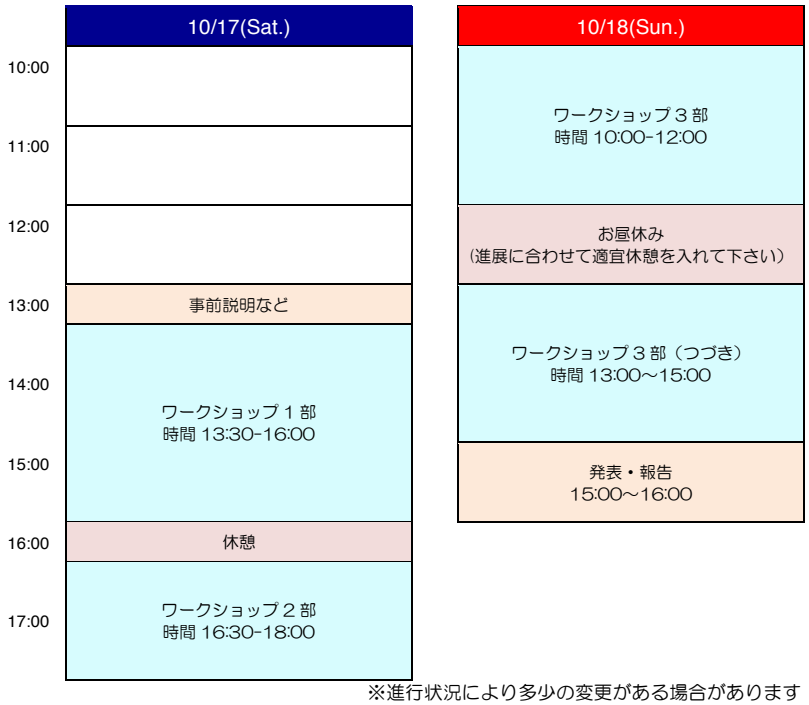
VSJ
Programing Workshop
2015

@The University of Electro-

Communications

2015.10.17(Sat.) – 18 (Sun.)

スケジュール



※進行状況により多少の変更がある場合があります（特に10/17）

連絡事項

- 建物内にはゴミ箱はありません．建物玄関より外にでて右手（喫煙所の近く）にあるゴミ箱に分別して捨てて下さい．
- ニ日目のお昼ご飯は各自で用意して下さい．（コンビニの場所等は地図を参照してください）

本ワークショップのねらい

- 実験プログラミングを自力で作成できるよう，その特徴を学ぶ

- 実験プログラミングの学習≠プログラミング言語の学習
  プログラミング言語の中身に関する内容の学習がメインテーマではない
  目的指向的な学習（やりたいことを実現するにはどうするか）
  グループ形式によるワークショップ（１人ではなく，みんなで学習する）

- 実験プログラミングに必要な要素（パーツ）を学習する．
  パーツの種類，役割，組み立て方

目次

第０章　Psychtoolbox (PTB)について

「Psychtoolbox (PTB)」は MATLAB または Octave というソフトウェア上で使うことが出来る実験用のツールです。ディスプレイ画面上に刺激提示するのに有用な関数が含まれています。MacOSX, windows, Linux で動きます。

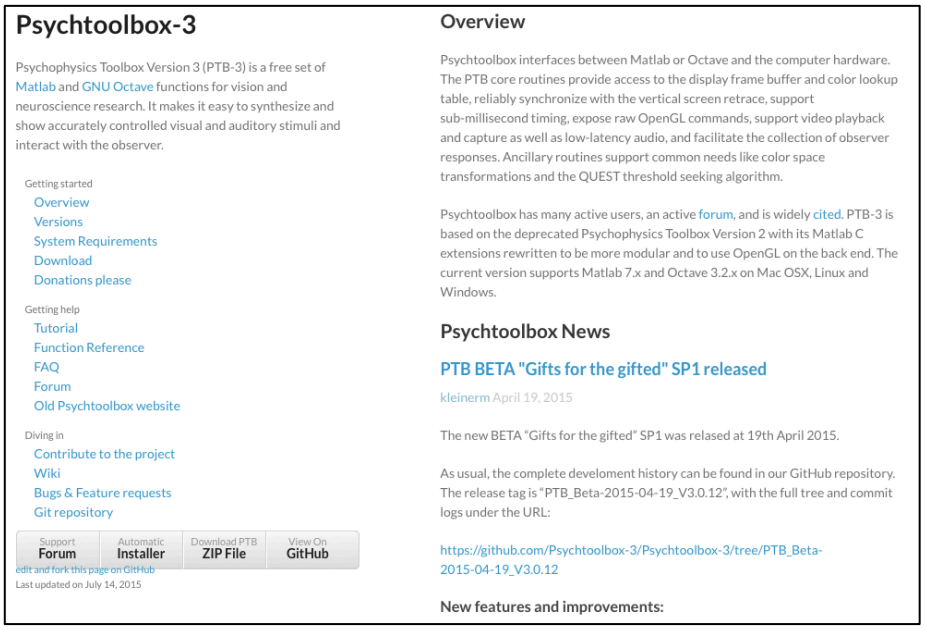
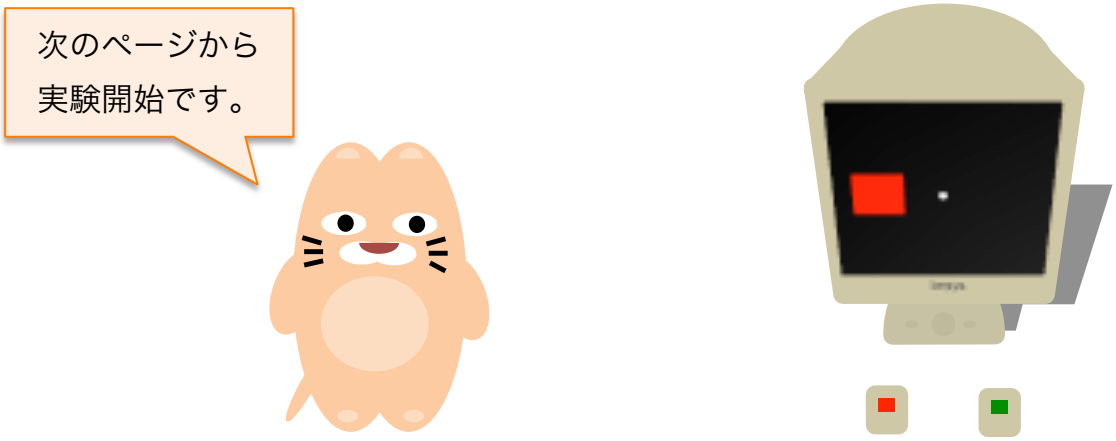


図 0- 1 PTB 本家のホームページ(http://psychtoolbox.org/)



第 1 章 Psychophysics Toolbox (PTB)の実験をしよう

1.1. サンプルプログラムを実行する。

http://visitope.org/workshop/ProgrammingWorkshop/PTB/
から「PWS2015PTB」をダウンロードしてデスクトップに保存してください。

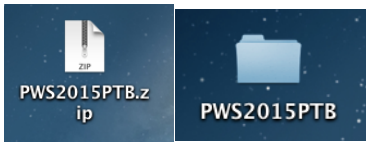
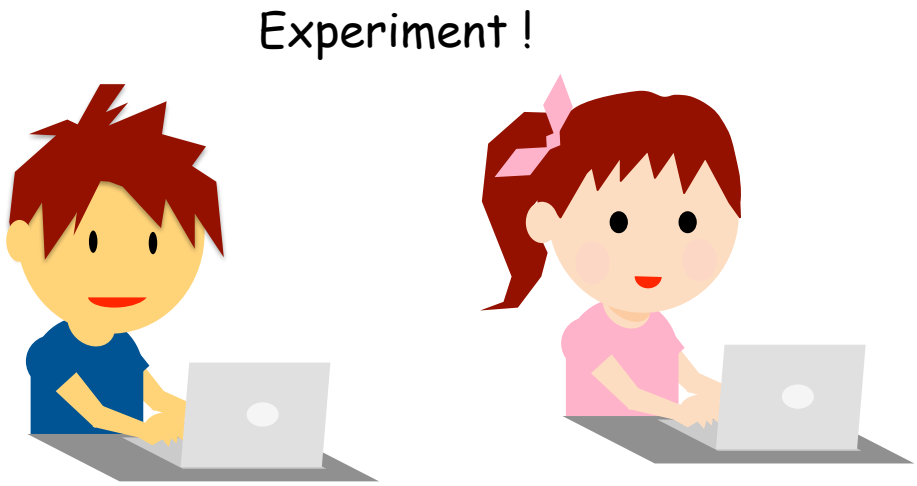


図 1- 1 サンプルプログラムのフォルダ



① MATLAB または Octave を開く

Mac は「アプリケーション」，Windows は，「スタート→すべてのプログラム」をクリックし，MATLAB または Octave のアイコンをクリックしてください。

以下のように，MATLAB の表示画面は以下のように分かれています。

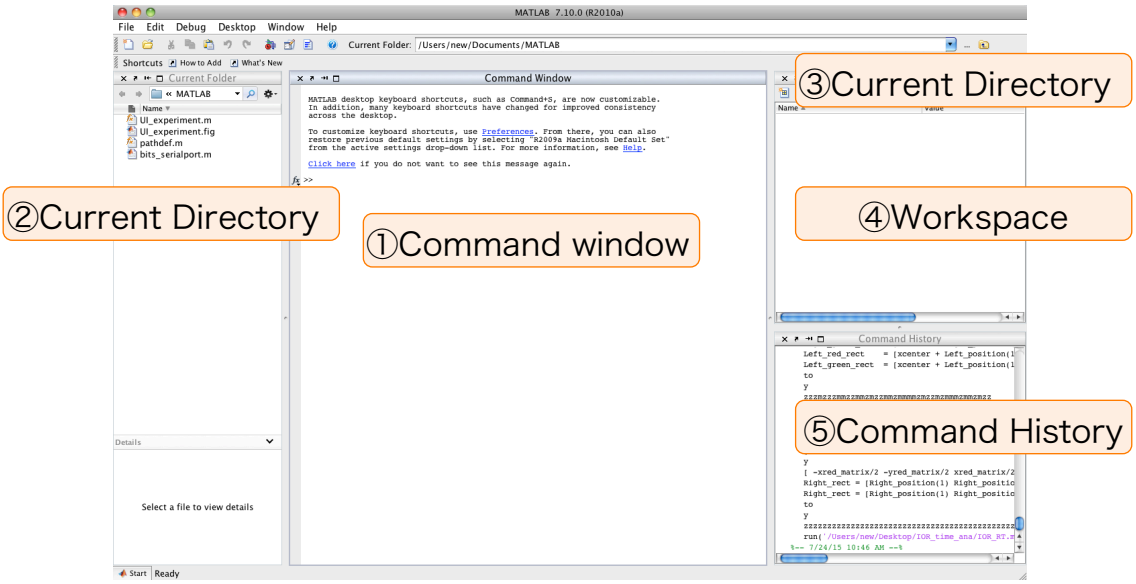


図 1- 2 MATLAB の表示画面例（MATLAB_R2010a)

① Command Window (コマンドウィンドウ)
プログラムの起動や，参加者情報等を入力するのに使います。エラーメッセージも表示されます。

② Current Directory (現在のフォルダ)
開いているフォルダの中身が表示されます。

③ Current Directory (フォルダの場所を表示)
クリックするとフォルダを指定する事ができます。

④ Workspace（ワークスペース）
「変数」が表示されます。MATLAB では，変数名をダブルクリックするとエクセルのような表が開き，中身を確認できます。

⑤ Command History（コマンド履歴）
コマンドウィンドウ上に打ち込んだ履歴が表示されます。

② 現在のフォルダを変える

「カレントフォルダ」を「PWS2015PTB」に変えます。右上の**「…」**をクリックしてフォルダを指定してください。

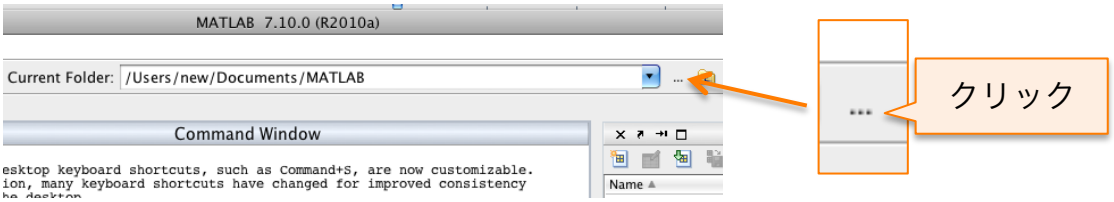


図 1- 3 現在のフォルダを変える

「PWS2015PTB」のフォルダを指定して「Open」してください。また，左側の「カレントフォルダ」は PWS2015PTB フォルダの中身が表示されています。

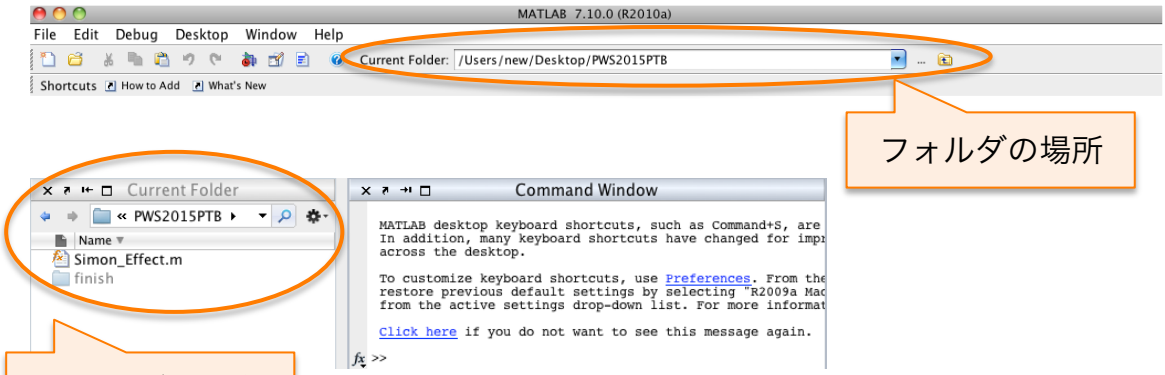


図 1- 4 カレントフォルダの中身の表示

「コマンドウィンドウ」にプログラム名「**Simon_Effect**」を入力してください。(エラーになる場合は，スペルや大文字小文字が合っているか確認してください。)

