PSY8219 : Week 11

No Homework Handed Out Today

Homework 10 Due November 12

Readings for Today
   Attaway Chapter 9

Readings for Next Week
   None
Integrating Stimulus Timing and Measuring Responses and Response Times
Examples

Imagine the following experiment (e.g., for fMRI):
- each trial must begin every 2500ms
- on each trial a stimulus is shown
- subjects make a response
- choice and response time is recorded (if on time)
- between each trial there must be at least 500ms blank ITI

let's think about this in pseudocode first
Examples

Imagine the following experiment (e.g., for fMRI):
- each trial must begin every 2500ms
- on each trial a stimulus is shown
- subjects make a response
- choice and response time is recorded (if on time)
- between each trial there must be at least 500ms blank ITI
- what if we want to allow people to respond during ITI?
  but you might worry about keypresses carrying over

let's think about this in pseudocode first
Examples

Imagine the following experiment:

- at the start of each trial, a fixation cross is shown for an exponentially distributed foreperiod (minimum of 250ms, maximum of 2250ms)
- during the fixation cross period, the subject must keep their finger on the space bar the whole time
- after the fixation cross appears, an image is shown
- subject makes a 1-9 preference rating
- choice and response time is recorded

let's think about this in pseudocode first
Examples

Imagine the following experiment:
- each stimulus is shown for 2, 4, or 8 screen refreshes and then cleared
- subject makes a two-alternative forced choice decision any time after stimulus onset
- choice and response time is recorded from stimulus onset
- after response, the stimulus is cleared
- 500ms ITI

Let's think about this in pseudocode first
File I/O
Basic file I/O

Save and Load

create and read .mat file (MAT-file)
Basic file I/O

Save and Load

create and read .mat file (MAT-file)

```matlab
>> save filename
>> save(filename)
```

creates file called filename.mat containing all variables in the workspace
Basic file I/O

Save and Load

create and read .mat file (MAT-file)

```matlab
>> save filename
>> save(filename)
>> save filename var1 var2 var3
>> save(filename, var1, var2, var3)
```

saves particular variables to filename.mat
Basic file I/O

Save and Load

create and read .mat file (MAT-file)

\[
\text{>> load filename} \\
\text{>> load (filename)}
\]

reads file called filename.mat and all its variables into the workspace
Basic file I/O

Save and Load

create and read .mat file (MAT-file)

```matlab
>> load filename
>> load (filename)
>> load filename var1 var2 var3
>> load(filename, var1, var2, var3)
```

load particular variables from filename.mat
Basic file I/O

Save and Load

create and read .mat file (MAT-file)

For any data files used for research of any kind, **NEVER** use MAT-files as your only data file! Please!

This includes data files from experiments, analyses conducted on your data, simulations of models.
Basic file I/O

Save and Load

create and read .mat file (MAT-file)

MAT-files are binary files, which means that they can only be (easily) read by Matlab and generally only the right versions of Matlab. This limits your ability to share files with others and your ability to read your data files years from now.
Recently, someone asked me if I had data from a paper we published in 1994. It was based on data that was more 25 years old.

I was not only able to find the data files, but I was able to easily share the files because they were simply text (ASCII) files.*

* the biggest challenge was that I had ZIPed the files because hard drive space and burnable CD space was so expensive then - we had to find a more general unZIP program that worked
binary vs. ASCII file formats
binary vs. ASCII file formats

Every file is a "binary" file in the sense that it's contents is merely bits (0s and 1s) and bytes (8 bit).
binary vs. ASCII file formats

Every file is a "binary" file in the sense that it's contents is merely bits (0s and 1s) and bytes (8 bit).

e.g.,

```
  bits
01001101
```
Every file is a "binary" file in the sense that it's contents is merely bits (0s and 1s) and bytes (8 bit).

<table>
<thead>
<tr>
<th>binary</th>
<th>hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>A</td>
</tr>
<tr>
<td>1011</td>
<td>B</td>
</tr>
<tr>
<td>1100</td>
<td>C</td>
</tr>
<tr>
<td>1101</td>
<td>D</td>
</tr>
<tr>
<td>1110</td>
<td>E</td>
</tr>
<tr>
<td>1111</td>
<td>F</td>
</tr>
</tbody>
</table>

E.g.,

```
01001101
```

4   D
binary vs. ASCII file formats

Every file is a "binary" file in the sense that it's contents is merely bits (0s and 1s) and bytes (8 bit).

