CHAPTER 1

Introduction and Aims

1.1 WHAT IS PSYCHOPHYSICS?

According to the online encyclopedia Wikipedia, psychophysics is “… a sub-discipline of psychology dealing with the relationship between physical stimuli and their subjective correlates, or percepts.” The term psychophysics was first coined by Gustav Theodor Fechner (see front cover). In his Elements of Psychophysics (1860/1966) Fechner set out the principles of psychophysics, describing the various procedures that experimentalists use to map out the relationship between matter and mind. Although psychophysics is a methodology, it is also a research area in its own right, and a great deal of time is devoted to developing new psychophysical techniques and new methods for analyzing psychophysical data.

Psychophysics can be applied to any sensory system, whether vision, hearing, touch, taste or smell. This book primarily draws on research into the visual system to illustrate psychophysical principles, but for the most part the principles are applicable to all sensory domains.

<table>
<thead>
<tr>
<th>1.1 What is Psychophysics?</th>
<th>1.4.2 Functions and Demonstration Programs in Palamedes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2 Aims of the Book</td>
<td>1.4.3 Error Messages in Palamedes</td>
</tr>
<tr>
<td>1.3 Organization of the Book</td>
<td>References</td>
</tr>
<tr>
<td>1.4 Introducing Palamedes</td>
<td></td>
</tr>
<tr>
<td>1.4.1 Organization of Palamedes</td>
<td></td>
</tr>
</tbody>
</table>
1. INTRODUCTION AND AIMS

1.2 AIMS OF THE BOOK

Broadly speaking, the book has three aims. The first is to provide newcomers to psychophysics with an overview of different psychophysical procedures in order to help them select the appropriate designs and analytical methods for their experiments. The second aim is to provide software for analyzing psychophysical data, and this is intended for both newcomers and experienced researchers alike. The third aim is to explain the theory behind the analyses, again mainly for newcomers, but also for experienced researchers who may be unfamiliar with a particular method or analysis. Thus, although *Psychophysics: A Practical Introduction* is primarily directed at newcomers to psychophysics, it is intended to provide sufficient new material, either in the form of software or theory, to engage the seasoned practitioner. To this end we have made every effort to make accessible, to both expert and non-expert alike, the theory behind a wide range of psychophysical procedures, analytical principles and mathematical computations: for example, Bayesian curve fitting; the calculation of d-primes ($d'$); maximum likelihood difference scaling; goodness-of-fit measurement; bootstrap analysis and likelihood-ratio testing, to name but a few. In short, the book is intended to be both practical and pedagogical.

The emphasis on practical implementation will hopefully offer the reader something not available in textbooks such as Gescheider’s (1997) excellent *Psychophysics: The Fundamentals*. If there is a downside, however, it is that we do not delve as deeply into the relationship between psychophysical measurement and sensory function as *The Fundamentals* does, except when necessary to explain a particular psychophysical procedure. In this regard *A Practical Introduction* is not intended as a replacement for other textbooks on psychophysics, but as a complement to them, and readers are encouraged to read other relevant texts alongside our own.

Our approach of combining the practical and the pedagogical into a single book may not be to everyone’s taste. Doubtless some would prefer to have the description of the software routines separate from the theory behind them. However we believe that by integrating the software with the theory, newcomers will be able to get a quick handle on the nuts-and-bolts of psychophysics methodology, the better to go on to grasp the underlying theory if and when they choose.

1.3 ORGANIZATION OF THE BOOK

The book can be roughly divided into two parts. Chapters 2 and 3 provide an overall framework and detailed breakdown of the variety of psychophysical procedures available to the researcher. Chapters 4–8 can be considered as technical chapters. They describe the software routines and background theory for five specialist...
topics: Psychometric Functions; Adaptive Methods; Signal Detection Measures; Scaling Methods; and Model Comparisons.

In Chapter 2 we provide an overview of some of the major varieties of psychophysical procedures, and offer a framework for classifying psychophysics experiments. The approach taken here is an unusual one. Psychophysical procedures are discussed in the context of a critical examination of the various dichotomies commonly used to differentiate psychophysics experiments: Class A versus Class B; objective versus subjective; Type 1 versus Type 2; performance versus appearance; forced-choice versus non-forced-choice; criterion-dependent versus criterion-free; detection versus discrimination; threshold versus suprathreshold. We consider whether any of these dichotomies could usefully form the basis of a fully-fledged classification scheme for psychophysics experiments, and conclude that one, the performance versus appearance distinction, is the best candidate.