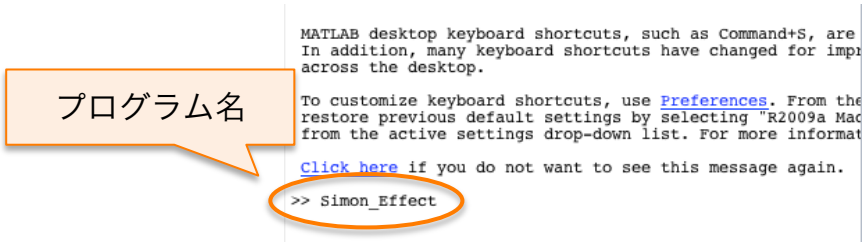


図 1- 5 コマンドウィンドウでの実行

「Subject Name?」と表示されます。名前のイニシャルを入力してください。入力した名前で結果ファイルが出力されます。

```
Subject Name?TO
```

確認メッセージが表示されます。このファイル名で良い場合は「y」 を，変える場合は「n」を入力してください。
(「n」を入力した場合にはプログラムが終了します。)

```
File name, OK? y:yes, n:no....
```

以下のようなメッセージが画面に表示されます。しばらく止まる場合がありますが，そのまま待っていてください（画面と同期しているかを確かめるテストを行っています。)

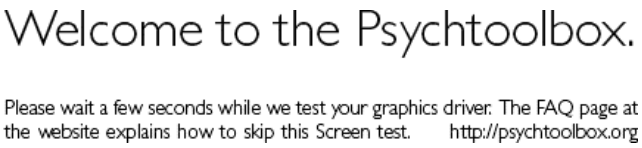


図 1- 6 画面上に表示されるメッセージ

1.2. 実験の説明

（教示）画面中央の「注視点（白い点）」を見てください。右側か左側に「緑色か赤色の四角」が提示されます。どちらの色が提示されたか判断してください。赤色の場合は「z キー」，緑色の場合は「m キー」を押してください。各キーは左手または右手の人差し指で押してください。出来るだけ正確かつ早く押してください。四角の提示位置は課題とは無関係なので気にしないでください。約５分程度で実験が終了します。実験が終わるとデスクトップ画面に戻ります。

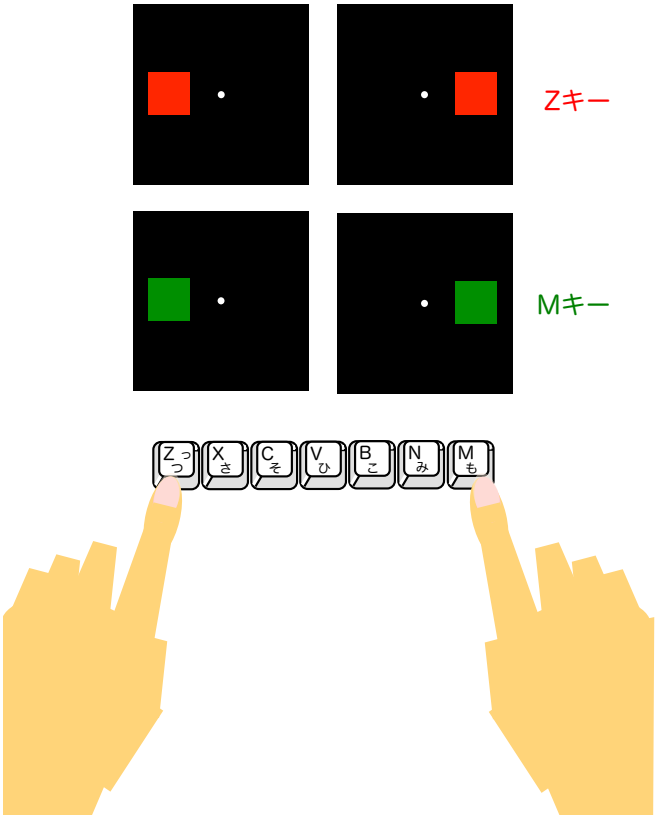
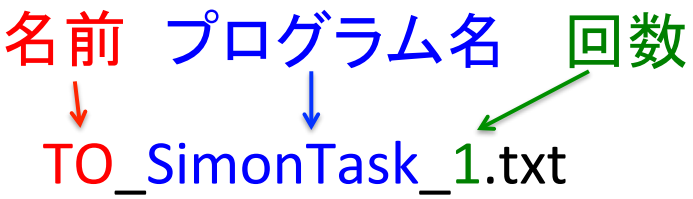


図 1- 7 サンプルプログラムの実験

赤色の四角は「z キー」，緑色の視覚は「m キー」を押してください。

1.3. 実験結果をエクセルで開く方法

実験が終了するとカレントフォルダに**「TO_SimonTask_1.txt, TO_SimonTask_1.mat」**のような２つのファイルが出来ます。



以下はエクセルで分析する方法です。

1. カレントフォルダにある結果ファイル(.txt)を右クリックし，「Locate on Deck」をクリックしてください。

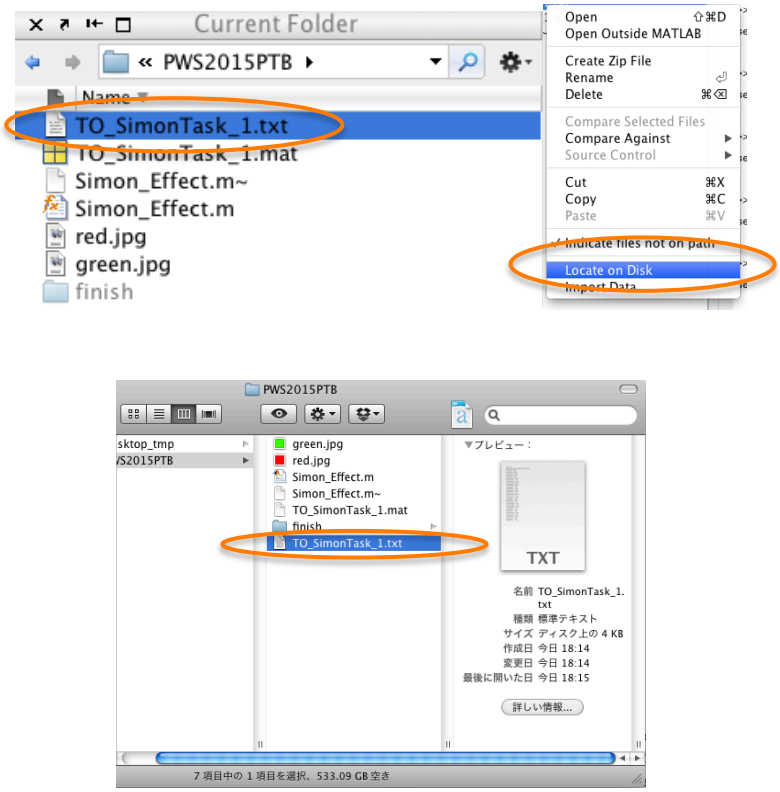


図 1- 8 結果ファイルの場所

2. 結果ファイルを右クリックして「このアプリケーションで開く」,「Microsoft Excel」を選択してください。

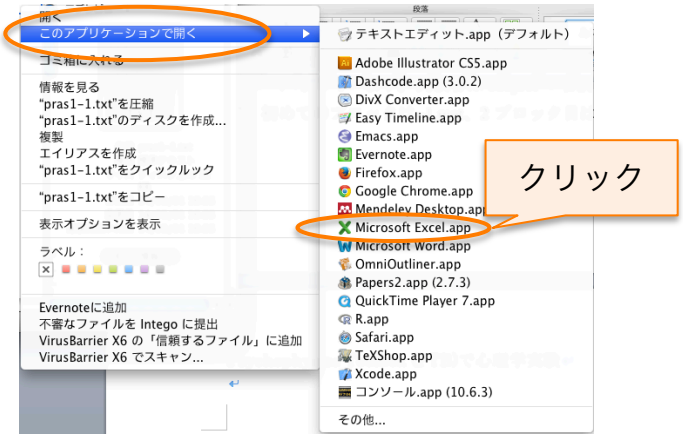


図 1- 9 エクセルで結果を開く

3. Excel で結果を開き,「A の列」を選択し,「データ」タブの「ツール」の中にある「区切り位置」を選択して「次へ」をクリックしてください。

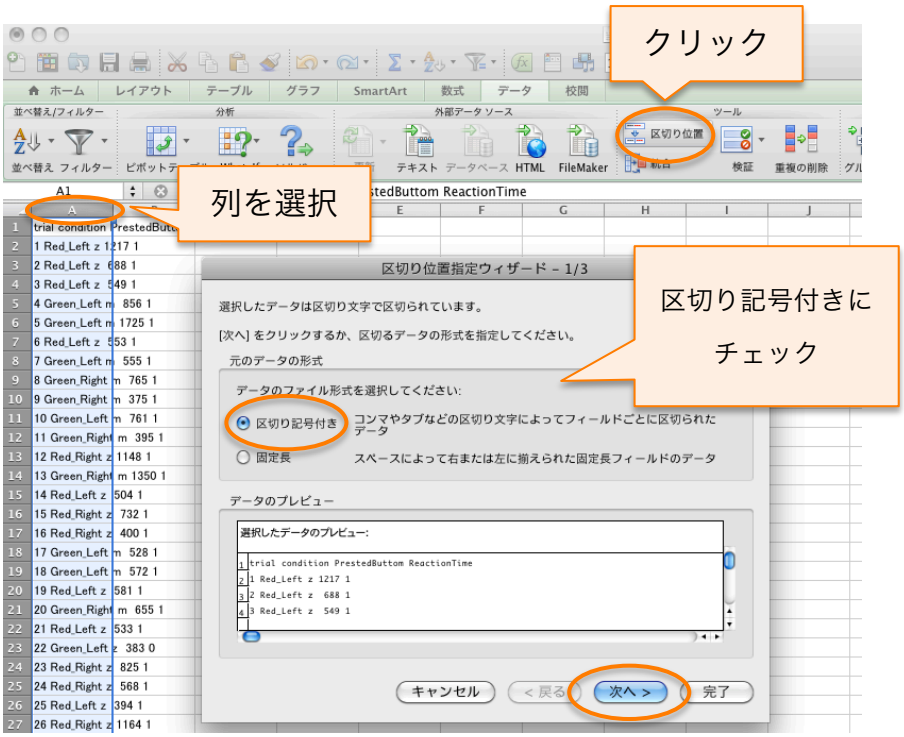


図 1- 10 区切り位置の指定の仕方

4. 区切り文字の「スペース」を選択して「次へ」をクリック

図 1- 11 ダイアログでスペースを選択

5. スペースごとに別の列にデータを分けることが出来ます。

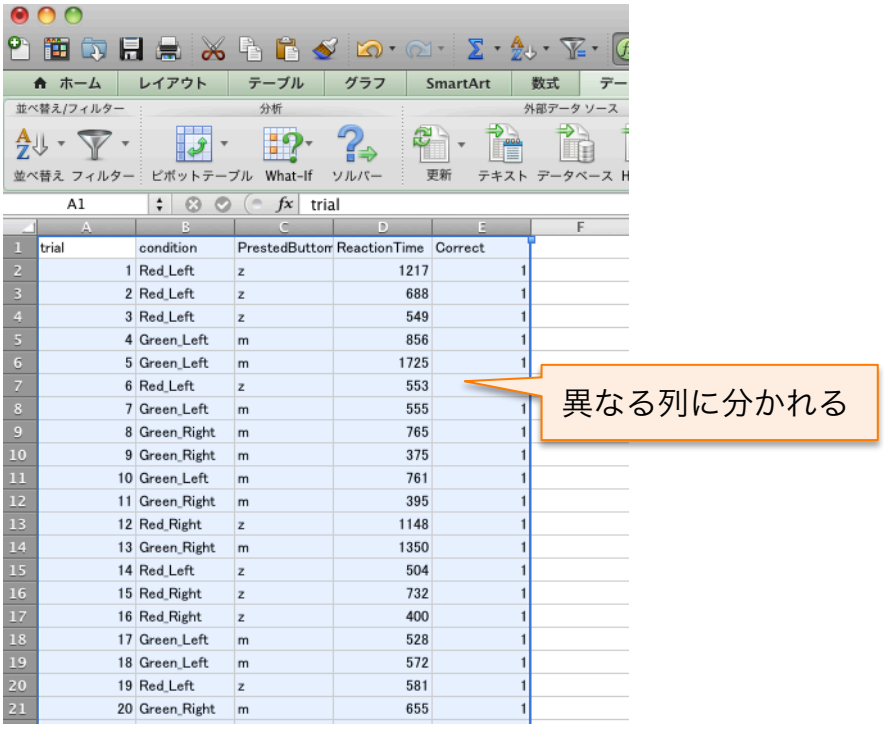


図 1- 12 完成

★ ステップアップ：「mat ファイル」を分析する方法について

「mat ファイル」で保存すると，たくさんの情報をまとめて保存出来ます。また，MATLAB 上で分析するプログラムを作れば，１クリックで複雑な分析をすることもできます。

「mat ファイル」をエクセルで開く方法

コマンドウィンドウにファイ名を入力して, MATLAB に読み込んでください。

「Data_raw」をダブルクリックしてください。シートが開きます。

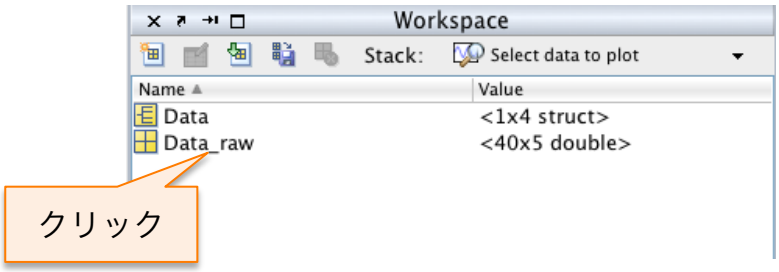
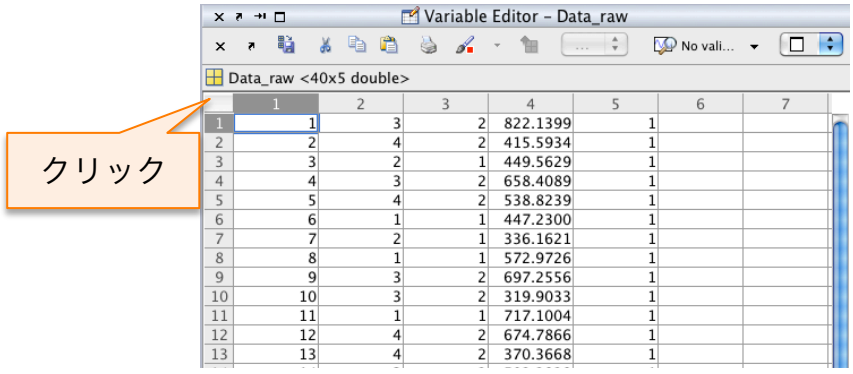


