Wk 5	Intro Stroop task, Presenting stimuli, Try-catch	Literature Study, Develop add-on idea	(Approval idea), start implementation
Wk 6	(Analysis)	Implementation, testing	Imlementation, testing, generate output
Wk 7	(potential lecture) Preparation exam, deadline code	Gather data	*Analysis, documentation*
Wk 8	Exam		Deadline final assignment

Note: The Tutorial on Wednesday in week 7 is the last one, there is only one tutorial, i.e. no more Friday tutorials

3.3.2 Peer-feedback

Hand in the part of your code you want to receive feedback on, before week Thurday 3 p.m. in week 5 and 6 via canvas. If it's not complete yet you can ask the reviewer for specific help by adding a comment. Make sure you peer review the code assigned to you on Thursday 3 p.m. before Friday's work group. You don't have to comment on every line of the code make sure you give a couple of compliments and tips.

3.3.3 Graded assignment

The code is due at the beginning of week 7 to ensure that you can gather data in the Wednesday tutorial. The documentation is due at the end of week 8, but can be handed in earlier.

The documentation is akin to the documentation from week 4 (see 3.2.10), thus use the instructions from week 4. For the testing part, give two representative examples of a dataset acquired from two people during the Wednesday session.
Expanded documentation: In addition to the typical documentation, you need to write an introduction about the task and the motivation for your twist (max 350 words excluding citations) as well as a scientific discussion interpreting your results in the context of the literature (max 350 words excluding citations).

4 Account for exercises & manual

The exercises are partly based on chapter 3 of Wallisch [6] and chapters 1-5 and 8 of Attaway [2]. This study manual is adopted from the study manual of the module 'Inleiding Programmeren' of the course 'Experimentatie jaar 2' from the bachelor program 'Psychobiologie' at the University of Amsterdam (UvA).

References

[1] Anne Treisman and Garry Gelade. A Feature-Integration Theory of Attention. *Cognitive Psychology*, 12:97–136, 1980.
[2] Stormy Attaway. *Matlab: A Practical Introduction to Programming and Problem Solving*. Elsevier Science, 5th edition, 2018.
[3] William James. The principles of. *Psychology*, 2, 1890.
[4] Jeremy M. Wolfe and Todd S. Horowitz. What attributes guide the deployment of visual attention and how do they do it? *Nature Reviews Neuroscience*, 5(6):495–501, jun 2004.
[5] Jeremy Wolfe and Todd S Horowitz. Visual search. *Scholarpedia*, 3(7):3325, 2008.
[6] P Wallisch, M Lusignan, M Benayoun, T I Baker, A S Dickey, and N G Hatsopoulos. *MATLAB for Neuroscientists An Introduction to Scientific Computing in MATLAB*. Academic Press, Amsterdam, 2009.
[7] David S Touretzky, Mark V Albert, Nathaniel D Daw, Alok Ladsariya, and Mahtiyar Bonakdarpour. Hhsim: Graphical hodgkin-huxley simulator.
[8] Stroop, John Ridley (1935). *"Studies of interference in serial verbal reactions"*. Journal of Experimental Psychology. **18** (6): 643–662. *doi*:*10.1037/h0054651*.