Chapter 3 takes as its starting point the classification scheme outlined in Chapter 2, and expands on it by incorporating a further level of categorization based on the number of stimuli presented per trial. The expanded scheme serves as the framework for detailing a much wider range of psychophysical procedures than described in Chapter 2.

Four of the technical chapters, Chapters 4, 6, 7 and 8, are divided into two sections. Section A introduces the basic concepts of the topic and takes the reader through the Palamedes routines (see below) that perform the relevant data analyses. Section B provides more detail as well as the theory behind the analyses. The idea behind the Section A versus Section B distinction is that readers can learn about the basic concepts and their implementation without necessarily having to grasp the underlying theory, yet have the theory available to delve into if they want. For example, Section A of Chapter 4 describes how to fit psychometric functions and derive estimates of their critical parameters such as threshold and slope, while Section B describes the theory behind the various fitting procedures. Similarly, Section A in Chapter 6 outlines why $d'$ measures are useful in psychophysics and how they can be calculated using Palamedes, while Section B describes the theory behind the calculations.

1.4 INTRODUCING PALAMEDES

According to Wikipedia, the Greek mythological figure Palamedes is said to have invented “… counting, currency, weights and measures, jokes, dice and a forerunner of chess called pessoi, as well as military ranks.” The story goes that Palamedes also uncovered a ruse by Odysseus. Odysseus had promised Agamemnon that he would defend the marriage of Helen and Menelaus, but pretended to be insane to avoid having to honor his commitment. Unfortunately, Palamedes’s unmasking of
Odysseus led to a gruesome end; he was stoned to death for being a traitor after Odysseus forged false evidence against him. We chose Palamedes as the name for the toolbox for his (presumed) contributions to the art of measurement, interest in stochastic processes (he did invent dice!), numerical skills, humor and wisdom. The Palamedes Swallowtail butterfly (*Papilio Palamedes*) on the front cover also provides the toolbox with an attractive icon.

Palamedes is a set of routines and demonstration programs written in MATLAB® for analyzing psychophysical data (Prins & Kingdom, 2009). The routines can be downloaded from www.palamedestoolbox.org. We recommend that you check the website periodically, because new and improved versions of the toolbox will be posted there for download. Chapters 4–8 explain how to use the routines and describe the theory behind them. The descriptions of Palamedes do not assume any knowledge of MATLAB, although a basic knowledge will certainly help. Moreover, Palamedes requires only basic MATLAB; the specialist toolboxes such as the Statistics Toolbox are not required. We have also tried to make the routines compatible with earlier versions of MATLAB, where necessary including alternative functions that are called when later versions are undetected.

It is important to bear in mind what Palamedes is not. It is not a package for generating stimuli, or for running experiments. In other words it is not a package for dealing with the “front-end” of a psychophysics experiment. The exceptions to this rule are the Palamedes routines for adaptive methods, which are designed to be incorporated into an actual experimental program, and the routines for generating stimulus lists for use in scaling experiments. But by-and-large, Palamedes is a different category of toolbox from the stimulus-generating toolboxes such as VideoToolbox (http://vision.nyu.edu/VideoToolbox/), PsychToolbox (http://psychtoolbox.org/wikka.php?wakka=HomePage), PsychoPy (http://www.psychopy.org, see also Peirce, 2007; 2009) and Psykinematix (http://psykinematix.kybervision.net/) (for a comprehensive list of such toolboxes see http://visionscience.com/documents/strasburger_files/strasburger.html). Although some of these toolboxes contain routines that perform similar functions to some of the routines in Palamedes, for example for fitting psychometric functions (PFs), they are in general complementary to, rather than in competition with Palamedes.