図 1- 13 mat ファイルを読み込む

シートの左上をクリックすると，全体が選択されます。コピーして，エクセル上でペーストすれば，エクセルで分析できます。



	1	2	3	4	5	6	7
1	1	3	2	822.1399	1		
2	2	4	2	415.5934	1		
3	3	2	1	449.5629	1		
4	4	3	2	658.4089	1		
5	5	4	2	538.8239	1		
6	6	1	1	447.2300	1		
7	7	2	1	336.1621	1		
8	8	1	1	572.9726	1		
9	9	3	2	697.2556	1		
10	10	3	2	319.9033	1		
11	11	1	1	717.1004	1		
12	12	4	2	674.7866	1		
13	13	4	2	370.3668	1		

図 1- 14 Data_raw ファイル

1.5. サイモン効果とは？

課題と無関係な特徴（位置）が反応の選択に干渉するという効果です。刺激と反応の組み合わせが適合的な場合（例えば，刺激が左に提示されて反応も左の場合）の方が，非適合的な場合（例えば，刺激が左に提示されて反応は右の場合）に比べて反応が早くなります。赤色の刺激に対応するキーが左側（z キー），緑色の刺激に対応するキーが右側（m キー）にありました。赤色が左側，緑色が右側の場合の方が，赤色が右側，緑色が左側の場合よりも早くなることが予想されます。

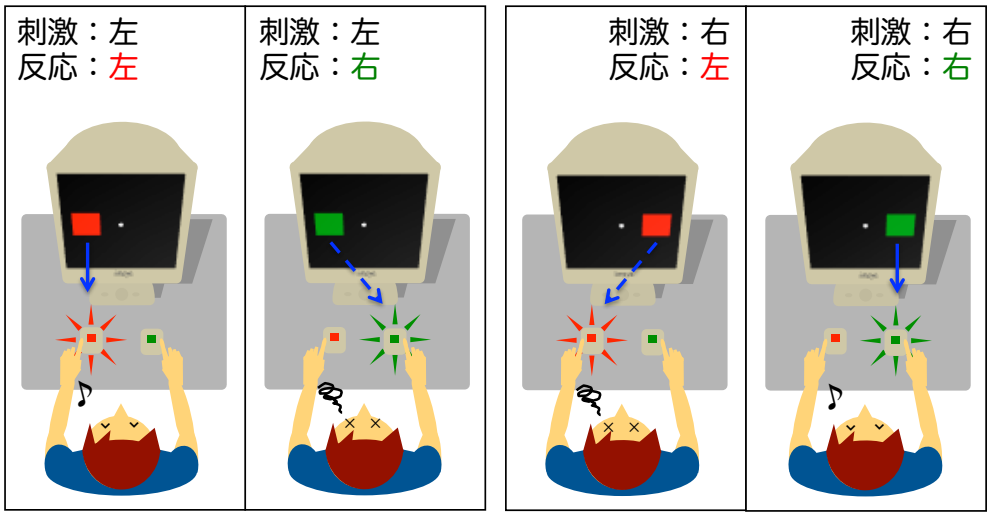


図 1- 15 サイモン課題の刺激提示例

1.6. サイモン効果の実験手続き

以下は，サイモン効果の実験結果をレポートにまとめる際の一例です。
（赤色の文字はプログラムのどこに書かれているでしょう？）

実験計画

独立変数は「位置に関する条件（左側か右側か）」と「色に関する条件（赤色か緑色か）」が組合わさっていました。位置や色といったレベルは「要因」，左側か右側か，赤色か緑色かといったレベルは「水準」と呼ばれます。２つのそれぞれの要因が２つの水準を持つため，２要因２水準の実験計画と呼ばれます。

本実験では各条件に **10 回の繰り返し**がありました。そのため２要因 × ２水準 × 10 回の繰り返しで，全部で **40 試行**の実験でした。

従属変数として，四角形が画面に提示されてから反応キーが押されるまでの反応時間，反応の正答率を測定しました。

刺激

刺激は，**赤色 (RGB: 255, 0 , 0) または緑色 (RGB: 0, 255, 0) の四角形 (縦 300 pixel × 横 300 pixel)** を用いました。注視点として白色の円を画面の中央に提示しました。四角形は画面の中心から左または右に **300 pixel 離れた位置**に提示されました。背景は**黒色**でした。

		要因１：刺激の提示位置	
		左	右
要因２：刺激の色	赤	左・赤	右・赤
	緑	左・緑	右・緑

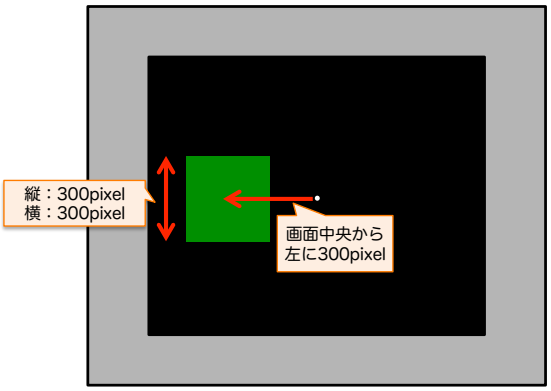


図 1- 16 実験計画と画面提示例

手続き

各試行では画面中央に注視点が **1 秒間**提示されました。その後に注視点の右側または左側に赤色か緑色の四角形が提示されました。「右・赤条件」，「右・緑条件」，「左・赤条件」，「右・緑条件」の順番はランダムに決定されました。実験参加者の課題は，提示された四角形の色が赤色か緑色かを識別し反応キーを押し分ける事でした。赤い四角形が提示された場合には**「z キー」**を，緑色の四角形が提示された場合には**「m キー」**を押す事が求められました。

サイモン効果のプログラムは少し複雑そうなので，まずはサイモン効果の簡易版プログラムから見ていきます。

第２章 実験プログラムの一部を変えよう

2.1.プログラムを見てみよう

サイモン効果の簡易版のプログラムを用いて，実験を作っていきます。まずはプログラムを実行してください。コマンドウィンドウに「Presentation」と入力してください。画面に刺激が５回提示されます。

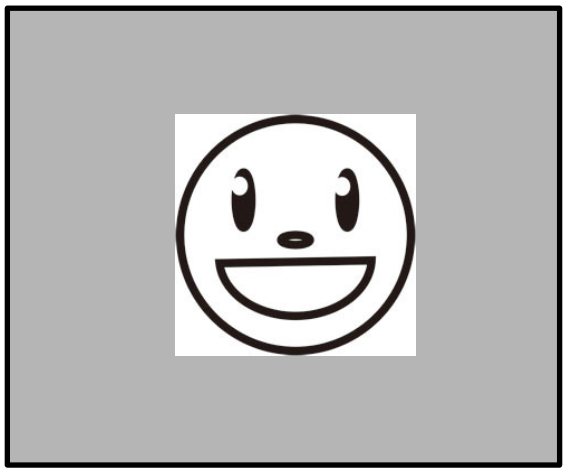


図 2- 1 Presentation デモ

2.2. エディターでプログラムを開こう

プログラムの作成，編集にはエディターを使用します。コマンドウィンドウに「edit Presentation」と入力してください（間にスペースが必要）。

```
>> edit Presentation
>>
```

黒色と紫色の文字はプログラムのコードです。緑色の文字はプログラムの説明等（メモ書き）です。（拡張子は「.m」です。「M ファイル」またはスクリプトファイルと呼ばれます。）

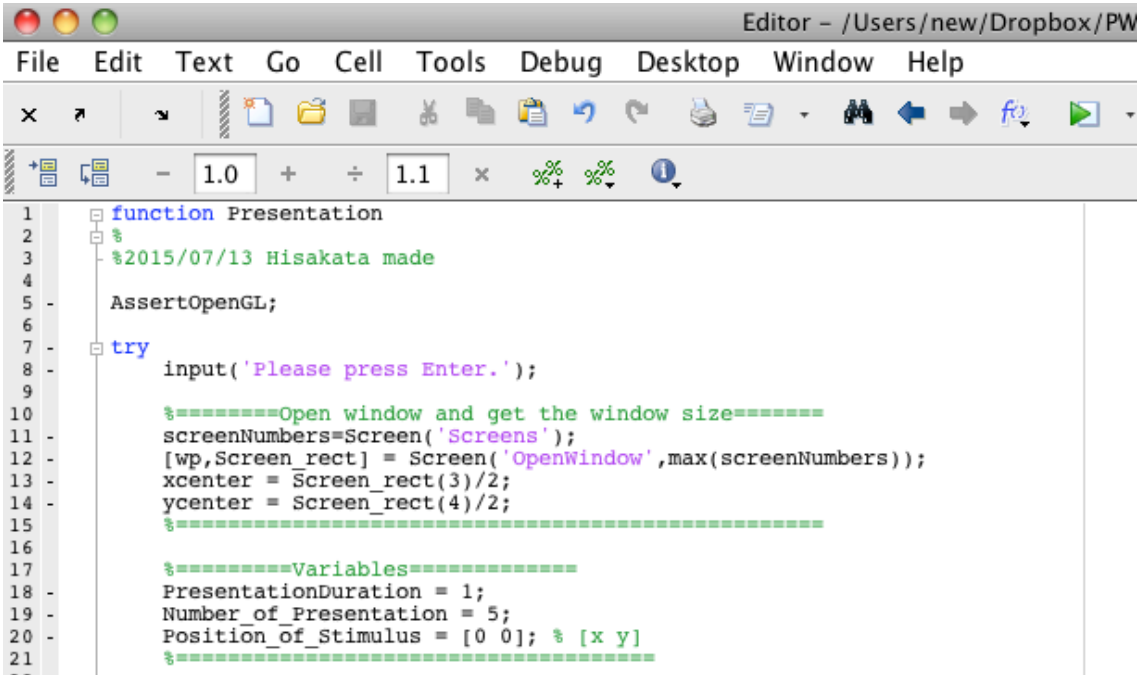


図 2- 2 コマンドウィンドウ

行の最初に「%」のマークをつけるとコメントになります。コメントは，プログラムを実行した際に無視されます。実験プログラムの説明や作成者，日時等の情報をコメントにします。

2.3. プログラムのソースコードの構成

プログラムは以下のような構成になっています。

・赤い四角：条件の初期設定

・青い四角：画面描画

プログラムコードを赤と青のマーカーで囲ってください。また，各プログラムが何をしているのかメモをしてください。

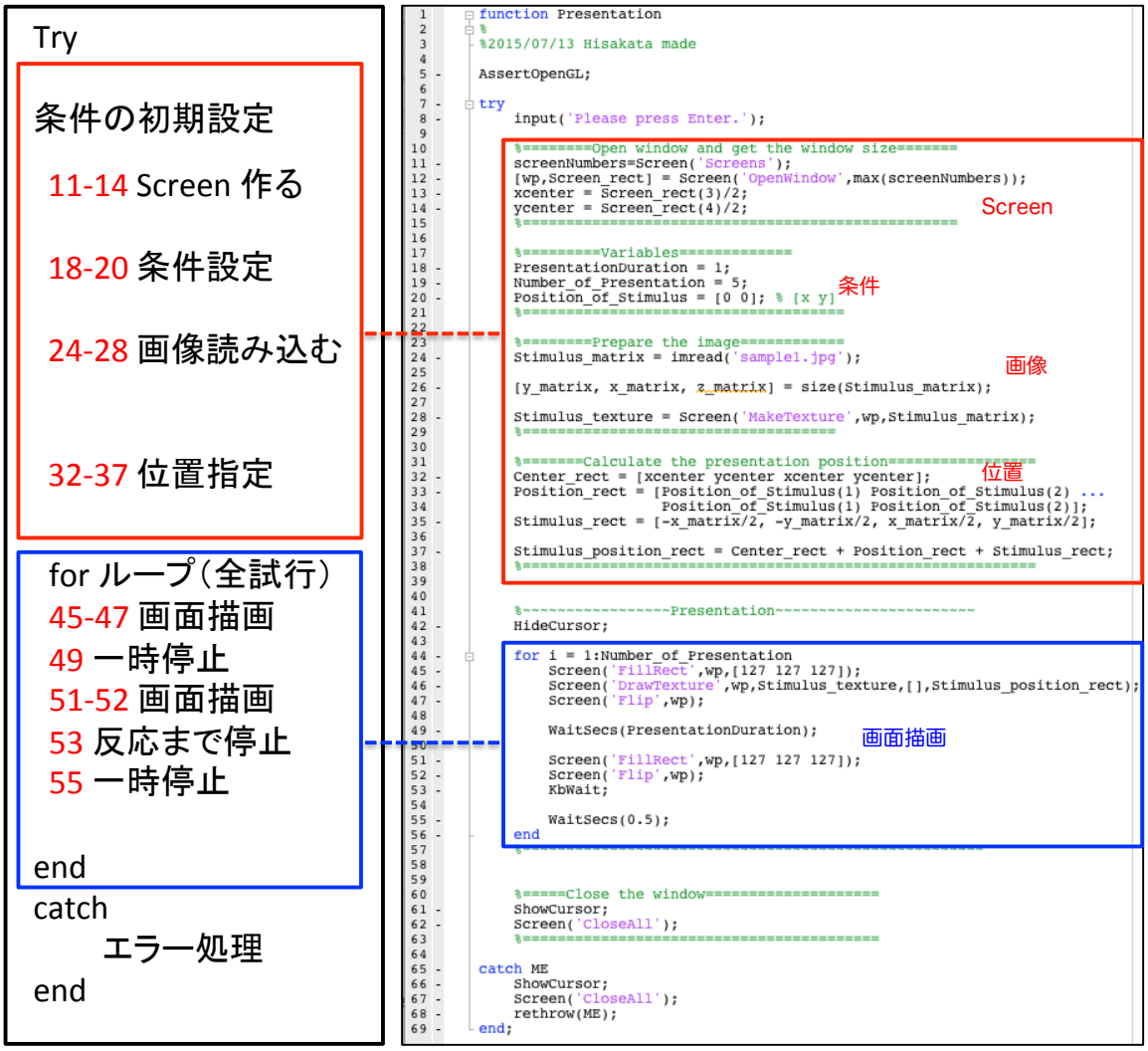


図 2- 3 コードの構成

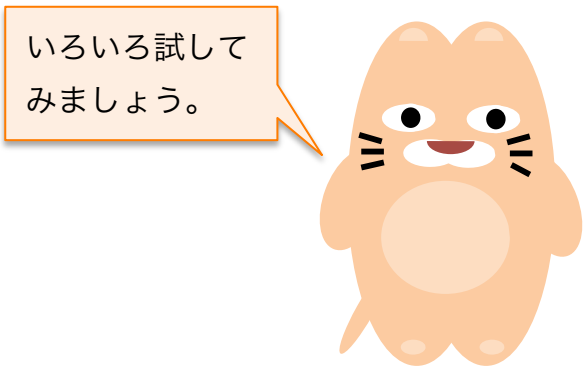
2.4. サイモン効果の簡易版「Presentation」をいろいろ変えてみよう