e.g.,

<table>
<thead>
<tr>
<th>bits</th>
<th>bytes</th>
<th>ASCII character</th>
</tr>
</thead>
<tbody>
<tr>
<td>01001101</td>
<td>4D</td>
<td>M</td>
</tr>
<tr>
<td>01101101</td>
<td>6D</td>
<td>m</td>
</tr>
<tr>
<td>00100011</td>
<td>23</td>
<td>#</td>
</tr>
<tr>
<td>Dec</td>
<td>Hx</td>
<td>Oct</td>
</tr>
<tr>
<td>-----</td>
<td>----</td>
<td>-----</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0x00</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>0x01</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>0x02</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>0x03</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>0x04</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>0x05</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td>0x06</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td>0x07</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>0x0B</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>0x0C</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>0x0D</td>
</tr>
<tr>
<td>11</td>
<td>13</td>
<td>0x0E</td>
</tr>
<tr>
<td>12</td>
<td>14</td>
<td>0x0F</td>
</tr>
<tr>
<td>13</td>
<td>15</td>
<td>0x10</td>
</tr>
<tr>
<td>14</td>
<td>16</td>
<td>0x11</td>
</tr>
<tr>
<td>15</td>
<td>17</td>
<td>0x12</td>
</tr>
<tr>
<td>16</td>
<td>20</td>
<td>0x14</td>
</tr>
<tr>
<td>17</td>
<td>21</td>
<td>0x15</td>
</tr>
<tr>
<td>18</td>
<td>22</td>
<td>0x16</td>
</tr>
<tr>
<td>19</td>
<td>23</td>
<td>0x17</td>
</tr>
<tr>
<td>20</td>
<td>24</td>
<td>0x18</td>
</tr>
<tr>
<td>21</td>
<td>25</td>
<td>0x19</td>
</tr>
<tr>
<td>22</td>
<td>26</td>
<td>0x1A</td>
</tr>
<tr>
<td>23</td>
<td>27</td>
<td>0x1B</td>
</tr>
<tr>
<td>24</td>
<td>30</td>
<td>0x1E</td>
</tr>
<tr>
<td>25</td>
<td>31</td>
<td>0x1F</td>
</tr>
<tr>
<td>26</td>
<td>32</td>
<td>0x20</td>
</tr>
<tr>
<td>27</td>
<td>33</td>
<td>0x21</td>
</tr>
<tr>
<td>28</td>
<td>34</td>
<td>0x22</td>
</tr>
<tr>
<td>29</td>
<td>35</td>
<td>0x23</td>
</tr>
<tr>
<td>30</td>
<td>36</td>
<td>0x24</td>
</tr>
<tr>
<td>31</td>
<td>37</td>
<td>0x25</td>
</tr>
</tbody>
</table>

Note: These codes are DIFFERENT from keyboard key codes in Psych Toolbox. ASCII refers to a symbols. Keycodes refer to a keyboard key.

Source: [www.LookupTables.com](http://www.LookupTables.com)
binary vs. ASCII file formats

ASCII files are read by every operating system and many many programs exactly the same way. The extension (.txt) by itself doesn't make it a "text file" (ASCII), but does give a clue to the O/S about what default program to use to read the file.

Text files (.txt)
Configuration files (.ini)
Batch files (.bat)
Scripts and source code (.m, .c, .py)

If you have a data file (or program code) that is in ASCII format you are guaranteeing it can be read by anyone anywhere in the world.
binary vs. ASCII file formats

binary files are also made of bits and bytes but the way those bits and bytes are interpreted depends on the particular O/S and program.

you either need to have the program (and perhaps the O/S) that created the file or hope that a newer version (or a different program) can read those files.

e.g., I have some analysis files from graduate school created by a program that hasn't existed for a decade.
binary vs. ASCII file formats

If you open a binary file in an application that doesn't understand it, it probably defaults to an ASCII interpretation, which could be garbage...
low(ish)-level file I/O

opening a file for *writing* in ASCII format

```c
fp = fopen('data.dat', 'w');
```

file name
low(ish)-level file I/O

opening a file for writing in ASCII format

```c
fp = fopen('data.dat', 'w');
```

open for writing
(discard existing contents)
low(ish)-level file I/O

opening a file for writing in ASCII format

```c
fp = fopen('data.dat', 'w');
```

file "identifier"
(like a handle for the file)
low(ish)-level file I/O

opening a file for writing in ASCII format

```c
fp = fopen('data.dat', 'w');

fprintf(fp, '%s %12.4f %d\n', 'this is a string', pi, 42);
```

write formatted data to file using fprintf and the file identifier fp
low(ish)-level file I/O

opening a file for **writing** in ASCII format

```c
fp = fopen('data.dat', 'w');

fprintf(fp, '%s %12.4f %d\n', 'this is a string', pi, 42);

fclose(fp);
```