Of the few software packages that deal primarily with the analysis of psychophysical data, psignifit is perhaps the best known (http://bootstrap-software.org/psignifit/; see also Wichmann & Hill, 2001a,b). Like Palamedes, psignifit fits PFs, obtains errors for the parameter estimates of PFs using bootstrap analysis, and performs goodness-of-fit tests of PFs. Palamedes, however, does more; it has routines for calculating signal detection measures, implementing adaptive procedures and analyzing scaling data. For fitting PFs, some advantages of psignifit over Palamedes are that it executes much faster, its core function is a standalone executable file (i.e., it does not require MATLAB) and it has a few useful options that Palamedes
does not have (notably, it allows one to perform a few different types of bootstrap, each with their own advantages). The advantage of Palamedes is that it can fit PFs to multiple conditions simultaneously, while providing the user considerable flexibility in defining a model to fit. Just to give one example, one might assume that the lapse rate and slope of the PF are equal between conditions, but that thresholds are not. Palamedes allows one to specify and implement such assumptions and fit the conditions accordingly. Palamedes can also be used to perform statistical comparisons between models. Examples are to test whether thresholds differ significantly between two or more conditions, to test whether it is reasonable to assume that slopes are equal between the conditions, to test whether the lapse rate differs significantly from zero (or any other specific value), etc. Finally, Palamedes allows one to test the goodness-of-fit of a user-defined model that describes performance across multiple conditions of an experiment.

1.4.1 Organization of Palamedes

All the Palamedes routines are prefixed by an identifier \texttt{PAL}, to avoid confusion with the routines used by MATLAB. After \texttt{PAL}, many routine names contain an acronym for the class of procedure they implement. Table 1.1 lists the acronyms currently in the toolbox, what they stand for, and the book chapter where they are described. In addition to the routines with specialist acronyms, there are a number of general-purpose routines.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Meaning</th>
<th>Chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{PF}</td>
<td>Psychometric function</td>
<td>4</td>
</tr>
<tr>
<td>\texttt{PFBA}</td>
<td>Psychometric function: Bayesian</td>
<td>4</td>
</tr>
<tr>
<td>\texttt{PFML}</td>
<td>Psychometric function: maximum likelihood</td>
<td>4, 8</td>
</tr>
<tr>
<td>\texttt{AMPM}</td>
<td>Adaptive methods: psi method</td>
<td>5</td>
</tr>
<tr>
<td>\texttt{AMRF}</td>
<td>Adaptive methods: running fit</td>
<td>5</td>
</tr>
<tr>
<td>\texttt{AMUD}</td>
<td>Adaptive methods: up/down</td>
<td>5</td>
</tr>
<tr>
<td>\texttt{SDT}</td>
<td>Signal detection theory</td>
<td>6</td>
</tr>
<tr>
<td>\texttt{MLDS}</td>
<td>Maximum likelihood difference scaling</td>
<td>7</td>
</tr>
<tr>
<td>\texttt{PFLR}</td>
<td>Psychometric function: likelihood ratio</td>
<td>8</td>
</tr>
</tbody>
</table>
1.4.2 Functions and Demonstration Programs in Palamedes

1.4.2.1 Functions

In MATLAB there is a distinction between a function and a script. A function accepts one or more input arguments, performs a set of operations and returns one or more output arguments. Typically, Palamedes functions are called as follows:

```matlab
>> [x y z] = PAL_FunctionName(a,b,c);
```

where \(a\), \(b\) and \(c\) are the input arguments, and \(x\), \(y\) and \(z\) the output arguments. In general, the input arguments are “arrays.” Arrays are simply listings of numbers. A scalar is a single number, e.g., 10, 1.5, 1.0e–15. A vector is a one-dimensional array of numbers. A matrix is a two-dimensional array of numbers. It will help you to think of all as being arrays. As a matter of fact, MATLAB represents all as two-dimensional arrays. That is, a scalar is represented as a \(1 \times 1\) (1 row \(\times 1\) column) array, vectors either as an \(m \times 1\) array or a \(1 \times n\) array, and a matrix as an \(m \times n\) array. Arrays can also have more than two dimensions.