それでは，「条件の初期設定」をいろいろ変えてみましょう。

```
10        %========Open window and get the window size=======
11 -      screenNumbers=Screen('Screens');
12 -      [wp,Screen_rect] = Screen('OpenWindow',max(screenNumbers));
13 -      xcenter = Screen_rect(3)/2;
14 -      ycenter = Screen_rect(4)/2;
15        %================================================
16
17        %=========Variables=============
18 -      PresentationDuration = 1;
19 -      Number_of_Presentation = 5;
20 -      Position_of_Stimulus = [0 0]; % [x y]
21        %================================
22
23        %========Prepare the image============
24 -      Stimulus_matrix = imread('sample1.jpg');
25
26 -      [y_matrix, x_matrix, z_matrix] = size(Stimulus_matrix);
27
28 -      Stimulus_texture = Screen('MakeTexture',wp,Stimulus_matrix);
29        %====================================
30
31        %=======Calculate the presentation position=================
32 -      Center_rect = [xcenter ycenter xcenter ycenter];
33 -      Position_rect = [Position_of_Stimulus(1) Position_of_Stimulus(2) ...
34                         Position_of_Stimulus(1) Position_of_Stimulus(2)];
35 -      Stimulus_rect = [-x_matrix/2, -y_matrix/2, x_matrix/2, y_matrix/2];
36
37 -      Stimulus_position_rect = Center_rect + Position_rect + Stimulus_rect;
38        %============================================================
```

提示時間
繰り返し数
刺激の位置
別の画像に

図 2- 4 条件の初期設定を変えてみよう。



2.4.1. スクリーンを作る
(Presentation.m Line 10-15)

ここはおまじないとして覚えてしまいましょう。

左ページ

```
10    %========Open window and get the window size=======
11    screenNumbers=Screen('Screens');
12    [wp,Screen_rect] = Screen('OpenWindow',max(screenNumbers));
13    xcenter = Screen_rect(3)/2;
14    ycenter = Screen_rect(4)/2;
15    %================================================
```

右ページ

```
10    %========提示するモニタの準備をします=======
11    11% 初期化１
12    12% 初期化２
13    13-14  % 画面の中心座標の取得
```

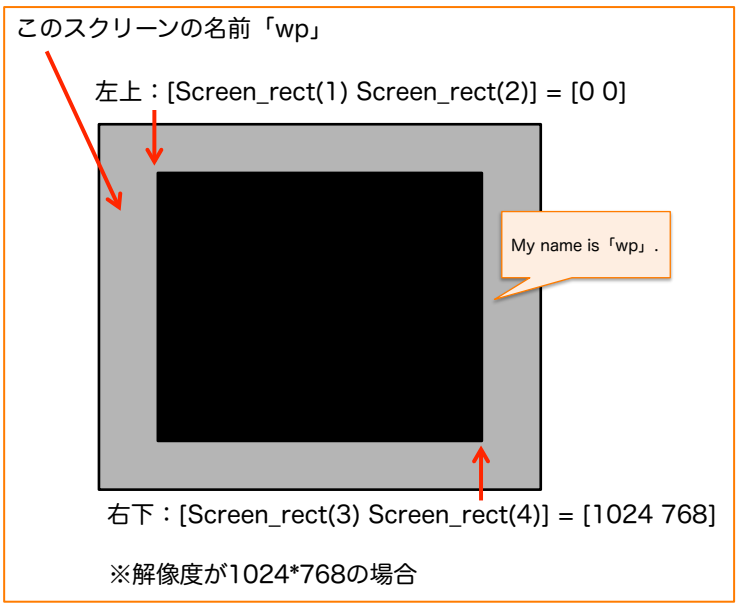


図 2- 5 スクリーンを作る。

初期化１（PTB関数）

```
11 screenNumbers = Screen( 'Screens' );
```

PTBを使う入り口です。何台のディスプレイがつながっているか数えます。１台の場合は，「０」が入ります。２台接続されている場合は，「１」と「２」が入ります。「１」はメイン，「２」はサブディスプレイの番号です。

初期化２（PTB関数）

名前　左上と右下　どのディスプレイに表示するか？

```
12 [ wp, Screen_rect ] = Screen( 'OpenWindow' , max(screenNumbers) );
```

この命令で「screen」が開きます。ついでに「wp」と「Screen_rect」の２つの情報をもらいます。２つ目の引数には表示したいディスプレイを指定します。サブディスプレイに表示するためには「max関数」で大きい値，メインディスプレイに表示したい場合は「min関数」で小さい値を取得してください。「wp」はスクリーンの名前です。以降はこの名前を指定してスクリーン関数を呼び出します。「wp」はウィンドウポインタと呼ばれています。

「Screen_rect」にはスクリーンの左上と右下の座標が入ります。

```
Screen_rect =

           0           0        1024         768

>> |
```

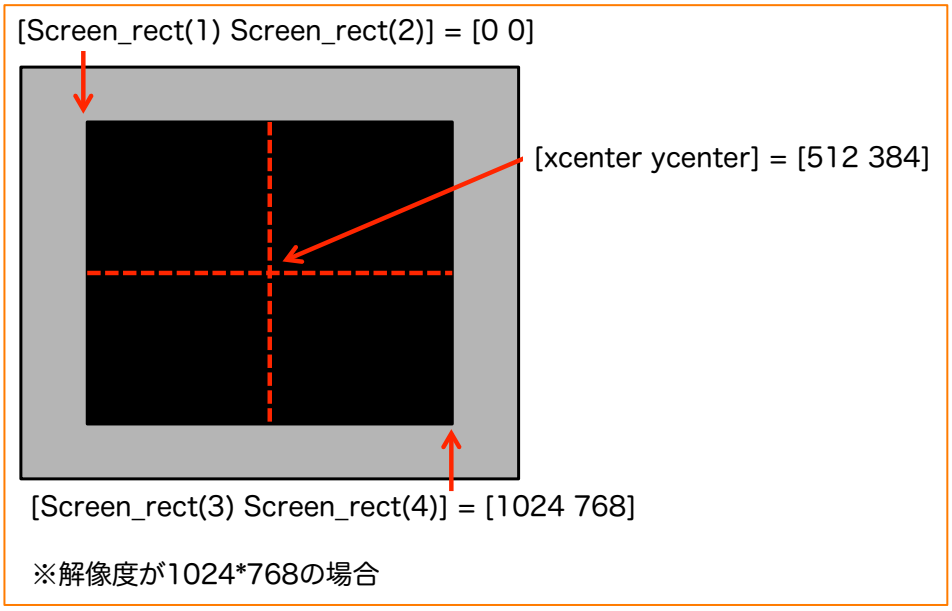


図 2- 6 刺激の提示位置について

画面の中心座標取得

```
13 xcenter = Screen_rect (3)/2;
14 ycenter = Screen_rect (4)/2;
```

「画面の中心の x 座標と y 座標」を取得します。以降は中心の座標を基準として刺激の位置を指定していきます。

```
Screen_rect =

           0           0        1024         768

>> xcenter = Screen_rect(3)/2

xcenter =

   512

>> ycenter = Screen_rect(4)/2

ycenter =

   384
```

2.4.2. 画像を読み込む

(Presentation.m Line 23-29)

ここは３つセットで覚えてしまいましょう。

① MATLABに読み込む。「 imread 」

② PTBに画像を渡す。「 Screen('MakeTexture') 」

③ PTBで画像を提示する。「 Screen('DrawTexture') 」と「 Screen('Flip') 」

ここでは①と②を説明します。（③はプログラムのどこに書いてある？）

左ページ

```
23    %========Prepare the image========
24    Stimulus_matrix = imread('sample1.jpg');
25
26    [y_matrix, x_matrix, z_matrix] = size(Stimulus_matrix);
27
28    Stimulus_texture = Screen('MakeTexture',wp,Stimulus_matrix);
29    %=================================
```

① MATLAB に読み込み

大きさを取得

② PTB に渡す

右ページ

```
23    %=======提示する画像の準備をします========
24    24 % imread は Matlab の関数で、画像を読み込んで matlab で扱える行列に変換します。
25
26    26 % 行列のサイズはそのまま、画像のサイズ(ピクセル)になります。
27
28    28 % PTBで提示できるように、Screen関数のMakeTextureを使って描画する準備をします。
```

画像の読み込み

配列で保存　　画像ファイル

```
24 Stimulus_matrix = imread('sample1.jpg');
```

図 2- 7 画像の読み込み

「imread」は画像ファイルを読み込むための命令です。「sample1.jpg」ファイルを読み込み，配列「Stimulus_matrix」として保存しています。
「Stimulus_matrix」の中に何が入っているでしょう？

画像サイズの取得

配列の行数　配列の列数　色チャンネル　　画像ファイルの配列

```
26 [y_matrix, x_matrix, z_matrix] = size(Stimulus_matrix);
```

読み込んだ画像ファイルの縦のサイズ，横のサイズ，色チャンネル数を取得します。「位置指定」で使います。

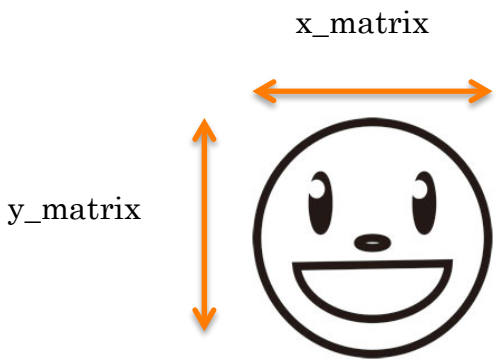
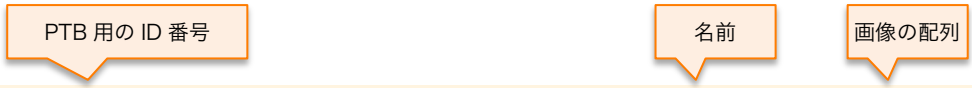


図 2- 8 画像のサイズ

PTB に画像を渡す。テクスチャ化　(PTB 関数)




```
28 Stimulus_texture = Screen('MakeTexture', wp, Stimulus_matrix);
```

画像ファイルの配列を PTB 用の「テクスチャ」として保存します。
「Stimulus_texture」に呼び出し用の「ID 番号」が渡されます。③画像を画面に提示する際は,「DrawTexture」関数とともにこの ID 番号が必要になります。

2.4.3. 刺激の提示位置の指定

(PTB 特有) (Presentation.m Line 10-38)

刺激位置の取得は PTB の大きな山場です。

左ページ

```
10  %========Open window and get the window size=======
11  screenNumbers=Screen('Screens');
12  [wp,Screen_rect] = Screen('OpenWindow',max(screenNumbers));
13  xcenter = Screen_rect(3)/2;
14  ycenter = Screen_rect(4)/2;
15  %==================================================
16
17  %=========Variables===============
18  PresentationDuration = 1;
19  Number_of_Presentation = 5;
20  Position_of_Stimulus = [0 0]; % [x y]
21  %=================================
22
23  %========Prepare the image============
24  Stimulus_matrix = imread('sample1.jpg');
25
26  [y_matrix, x_matrix, z_matrix] = size(Stimulus_matrix);
27
28  Stimulus_texture = Screen('MakeTexture',wp,Stimulus_matrix);
29  %=================================
30
31  %=======Calculate the presentation position=================
32  Center_rect = [xcenter ycenter xcenter ycenter];
33  Position_rect = [Position_of_Stimulus(1) Position_of_Stimulus(2) ...
34                   Position_of_Stimulus(1) Position_of_Stimulus(2)];
35  Stimulus_rect = [-x_matrix/2, -y_matrix/2, x_matrix/2, y_matrix/2];
36
37  Stimulus_position_rect = Center_rect + Position_rect + Stimulus_rect;
38  %==================================================
```

画面の中心

刺激の中心の位置

刺激の提示範囲

全部足す

左ページ

```
10  %========提示するモニタの準備をします=======
11  11% 初期化１
12  12% 初期化２
13  13-14  % 画面の中心座標の取得
14
15
16
17  %========いくつかの変数をいじれます==========
18  18 % 提示時間を指定
19  19 % 提示の回数を指定
20  20 % 刺激の提示位置
21
22
23  %=======提示する画像の準備をします========
24  24 % imread は Matlab の関数で、画像を読み込んで matlab で扱える行列に変換します。
25
26  26 % 行列のサイズはそのまま、画像のサイズ(ピクセル)になります。
27
28  28 % PTBで提示できるように、Screen関数のMakeTextureを使って描画する準備をします。
29
30
31  %==== 刺激の提示位置の計算================
32  %刺激の提示位置を取得します。
33  32 % ① 画面の中心座標，左右の四角形の中心座標をそれぞれ取得します。
34          x 座標と y 座標を２回ずつ取得しているのは「Stimulus_rect」と合わせるためです。
35  33 % ② 刺激の中心位置を x 座標と y 座標を２回ずつ取得します。
36  35 % ③ 刺激の左上と右下の x 座標と y 座標を取得します。
37  37 % ①+②+③で位置を計算
```