`close the file`
low(ish)-level file I/O

opening a file for reading in ASCII format

```c
fp = fopen('data.dat', 'r');
```

open for reading
low(ish)-level file I/O

opening a file for reading in ASCII format

```c
fp = fopen('data.dat', 'r');

fscanf();
geticl();
```

various methods for reading from a file

```c
fclose(fp);
```
low(ish)-level file I/O

opening a file for reading in ASCII format

```c
fp = fopen('data.dat', permission);
```

permission

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'r'</td>
<td>Open file for reading.</td>
</tr>
<tr>
<td>'w'</td>
<td>Open or create new file for writing. Discard existing contents, if any.</td>
</tr>
<tr>
<td>'a'</td>
<td>Open or create new file for writing. Append data to the end of the file.</td>
</tr>
<tr>
<td>'r+'</td>
<td>Open file for reading and writing.</td>
</tr>
<tr>
<td>'w+'</td>
<td>Open or create new file for reading and writing. Discard existing contents, if any.</td>
</tr>
<tr>
<td>'a+'</td>
<td>Open or create new file for reading and writing. Append data to the end of the file.</td>
</tr>
<tr>
<td>'A'</td>
<td>Open file for appending without automatic flushing of the current output buffer.</td>
</tr>
<tr>
<td>'W'</td>
<td>Open file for writing without automatic flushing of the current output buffer.</td>
</tr>
</tbody>
</table>
low(ish)-level file I/O

e.g., fscanf

let's see what it does with data.txt

fp = fopen('data.txt', 'r');
fn = fscanf(fp, '%s');
fprintf('fname = %s\n', fn);
fclose(fp);
low(ish)-level file I/O

e.g., fscanf

let's see what it does with data.txt

```c
fp = fopen('data.txt', 'r');
fn = fscanf(fp, '%s');
fprintf('fname = %s\n', fn);
fclose(fp); the string keeps going until a "white space" is hit
```
low(ish)-level file I/O

e.g., line-by-line with `fgetl()`

each line is read as a string; you need to use string operations to decode that into what you need
low(ish)-level file I/O

fp = fopen('data.txt', 'r');
line = fgetl(fp); % first line
fname = line;
line = fgetl(fp); % second line
seed = str2num(line);
line = fgetl(fp); % third line
Timestamp = str2num(line);
line = fgetl(fp); % fourth line
arr = str2num(line);
Nrows = arr(1);  
Ncols = arr(2);
Try these:
>> str2num('23');
>> str2num('4 5 12 19');
>> str2num('2, 5, 12, 54.2');
>> str2num('5; 4; 12');

str2num can decode single numbers, space delimited, comma delimited, semi-colon delimited, tab-delimited strings (lines of a text file)

str2num can't understand a line like this:
str2num('4 A 153 K');
low(ish)-level file I/O

line = ' 4 A 1 453 K';
trial = str2num(line(1:3));
code = line(7);
response = str2num(line(10:11));
RT = str2num(line(13:17));
stimcode = line(21);

Well-structured data files are easy to read. Every element on a line starts at a particular position along that line.

Poorly-structured data files can take far more work to decode.
low(ish)-level file I/O

Poorly-structured:

```c
fp = fopen('data1.txt', 'w');
fprintf(fp, '%c %s %d %f
', 'A', 'Tom', 23, 43.12);
fprintf(fp, '%c %s %d %f
', 'C', 'Billy', 1, 143.12);
fprintf(fp, '%c %s %d %f
', 'G', 'Jo', 153, 152.12);
fclose(fp);
```
low(ish)-level file I/O

Well-structured:

```c
fp = fopen('data1.txt', 'w');
fprintf(fp, '%c %10s %5d %12.4f
', 'A', 'Tom', 23, 43.12);
fprintf(fp, '%c %10s %5d %12.4f
', 'C', 'Billy', 1, 143.12);
fprintf(fp, '%c %10s %5d %12.4f
', 'G', 'Jo', 153, 152.12);
fclose(fp);
```
Recommendations

Data files:
- should be text (ASCII) files (plus MAT files okay too)
- should include a header with a description of the contents of the data file (what it is, how it is structured, what it contains)
- it should include every piece of information needed to exactly reconstruct the conditions under which the data were originally collected
- all relevant measurements should be recorded, even if you do not think you need them
- never simply record only summary data, but record every single trial of an experiment in excruciating detail
Recommendations

Data files:
- the file name should be parsable

E23-01-03-034.DAT

which experiment
which condition
which session
which subject