In order to demonstrate the general usage of functions in MATLAB, Palamedes includes a function named `PAL_ExampleFunction` which takes two arrays of any dimensionality as input arguments and returns the sum, the difference, the product, and the ratio of the numbers in the arrays corresponding to the input arguments. For any function in Palamedes you can get some information as to its usage by typing `help` followed by the name of the function:

```matlab
>> help PAL_ExampleFunction
```

MATLAB returns:

```matlab
PAL_ExampleFunction calculates the sum, difference, product, and ratio of two scalars, vectors or matrices.

syntax: [sum difference product ratio] = ...
PAL_ExampleFunction(array1, array2)
```

This function serves no purpose other than to demonstrate the general usage of Matlab functions.

For example, if we type and execute:

```matlab
[sum difference product ratio] = PAL_ExampleFunction(10, 5);
```

MATLAB will assign the arithmetic sum of the input arguments to a variable labeled `sum`, the difference to `difference`, etc. In case the variable `sum` did not previously exist, it will have been created when the function was called. In case it did exist, its previous value will be overwritten (and thus lost). We can inquire about the value of a variable by typing its name, followed by <return>:

```matlab
>> sum
```
MATLAB returns:

\[ \text{sum} = 15 \]

We can use any name for the returned arguments. For example, typing:

```matlab
>> [s d p r] = PAL_ExampleFunction(10, 5)
```

creates a variable \( s \) to store the sum, etc.

Instead of passing values directly to the function, we can assign the values to variables and pass the name of the variables instead. For example the series of commands:

```matlab
>> a = 10;
>> b = 5;
>> [sum difference product ratio] = PAL_ExampleFunction(a, b);
```

generates the same result as before. You can also assign a single alphanumeric name to vectors and matrices. For example, to create a vector called \( \text{vect1} \) with values 1, −2, 4, and 105 one can simply type and follow with a <return>:

```matlab
>> vect1 = [1 -2 4 105]
```

Note the square, not round brackets. \( \text{vect1} \) can then be entered as an argument to a routine, provided the routine is set up to accept a \( 1 \times 4 \) vector. To create a matrix called \( \text{matrix1} \) containing two columns and three rows of numbers, type and follow with a <return>, for example:

```matlab
>> matrix1 = [.01 .02; .04 .05; 0.06 0.09]
```

where the semicolon separates the values for different rows. Again, \( \text{matrix1} \) can now be entered as an argument, provided the routine accepts a \( 3 \times 2 \) (rows by columns) matrix.

Whenever a function returns more than one argument, we do not need to assign them all to a variable. Let’s say we are interested in the sum and the difference of two matrices only. We can type:

```matlab
>> [sum difference] = PAL_ExampleFunction([1 2; 3 4], [5 6; ... 7 8]);
```

1.4.2.2 Demonstration Programs

A separate set of Palamedes routines are suffixed by \_Demo. These are demonstration scripts that in general combine a number of Palamedes function routines into a sequence to demonstrate some aspect of their combined operation. They produce a variety of types of output to the screen, such as numbers with headings,
graphs, etc. While these programs do not take arguments when they are called, the user might be prompted to enter something when the program is run, e.g.:

```matlab
>>PAL_Example_Demo
Enter a vector of stimulus levels
```

Then the user might enter something like `[0.1 0.2 0.3]`. After pressing return there will be some form of output, for example data with headings, a graph, or both.

### 1.4.3 Error Messages in Palamedes

The Palamedes toolbox is not particularly resistant to user error. Incorrect usage will more often result in a termination of execution accompanied by an abstract error message than it will in a gentle warning or a suggestion for proper usage. As an example, let us pass some inappropriate arguments to our example function and see what happens. We will pass two arrays to it of unequal size:

```matlab
>>a = [1 2 3];
>>b = [4 5];
>>sum = PAL_ExampleFunction(a, b);
```

MATLAB returns:

```matlab
??? Error using ==> unknown
Matrix dimensions must agree.
Error in ==> PAL_ExampleFunction at 15
sum = array1 + array2;
```

This is actually an error message generated by a resident MATLAB function, not a Palamedes function. Palamedes routines rely on many resident MATLAB functions and operators (such as `+`), and error messages you see will typically be generated by these resident MATLAB routines. In this case, the problem arose when `PAL_ExampleFunction` attempted to use the `+` operator of MATLAB to add two arrays that are not of equal size.

## References