画面の中心

刺激の中心の位置

刺激の提示範囲

全部足す

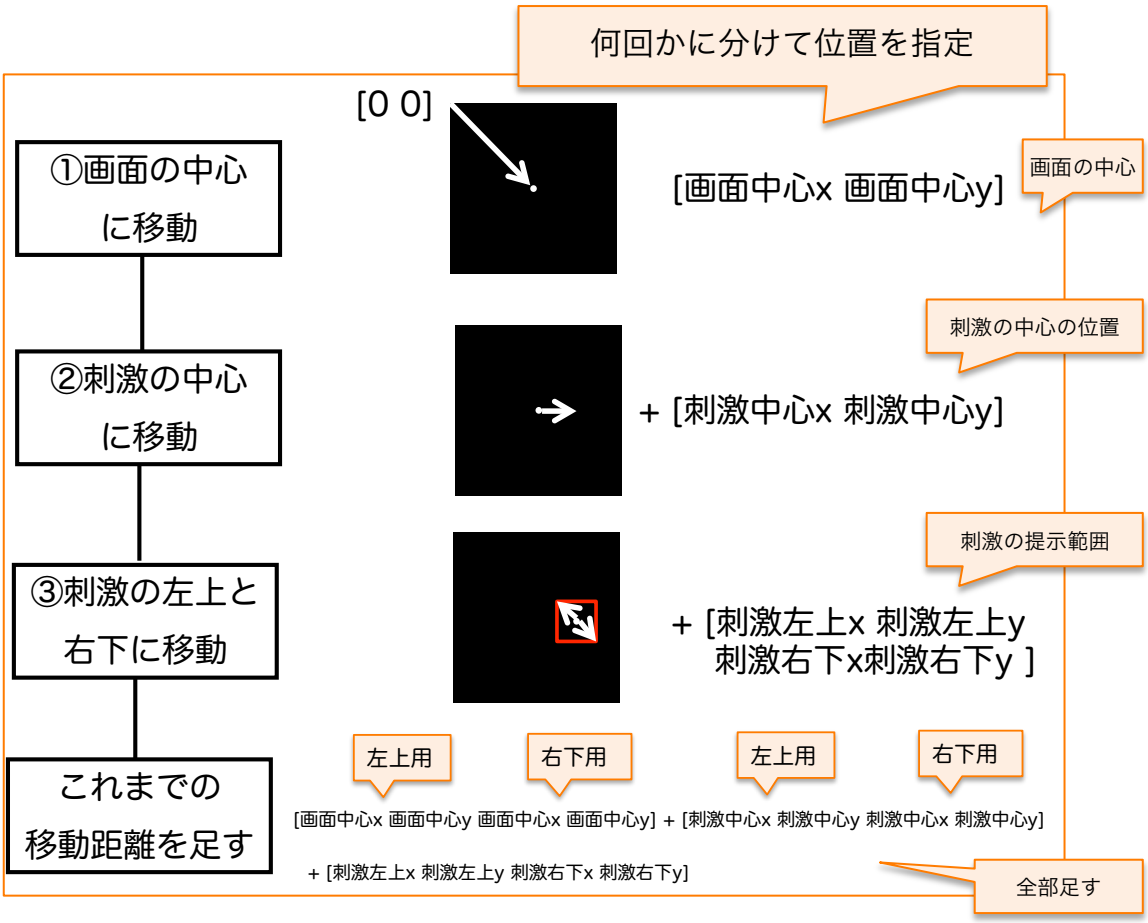
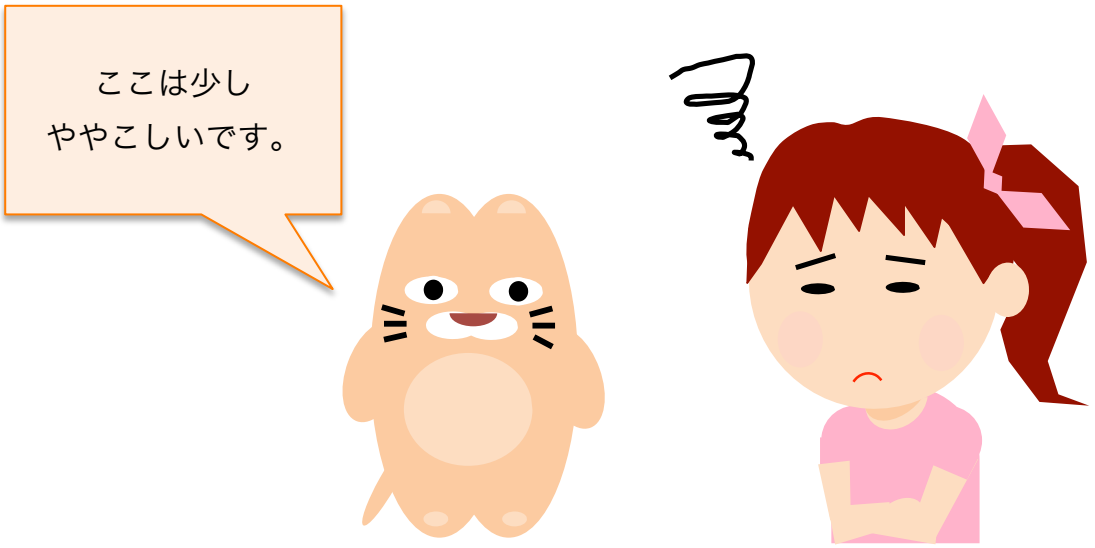


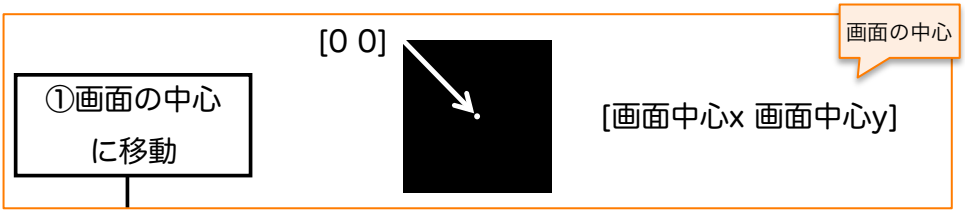
図 2- 9 刺激の提示位置の指定



① 左上から画面の中心に移動 (Line 13-14)

「xcenter」，「ycenter」 として既に取得済

```
10    %========Open window and get the window size=======
11    screenNumbers=Screen('Screens');
12    [wp,Screen_rect] = Screen('OpenWindow',max(screenNumbers));
13    xcenter = Screen_rect(3)/2;
14    ycenter = Screen_rect(4)/2;
15    %=========================================
```



② 刺激の中心に移動 (Line 20)

[0 0]が入力されています。好きな数字を入れてみてください。
刺激の提示位置が変わります。

```
17    %==========Variables==============
18    PresentationDuration = 1;
19    Number_of_Presentation = 5;
20    Position_of_Stimulus = [0 0]; % [x y]
21    %=========================================
```

③ 刺激の左上と右下に移動（Line 26, 35）

刺激の大きさは「y_matrix」，「x_matrix」 として既に取得済

```
23   %========Prepare the image============
24   Stimulus_matrix = imread('sample1.jpg');
25
26   [y_matrix, x_matrix, z_matrix] = size(Stimulus_matrix);
27
28   Stimulus_texture = Screen('MakeTexture',wp,Stimulus_matrix);
29   %======================================
```

刺激の大きさを取得

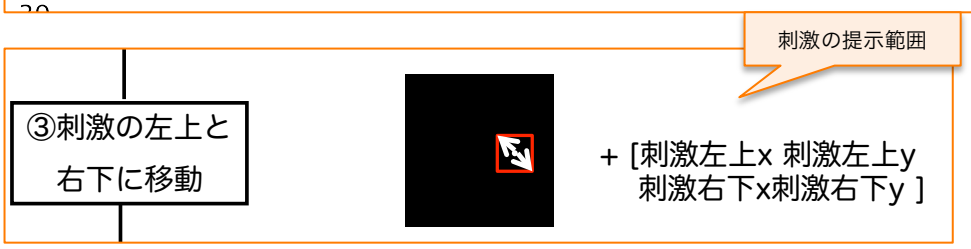
刺激の大きさ情報を基に刺激の**左上と右下の座標「Stimulus_rect」**を計算します。中心の座標を「[0 0]」とすると，左上の座標は刺激サイズの半分だけ，左側，上側（マイナス方向）に移動した座標になります。

[-x_matrix/2 -y_matrix/2]

また，右下の座標は刺激サイズの半分だけ，右側，または下側（プラス方向）に移動した座標になります。

[x_matrix/2 y_matrix/2]

```
31   %========Calculate the presentation position==================
32   Center_rect = [xcenter ycenter xcenter ycenter];
33   Position_rect = [Position_of_Stimulus(1) Position_of_Stimulus(2) ...
34                    Position_of_Stimulus(1) Position_of_Stimulus(2)];
35   Stimulus_rect = [-x_matrix/2, -y_matrix/2, x_matrix/2, y_matrix/2];
36
37   Stimulus_position_rect = Center_rect + Position_rect + Stimulus_rect;
38   %=============================================================
```



実際の座標「①＋②＋③」を取得する。(Line 31-38)

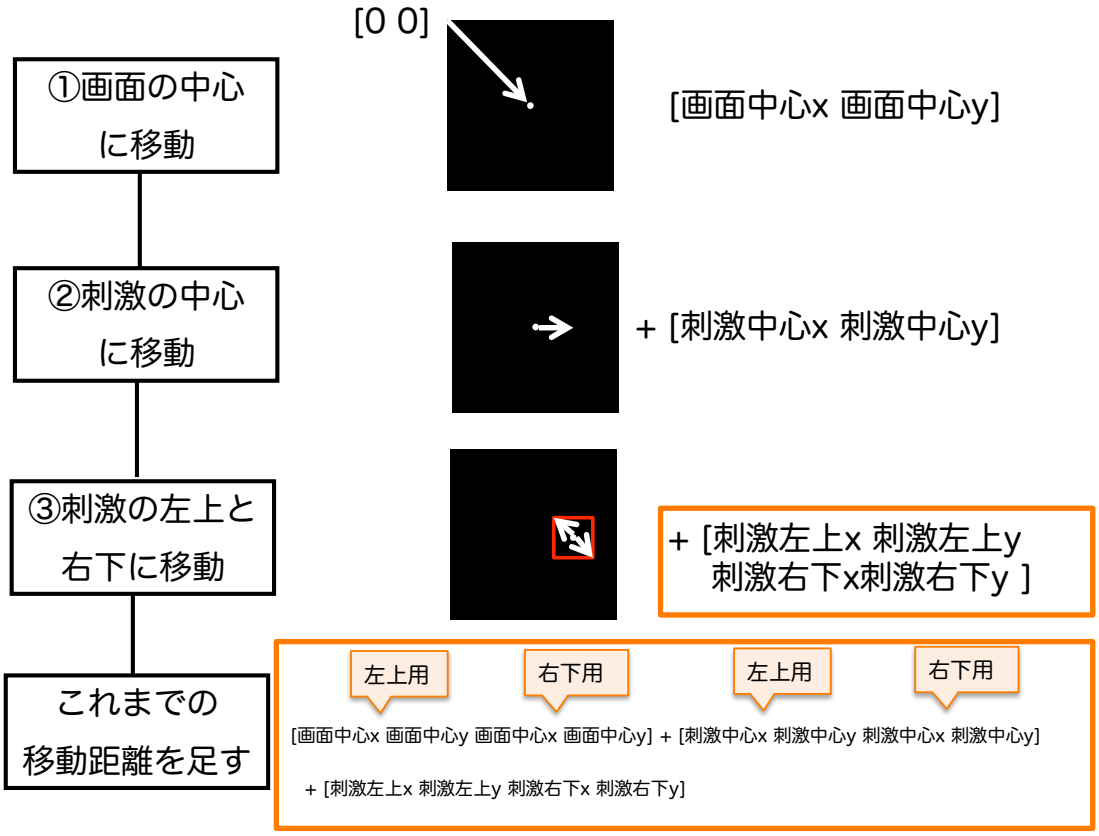
ここまで，①画面の中心の座標，②各刺激の中心の座標，③刺激の左上と右下の座標を取得しました。これらを足すと実際の画像の提示範囲を指定出来ます。①「Center_rect」と②「Position_rect」は左上用と右下用に２回ずつ xy 座標を代入します。

①画面の中心，②刺激の中心

```
31  %========Calculate the presentation position=================
32  Center_rect = [xcenter ycenter xcenter ycenter];
33  Position_rect = [Position_of_Stimulus(1) Position_of_Stimulus(2) ...
34                   Position_of_Stimulus(1) Position_of_Stimulus(2)];
35  Stimulus_rect = [-x_matrix/2, -y_matrix/2, x_matrix/2, y_matrix/2];
36
37  Stimulus_position_rect = Center_rect + Position_rect + Stimulus_rect;
38  %-----------------------------------------------------------=====
```

①＋②＋③

①＋②＋③で，各四角形の左上の座標と右下の座標を取得しています。

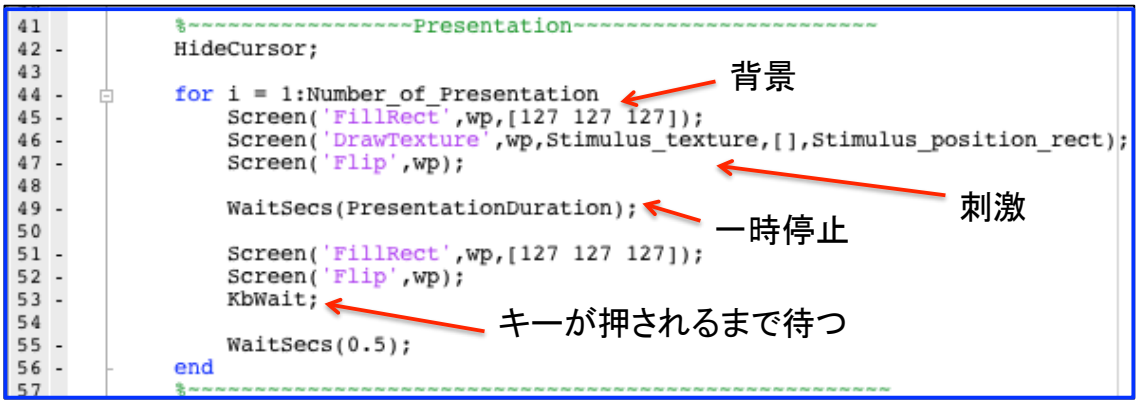


2.4.4. 画面提示(flip)

(Presentation.m Line 44-56)

画面上に刺激を描画していくプロセスについて説明をしていきます。PTB で画像を提示するための「 Screen('DrawTexture') 」と「 Screen('Flip') 」の命令が登場します。

左のページ




```
41      %~~~~~~~~~~~~~~~~~Presentation~~~~~~~~~~~~~~~~~~~~~~
42 -    HideCursor;
43
44 -    for i = 1:Number_of_Presentation
45 -        Screen('FillRect',wp,[127 127 127]);
46 -        Screen('DrawTexture',wp,Stimulus_texture,[],Stimulus_position_rect);
47 -        Screen('Flip',wp);
48
49 -        WaitSecs(PresentationDuration);
50
51 -        Screen('FillRect',wp,[127 127 127]);
52 -        Screen('Flip',wp);
53 -        KbWait;
54
55 -        WaitSecs(0.5);
56 -    end
57      %~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

右のページ

```
45    45 % FillRect  は画面を指定した色で一面塗ります
46    46 % DrawTexture  は MakeTexture した画像を画面に描画します。
47    47 %  最後に Flip します。これをしないとモニタに提示されません。
48
49    49 %  指定した提示時間だけ待ちます。
50    51 %  もう一度画面を塗りつぶして、描画した画像を消します。
51    52 %  塗りつぶした画面をモニタに表示します。
52    53 %  ボタンが押されるまで待ちます。
53
54
55    55 % KbWait の後に次の命令の実行まで少し時間を入れます。
```

背景として，画面全体を塗りつぶす（PTB 関数）

名前　色

```
45 Screen('FillRect',wp,[127 127 127]);
```

まずは画面全体を塗りつぶすために，「'FillRect'」を使います。2 つめの引数はウィンドウポインタ，3 つめは塗りつぶす色を指定します。4 つめの引数として提示領域を指定すると指定された範囲に四角形が提示されます。今回のように 4 つめの引数を省略すると画面全体が塗りつぶされます。

画像を提示　テクスチャを描画（PTB 関数）

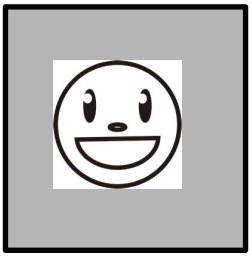
名前　テクスチャの ID 番号　提示範囲

```
46 Screen('DrawTexture',wp, Stimulus_texture, [], Stimulus_pos);
```

この関数は「'MakeTexture'」に対応する関数で，テクスチャ化した画像情報を「offscreen」に描画します。2 つめの引数はウィンドウポインタ，3 つめは描画するテクスチャの ID 番号です。4 つめの引数は刺激の提示位置と範囲の情報です。今回は省略しているのですが，５つめの引数は画像の回転角度を指定する事ができます。もし画像を「180°」回転させて表示する場合には，「180」にしてください。

画面をフリップ（PTB 関数）

```
47 Screen('Flip',wp);
```



Flip 関数が描かれるまで画面に何も提示されません。PTB では同じタイミングで画面に表示するために，「ビデオカードの仮想的なメモリーバッファ (offscreen)」にあらかじめ提示するグラフィックを書いておき，それを「別の

バッファ(onscreen)」と「置き換えること（Flip)」で画面に提示しています(double buffering)。プログラム上では「FillRect」が「DrawTexture」よりも先に書かれていますが，この２つの図形は同じタイミングで onscreen に置き換えられています。

では「FillRect」と「FillArc」を書く順番を逆にするとどうなるかというと，画像が表示されなくなります。これは画像を描画した上から画面全体を塗りつぶしているためです。

一定時間待つ（PTB 関数）

```
49 WaitSecs(PresentationDuration);
```

画面を停止するために「WaiteSecs コマンド」を使用します。この関数は指定した秒数だけ画面を静止します。１を入れると１秒間画面が停止します。（ミリ秒単位ではなく秒単位）.

キー押しまで待つ（PTB 関数）

```
53 Kbwait;
```

何かキーが押されるまで待ちます。

2.5. サイモン効果のプログラムも見てみよう。

・赤い四角：条件の初期設定
・青い四角：画面描画

サイモン効果のプログラムコードも赤と青のマーカーで囲ってください。また,
各プログラムが何をしているのかメモをしてください。
どこで何をしているかなんとなくつかめてきましたか？

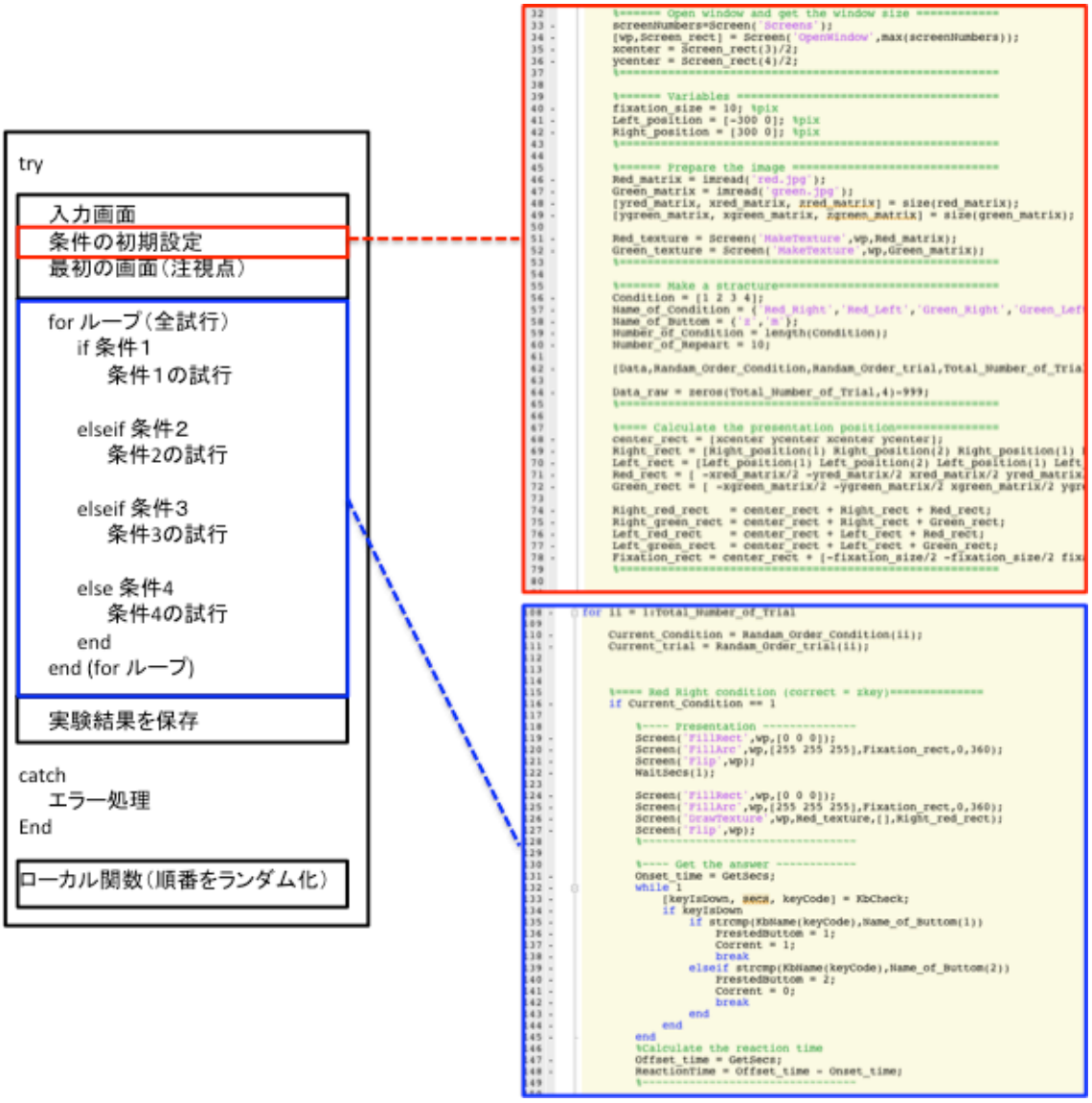


図 2- 10 サイモン効果のプログラム

2.5.1. サイモン効果の簡易版「Presentation2」

反応時間を計測するプログラムが追加されています。追加されている部分にチェックを入れてください。


```
function Presentation2
%
%2015/07/13 Hisakata made

AssertOpenGL;

try
    input('Please press Enter.');
    %========Open window and get the window size=======
    screenNumbers=Screen('Screens');
    [wp,Screen_rect] = Screen('OpenWindow',max(screenNumbers));
    xcenter = Screen_rect(3)/2;
    ycenter = Screen_rect(4)/2;
    %================================================

    %=========Variables============
    PresentationDuration = 1;
    Number_of_Presentation = 5;
    Position_of_Stimulus = [0 0]; % [x y]
    Name_of_Buttom = {'z','m'};
    %==============================

    %========Prepare the image============
    Stimulus_matrix = imread('sample1.jpg');

    [y_matrix, x_matrix, z_matrix] = size(Stimulus_matrix);

    Stimulus_texture = Screen('MakeTexture',wp,Stimulus_matrix);
    %=====================================

    %=======Calculate the presentation position=================
    Center_rect = [xcenter ycenter xcenter ycenter];
    Position_rect = [Position_of_Stimulus(1) Position_of_Stimulus(2) ...
                     Position_of_Stimulus(1) Position_of_Stimulus(2)];
    Stimulus_rect = [-x_matrix/2, -y_matrix/2, x_matrix/2, y_matrix/2];

    Stimulus_position_rect = Center_rect + Position_rect + Stimulus_rect;
    %===========================================================

    %~~~~~~~~~~~~~~~~~Presentation~~~~~~~~~~~~~~~~~~~~~~
    HideCursor;

    for i = 1:Number_of_Presentation
        Screen('FillRect',wp,[127 127 127]);
        Screen('DrawTexture',wp,Stimulus_texture,[],Stimulus_position_rect);
        Screen('Flip',wp);

        WaitSecs(PresentationDuration);

        Screen('FillRect',wp,[127 127 127]);
        Screen('Flip',wp);
        %~~~~ Get the answer ~~~~~~~~~~~~
        Onset_time = GetSecs;
        while 1
            [keyIsDown, secs, keyCode] = KbCheck;
            if keyIsDown
                if strcmp(KbName(keyCode),Name_of_Buttom(1))
                    PrestedButtom = 1
                    break
                elseif strcmp(KbName(keyCode),Name_of_Buttom(2))
                    PrestedButtom = 2
                    break
                end
            end
        end
        %Calculate the reaction time
        Offset_time = GetSecs;
        ReactionTime = Offset_time - Onset_time
        %~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

        WaitSecs(0.5);
    end
    %=============================================================


    %=====Close the window====================
    ShowCursor;
    Screen('CloseAll');
    %=========================================

catch ME
    ShowCursor;
    Screen('CloseAll');
    rethrow(ME);
end;
```

図 2- 11 差分はどこ？

2.5.2. 反応キーの取得

(Presentation2.m Line 21, 56-67)

「z」か「m」のキーが押されるまで画面を停止させます。

まずは，条件設定で反応キーを決めます。

左のページ

```
21    Name_of_Buttom = {'z','m'};
```

右のページ

```
21    21 % キー入力{z キーと m キー }
```

反応の取得は以下のように「while ループ」と「if 文」を組み合わせて行います。

左のページ

```
56      while 1
57          [keyIsDown, secs, keyCode] = KbCheck;
58          if keyIsDown
59              if strcmp(KbName(keyCode),Name_of_Buttom(1))
60                  PrestedButtom = 1
61                  break
```

```
62              elseif strcmp(KbName(keyCode),Name_of_Buttom(2))
63                  PrestedButtom = 2
64                  break
65              end
66          end
67      end
```

右のページ

```
56    Line 56-67   % z か m キーを押すまで while ループ
57    57 % 反応キーを取得
58    58 % キーは押された？
59    59 % strcmp 関数で押されたキーが z キーか他のキーかを判別しています。
60    60 % PrestedButtom = 1 zキーか
61    61 % ループから抜ける
```

```
63    62 %strcmp 関数で押されたキーが m キーか他のキーかを判別しています。
64    63 % PrestedButtom = 2 mキーか
65    64 % ループから抜ける
```

while ループを使い，反応キーが押されるまで同じ処理を繰り返します。

while ループで反応取得

```
56  while 1
                繰り返す内容
67 end
```

「while ループ」は「while」の右側の条件が満たされるまで「end」までに書かれている内容を繰り返し実行するというものです。通常は「while (a > 2)」といったように特定の条件が達成された場合にループを抜けるように書きます。今回のように「while 1」と書くと無限にループが続きます。この無限に続くループを抜け出すためには「**break**」という関数が必要です。「while ループ」の中に「if 文」を書き，特定の条件が満たされた場合にのみ「break」するようにします。今回の場合は，特定のキーが押された場合に「break」が実行されるようにします。

反応キーを取得（PTB 関数）

```
57 [keyIsDown, secs, keyCode] = KbCheck;
```

反応キーのチェックは KbCheck で行います。この関数では，キーボードのキーが押された際に「keyIsDown：押された(1)・押されていない(0)の状態」，「secs：押された時刻」，「keyCode：何のキーが押されたか（キーコード）」を取得します。「KeyCode」はキーボードの番号が取得されますが，その番号は「mac」と「windows」ではどのキーに対応しているかが異なっています。「KeyCode」がどのキーに対応しているかは「KbName(keyCode)」で取得する事が出来ます。

反応キーの取得後，特定のキーが押されたかを判別。

```
58  if keyIsDown
59      if strcmp(KbName(keyCode),Name_of_Buttom(1))
61              break
62      elseif strcmp(KbName(keyCode),Name_of_Buttom(2))
64          break
65      end
66 end
```

反応キーが押された状態になった場合に，「keyIsDown」以降の処理が行われます。この処理は「if文」でさらに２つに分かれています。「**strcmp**」はMATLABの関数で，１つ目の「KbName(keyCode)」と２つ目の「Name_of_Buttom(1)」を比較する関数です。この２つが同じ場合に以下の処理「break」が実行されます。「Name_of_Buttom(1)」は上述の「初期設定」の部分で指定したように，「zキー」が保存されています。同様に「Name_of_Buttom(2)」は「mキー」が保存されています。

今回は「Name_of_Buttom」が「１」の場合，つまり「zキー」が押された場合には，「PrestedButtom」に１が入り，「Name_of_Buttom」が「２」の場合，つまり「mキー」が押された場合には，「PrestedButtom」に２が入るようになっています。

2.5.3. 反応時間の測定
(Presentation2.m Line 55, 59-70)

刺激提示から反応キーが押されるまでの反応時間を測定します。「時間を測定し始めた時刻」と「時間を測定し終えた時刻」の情報が必要になります。「GetSecs 関数」を用いて刺激提示直後と反応キーが押された直後の時刻を測定します。そして，２回目の時刻から１回目の時刻を引き算し，反応時間を測定します。反応時間を測定するコマンドは while 文と組み合わせて書きます。

「while ループ」の直前に「GetSecs関数」で時刻を取得し，「Onset_time」に保存します。ループから抜けた直後にもう一度「GetSecs関数」で時刻を取得し，「Offset_time」に保存します。反応時間は「Offset_time」から「Onset_time」を引くことで算出できます。反応時間は変数「ReactionTime」として保存しています。

```
55      Onset_time = GetSecs;
56              while 1
                        キーの取得
67              end
68      Offset_time = GetSecs;
69      ReactionTime = Offset_time - Onset_time;
```

2.6. 条件の初期設定をいろいろ変えてみよう

それでは，サイモン効果のプログラムコードに，メモをしていってください。
今度は「Simon_effect」の「条件の初期設定」をいろいろ変えてみましょう。

繰り返し回数を１０回から１回に変えてみよう。(Line 60)

繰り返し回数を「１回」に変えてみましょう。各条件が１試行ずつになるため「２要因 × ２水準 × １回で，全部で４試行」となります。

```
56    %====== Make a stracture=====================================
57    Condition = [1 2 3 4];
58    Name_of_Condition = {'Red_Right','Red_Left','Green_Right','Green_Left'};
59    Number_of_Condition = length(Condition);
60    Number_of_Repeart = 10;
```

１回に変更

四角形の提示位置を変えてみよう。(Line 41-42)

四角形の位置を色々と変えてください。

```
39    %====== Variables ============================================
40    fixation_size = 10; %pix
41    Left_position = [-300 0]; %pix
42    Right_position = [300 0]; %pix
43    Name_of_Buttom = {'z','m'};
44    %=============================================================
```

[300 0] から [150 0]
[-300 0] から [-150 0]

反応キーを変えてみよう。 (Line 43)

```
39    %====== Variables ============================================
40    fixation_size = 10; %pix
41    Left_position = [-300 0]; %pix
42    Right_position = [300 0]; %pix
43    Name_of_Buttom = {'z','m'};
44    %=============================================================
```

「{ ‘z’ , ‘m’ }」から「{ ‘f’ , ‘j’ }」

反応キーを「f キー」と「j キー」に変えてみましょう。「Name_of_Buttom」を「{ 'f' , 'j' }」に変えてください。「z キー」と「m キー」を押しても反応しなくなります。「f キー」と「j キー」を押すと実験が進みます。

2.7. サイモン効果のプログラムの解説

サイモン効果のプログラムとほぼ同じの「Presentation3」デモを見てみましょう。複雑そうな部分も，これまでの組み合わせで出来ていることが分かると思います。また，「Simon_Effect」のプログラムと見比べて何が違うのかもチェックしてみてください。

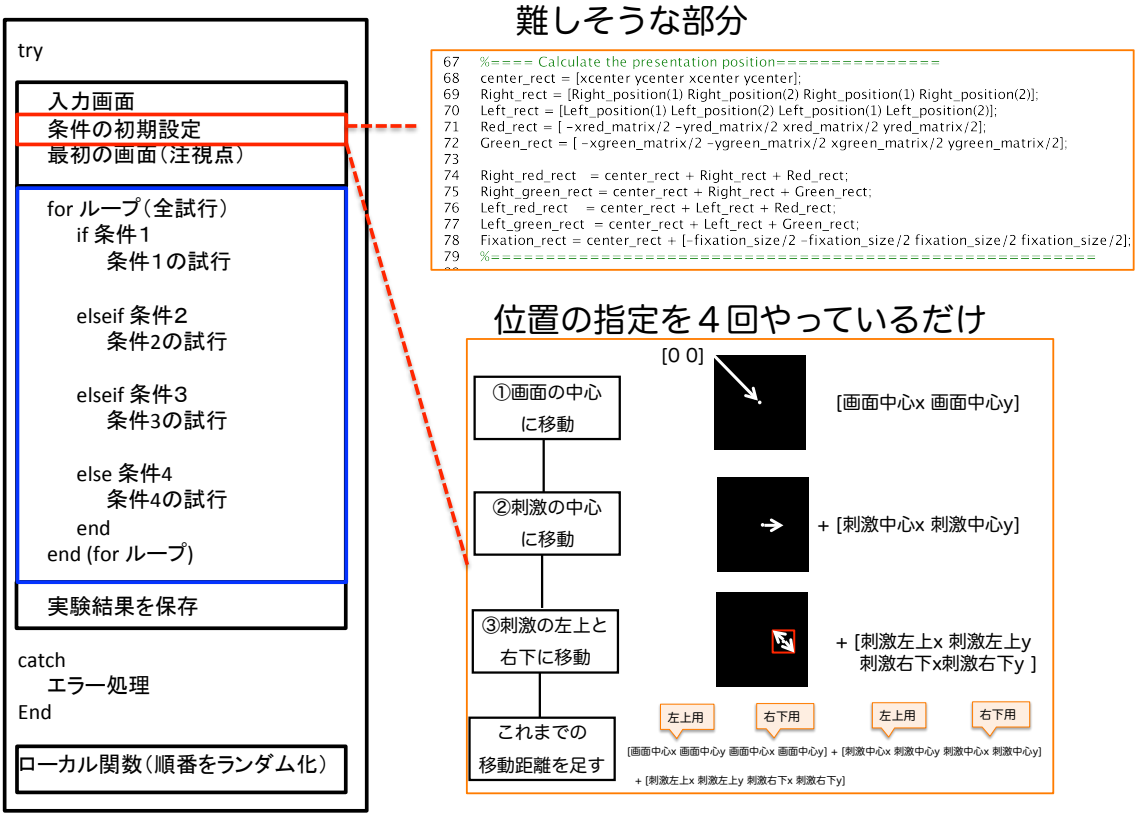


図 2- 12 複雑に見えてもこれまでの繰り返し①

2.7.1. 各試行の刺激描画

以降は各試行における刺激描画について説明していきます。「for ループ」を使い，決められた試行数だけ同じ処理を繰り返していきます。また，試行ごとに異なった条件の刺激が提示されるように「if 文」で条件分岐をして刺激を提示していきます。本実験では全部で４つの条件がありました。各条件の if 文以降に各条件の刺激描画を実行します。例えば，「if 条件１」の場合には，「条件１の試行」が実行されます。

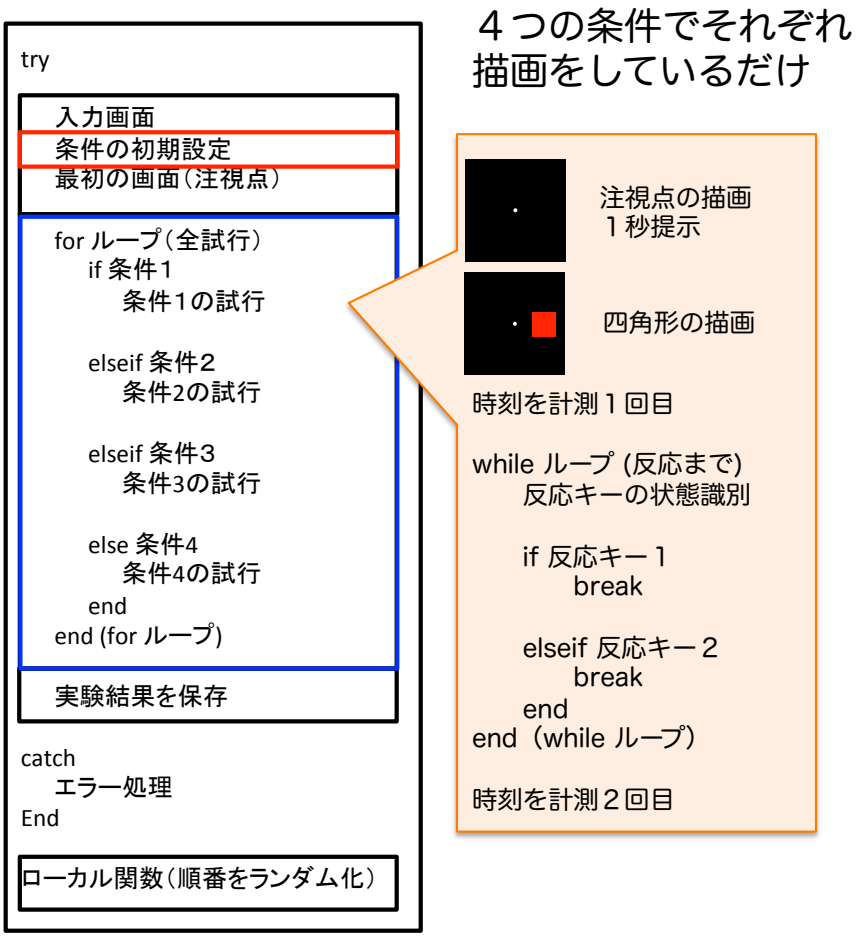


図 2- 13 複雑に見えてもこれまでの繰り返し②

2.7.2. 条件の設定

(Simon_Effect.m Line 56-65)

以下のプログラムによって刺激をランダムな順番で提示しています。
（この部分については第３章で）

左のページ

```
56    %====== Make a stracture=================================
57    Condition = [1 2 3 4];
58    Name_of_Condition = {'Red_Right','Red_Left','Green_Right','Green_Left'};
59    Number_of_Condition = length(Condition);
60    Number_of_Repeart = 10;
```

```
62    [Data,Randam_Order_Condition,Randam_Order_trial,Total_Number_of_Trial] =↙
MakeDataFile_ReactionTime(Number_of_Repeart,Number_of_Condition,Name_of_Condition);
63
64    Data_raw = zeros(Total_Number_of_Trial,4)-999;
65    %=========================================================
```

右のページ

```
55    %====== 構造体を作る ================================
56    %条件は番号と名前の両方で指定します。
57    %構造体には、各従属変数と被験者の回答を入れる Answer Sheet が含まれます
58    57 Condition...%自分で設定する条件の数を入れます
59    58 Name_of_Condition...%各条件名を文字列で入れておきます
60    59 Number_of_Condition...%条件数です。
```

```
64    Line 62, 294-313
65    %ローカル関数でデータを入れる構造体を作り、提示順序をランダムにします。
66    62 [Data,Randam_Order_Condition,Randam_Order_trial,Total_Number_of_Trial] =
67    MakeDataFile_ReactionTime(Number_of_Repeart,Number_of_Condition,Name_of_Condition);
```

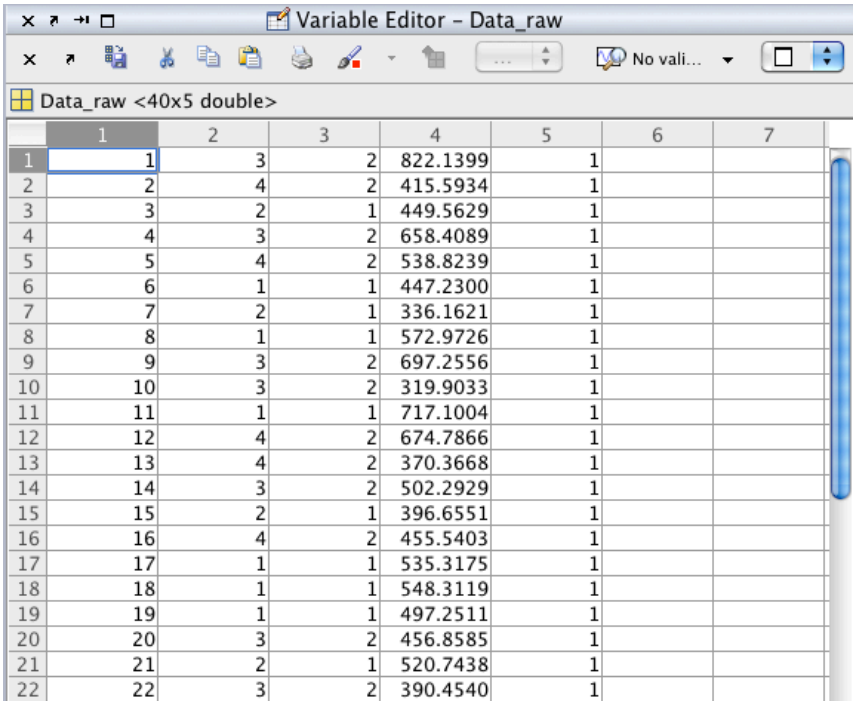
2.3.9. 実験結果の保存

(Simon_Effect.m Line 254-265)

試行順序を保ったままデータを保存

```
254    %~1) Put the answer into the matrix
255    Data_raw(ii,1) = ii;
256    Data_raw(ii,2) = Current_Condition;
257    Data_raw(ii,3) = PrestedButtom;
258    Data_raw(ii,4) = ReactionTime * 1000;
259    Data_raw(ii,5) = Corrent;
```

各試行の実験結果を配列として保存します。配列「Data_raw」の１列目には「試行数」，２番目には「条件」，３番目には「押したキーの情報」，４番目は「反応時間」，５番目は「正答・誤答」を保存しています。「ii」は，試行数の情報が入っています。各試行で「ii」の値が更新されるため，各試行の情報は，表の下側に追加されていきます。出来上がった「Data_raw」は以エクセルで分析出来ます。



	1	2	3	4	5	6	7
1	1	3	2	822.1399	1		
2	2	4	2	415.5934	1		
3	3	2	1	449.5629	1		
4	4	3	2	658.4089	1		
5	5	4	2	538.8239	1		
6	6	1	1	447.2300	1		
7	7	2	1	336.1621	1		
8	8	1	1	572.9726	1		
9	9	3	2	697.2556	1		
10	10	3	2	319.9033	1		
11	11	1	1	717.1004	1		
12	12	4	2	674.7866	1		
13	13	4	2	370.3668	1		
14	14	3	2	502.2929	1		
15	15	2	1	396.6551	1		
16	16	4	2	455.5403	1		
17	17	1	1	535.3175	1		
18	18	1	1	548.3119	1		
19	19	1	1	497.2511	1		
20	20	3	2	456.8585	1		
21	21	2	1	520.7438	1		
22	22	3	2	390.4540	1		

図 2- 14 実験結果の出力

2.7.3. 実験結果の出力

(Simon_Effect.m Line 271-282)

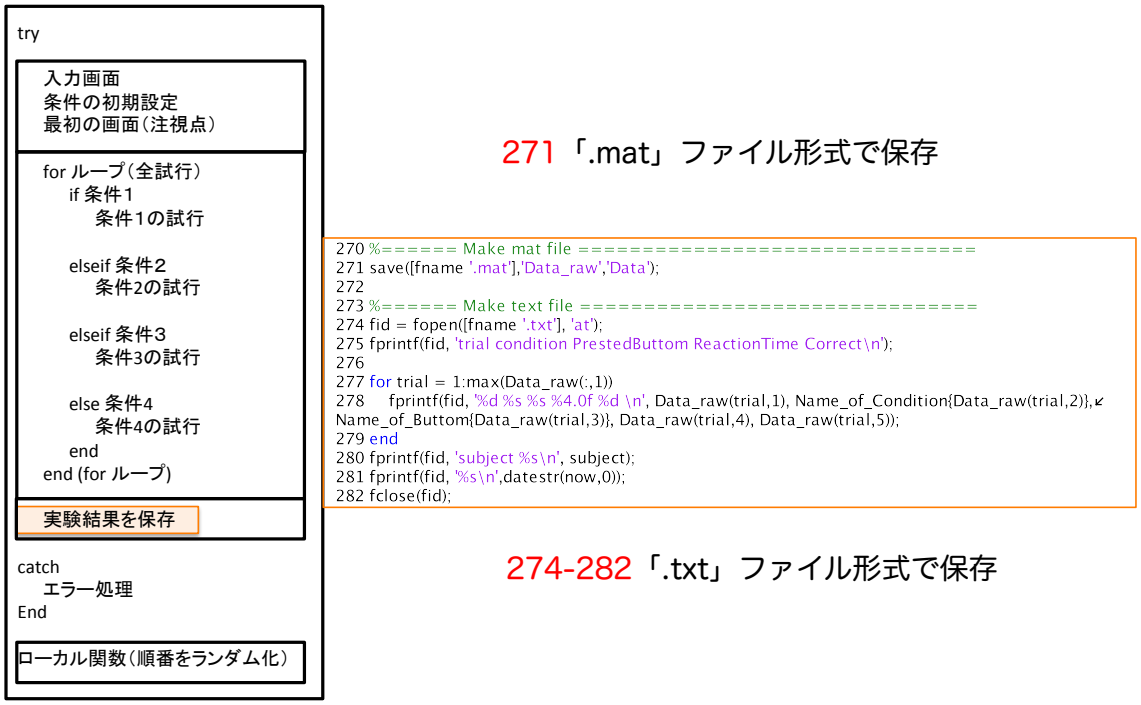


図 2- 15 実験結果の出力

「全試行の For ループ」を抜けた後に，これまでのデータをまとめて「mat ファイル」，「txt ファイル」として保存します。ファイルは必ず名前をつけて保存します。この 1 行を忘れるとこの実験が無駄になりますので忘れないようにしましょう。

プログラムの冒頭でファイル名を取得

```
9    subject = input('Subject Name?','s');
```

ファイルの名前はプログラムの最初の「fname」の部分で取得しています。

MAT file でデータを保存 (Simon_Effect.m Line 271)

```
270 %====== Make mat file ================================
271 save([fname '.mat'],'Data_raw','Data');
```

「mat ファイル」として保存する場合は，「save」関数で保存する変数を指定するだけです。「save 関数」の最初の引数は「ファイル名」，２つ目以降は保存する変数が入ります。「’’」を忘れないようにしましょう。

テキスト file でデータを保存 (Simon_Effect.m Line 273-282)

```
273 %====== Make text file ================================
274 fid = fopen([fname '.txt'], 'at');
275 fprintf(fid, 'trial condition PrestedButtom ReactionTime Correct\n');
276
277 for trial = 1:max(Data_raw(:,1))
278     fprintf(fid, '%d %s %s %4.0f %d \n', Data_raw(trial,1), Name_of_Condition{Data_raw(trial,2)},↙
Name_of_Buttom{Data_raw(trial,3)}, Data_raw(trial,4), Data_raw(trial,5));
279 end
280 fprintf(fid, 'subject %s\n', subject);
281 fprintf(fid, '%s\n',datestr(now,0));
282 fclose(fid);
```

テキストファイルの保存方法です。まずは「fid = fopen()」で保存するテキストファイルを開きます。その後，「fprintf 関数」でファイルに情報を書き込んでいきます。

第 3 章　動画を呈示してみよう

3.1 画像呈示の基本

動きのある刺激を呈示するには、基本的に少しずつ違う静止画を連続して呈示します。これはパラパラ漫画やアニメーションと同じ原理です。辞典の空白部分に絵を何枚も書いてパラパラ漫画を作ったことがあるでしょうか？原理的にあれと同じことをモニター上で行います。

一般的に、CRT モニターや液晶モニターは 1 秒間に何十回も画面をリフレッシュしています。これは**リフレッシュレート**と言います。たとえば、一般的な液晶モニターは大体 60Hz のリフレッシュレートを持ちますが、これは「1 秒間に 60 回画面を書き換えている」という意味です。動画を呈示する時には、モニターのリフレッシュレートがいくつなのか？が重要な情報となります。

PTB では、使用するモニターのリフレッシュレートがいくつなのかを簡単に調べる関数があります。

```
FRAME = Screen('FrameRate',wp);
```

この frame という変数にリフレッシュレートが何 Hz なのかという情報が入ります。リフレッシュレートは使用している OS のディスプレイ設定でも確認することができ、変更する時は PTB を通してではなく OS のディスプレイ設定で解像度と共に変更します。

★　***ちょっとポイント***

注意しなければならないのは、基本的に解像度とリフレッシュレートはトレ

ードオフの関係になるということです。つまり、解像度を高くするとリフレッシュレートは下がり、リフレッシュレートを上げると解像度は低くなります。モニターによって使用できる解像度とリフレッシュレートの組み合わせは違いますので、ディスプレイ設定で必ず確認してください。

また、モニターを複数人で共有して使用する時は、それぞれの実験で使用する解像度とリフレッシュレートが異なることがあります。その場合は、前の人が使った設定が基本的に残っていますので、確認しないまま実験を実行すると想定していない解像度とリフレッシュレートで実験をしてしまい、後で気づいて実験やり直しという事態になることがあります。

実験プログラムを作る際には、初めにモニターの解像度とリフレッシュレートをチェックする部分を作り、指定以外の解像度とリフレッシュレートの場合にはエラーを出して実験が始まらないようにしておきましょう。

3.2 動画を呈示してみよう

デモプログラムSimpleMovie_Demo.mを実行して動画を呈示してみましょう。

運動する刺激を用いる場合、まず初めにしなければいけないことは必要な静止画を必要な枚数用意することです。どういう静止画が何枚必要なのかは運動刺激の速度などにも関係するので重要な問題ですが、まずはとりあえず運動にみえる刺激を作って動かしてみましょう。

デモプログラム内では、用意したjpg画像を行列として読み込んでおきます。これは今までの手順と全く変わりはないです。読み込む画像が動画の枚数分あるというだけです。次に **Screen('MakeTexture'…)**の関数で画像の行列をOpenGL テクスチャに変換します。この際、読み込む画像の解像度が高かったり（＝行列のサイズが大きかったり)、枚数が多すぎるとメモリー不足になりますので注意しましょう。

```
%========Prepare the image============
NumberOfMovieFrames = 16;
Stimulus_texture = zeros(1,NumberOfMovieFrames);
for i = 1:NumberOfMovieFrames
    imagefilename = sprintf('%d.png',i);

    Stimulus_matrix = imread(imagefilename);

    Stimulus_texture(i) = Screen('MakeTexture',wp,Stimulus_matrix);
end
```

動画として呈示する際は、for文やwhile文の中に **Screen('DrawTexture',…)**と**Screen('Flip',…)**を入れて一枚一枚呈示します。ここではfor文を使って画像を更新し、Number_of_Presentationで指定された回数だけ刺激が呈示されるようになっています。この書き方の場合、リフレッシュレートに従って画像が更新されます。つまり60Hzであれば1/60秒（17ミリ秒）毎に画像が更新されることになります。

```
%~~~~~~~~~~~~~~~~~Presentation~~~~~~~~~~~~~~~~~~~~~~
HideCursor;

for ii = 1:Number_of_Presentation
    Screen('FillRect',wp,[0 0 0]);

    Onset_time = GetSecs;
    for k = 1:NumberOfMovieFrames
        Screen('DrawTexture',wp,Stimulus_texture(k),stimulus_rect,stimulus_position_rect);
        Screen('Flip',wp);
        %Screen('WaitBlanking',wp);
    end
    %Calculate the presentation time
    Offset_time = GetSecs;
    RealPresentationTime(ii) = Offset_time - Onset_time;

    WaitSecs(1);

    Screen('FillRect',wp,[0 0 0]);
    Screen('Flip',wp);

end
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

★　やってみよう

デモプログラムで動画は出ましたか？今のままではかなり高速に動いていて動画としては不自然だと思います。動きの速度を遅くしてみましょう。速度を簡単に操作する方法としてはおもに、

① 画像の描画を書き換える
② 画像はそのままで、更新するフレームを長くする

という方法があります。①の方法は単純に、用意する画像の運動している部分を小さくするという方法で書き換えです。滑らかな運動を呈示できるというメリットがありますが、画像の枚数が多くなりメモリー不足に陥りやすいというデメリットがあります。②の方法は、画像が更新されるフレーム数を長くすることで、遅い速度を表現するという方法です。画像を新たに用意する必要もなく簡単に速度を操作できるメリットがありますが、更新するフレーム数が長くなりすぎると運動が滑らかにみえずカクカクになってしまいます。つまり、これだけではあまり遅い速度は表現できないデメリットがあります。

とても遅い速度の運動を表現したいときは、①の方法と②の方法を組み合わせます。動きを小さく描画し、さらに更新フレームも長くします。

ここでは、②画像はそのままで、更新フレームを長くする方法をとってみましょう。一定のフレーム数分だけ次の画面の描画を待つ関数は Screen('WaitBlanking', wp, 引数)です。先ほどまではコメントアウトされていました。コメントアウトを外してみてください。またウィンドウポインタの次の項目に引数を入れ数値を変更してみてください。

```
Onset_time = GetSecs;
for k = 1:NumberOfMovieFrames
    Screen('DrawTexture',wp,Stimulus_texture(k),stimulus_rect,stimulus_position_rect);
    Screen('Flip',wp);
    Screen('WaitBlanking',wp);
end
%Calculate the presentation time
Offset_time = GetSecs;
RealPresentationTime(ii) = Offset_time - Onset_time;
```

注意すべきは、動画を呈示する for 文の中に多くの if 分岐や条件分けを入れると処理に時間がかかり、次のフレーム呈示のタイミングまで画像の準備が間に合わなくなることがあります。これを**フレーム落ち（コマ落ち）**と言います。フレーム落ちはいろいろな原因で起こり得ます。

フレーム落ちが頻繁に起きると狙った運動速度を呈示できません。また、例えば 10 フレームに 1 度画像を更新するような運動描画方法を取った時に、その 10 フレーム目がフレーム落ちしていた場合には次の更新まで 20 フレーム分画像が更新されないことになります。この場合、運動がカクカクに見えてしまいます。フレーム落ちを避けるためにも、for 文の中はできるだけ分岐を入れず、常にシリアルに処理されるようなプログラムにしましょう。

★ ***ちょっとポイント***

動画を呈示する場合は、メモリー不足をいかに防ぐか、フレーム落ちしにくいプログラムを書くかがポイントになります。メモリー不足については、動くところだけの画像を用意することによって更新する画像のサイズを小さくする、更新フレームを長くして用意する画像を少なくするなど工夫をします。

また、ループするような運動刺激の場合は、最低限必要な 1 サイクル分の画

像だけを用意し、呈示時間の間何サイクル分も呈示を繰り返すという方法もあります。この方法については次項のガボールパッチが運動するプログラムで使用していますので参考にしてください。

実際に起こったフレーム落ちを正確に把握するすべはソフトウェア上にはなく、本当にどのフレームが落ちていたか調べるためにはモニター上にフォトダイオード等の光検出器を置いて物理的に実測するしかありません。

これができない場合、目安となる方法としては動画呈示にかかった時間を**GetSecs 関数**を用いて記録するという方法があります（デモプログラム参照）。フレーム落ちしていると for 文の実行完了にフレームが落ちた分だけ時間がかかります。for 文の前後に、実行時間を計測する部分を作り、動画呈示にかかった時間を測って理論値と比較してみてください。これは、実際の呈示時間を把握する目安にもなりますので、画像呈示する場合は入れておいた方が後で役に立ちます。

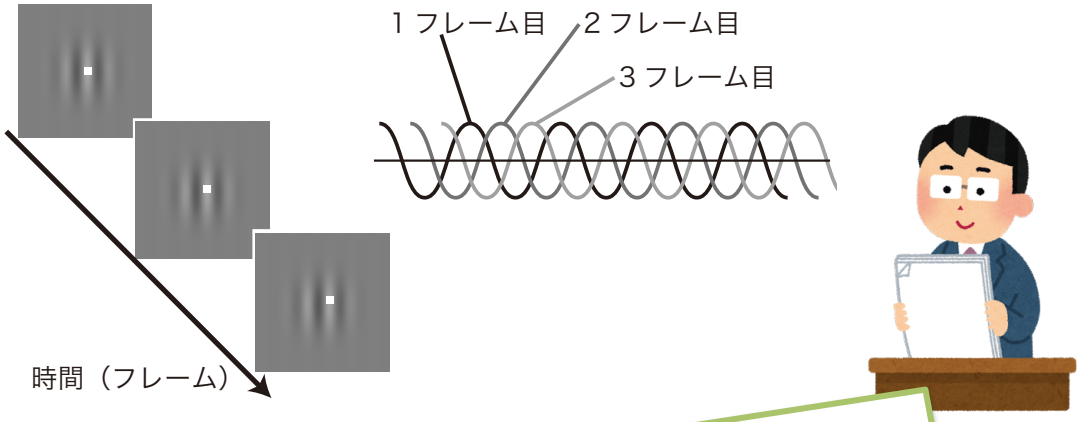
1 回の呈示で何枚もフレーム落ちする時は何か改善できる原因があります。動画を呈示している for 文内をできるだけ簡潔にする、更新する画像のサイズを小さくする等工夫をしましょう。突発的に発生する場合は、数ブロック毎に Matlab/Octave を再起動してみてください。

3.3 ガボールパッチを動かしてみよう

先ほどは用意された複数枚の絵を読み込んで運動にみせるプログラムでした。しかし実際の実験では、刺激の速度を操作しなければならず、あらかじめ計算してどういうタイミングでどの画像を出すのか決めなければいけません。ここでは速度操作のわかりやすい例として、**ガボールパッチ刺激**を用いることにします。

ガボールパッチの見た目は、ぼんやりした輪郭をもったシマシマの刺激で、視覚の働きを調べるのに適しています（詳しく知りたい方は巻末の参考文献にあたってみてください)。

ガボールパッチの本質は、中の”白と黒のシマシマ”、つまり輝度が空間的に変化する部分です。これは三角関数の正弦波（サイン波）で定義されています。ガボールパッチの中身を動かす、という時に実際に動くのは、中のサイン波の位相になります。



左の絵は、ガボールパッチの 1 フレーム目、2 フレーム目、3 フレーム目の画像を切り出したものです。縞が少しずつ右側に移動しています（わかりやすくするために、縞のあるところに白い点がついています）。
右側の図のように、1 フレーム目の中身のサイン波の位相と、2 フレーム目の中身のサイン波の位相は少しずれています。2 フレーム目のサイン波と、3 フレーム目のサイン波もずれています。

このように、輝度の波を少しずつ時間的にずらしていくと、シマシマが運動してみえるようになります。

デモプログラムSimpleGaborMotion_Demo.mを動かしてみましょう！

ガボールパッチは以下の部分で描画されています。出力 m はガボール状に輝度変調する行列になっています。

```
m = Contrast*cos(a*x+b*y+phase).* exp(-(x.^2/(2*Gabor_sigma.^2)+ y.^2/(2*Gabor_sigma.^2)));
```

何枚の画像を用意すればいいかは、ガボールパッチの空間周波数、モニターのリフレッシュレート、出したい運動速度、画像サイズ（メモリ上限に関係する）等によって決まります。

ここでは、空間周波数 1cycle/deg のガボールパッチが 1deg/s の速度で動く動画を作ります。モニターのリフレッシュレートは 60Hz だとします。観察距離は 57cm、モニターの解像度は 1024 x 768 で、モニターの横幅は 40cm でした。

観察距離が57cmの時は、1cmが1degとなりますので、27ピクセルが1degに含まれることになります。空間周波数は 1cycle/deg で呈示する速度は1deg/s ですから、1 秒間に 1cycle 分（=位相 360°）サイン波が移動すればいいことになります。つまり、動画を 1 サイクル分 60 枚で用意する場合、1 枚あたりサイン波の位相にして 6°（視角 0.017deg=0.43 ピクセル）進ませます。2 フレームに 1 回更新することにして、1 サイクル分 30 枚で用意する場合、位相にして 12°（視角 0.033deg＝0.85 ピクセル）進ませます。

運動速度が速くなれば、1 サイクルにかかる秒数も少なくなるので、用意すべき枚数も少なくなります。

★ ***注意！***

白黒 2 値の画像の場合は 1 ピクセル以下の運動は表現できません。ガボールパッチ内のサイン波は輝度が０～１（暗～明）まで連続的に変わりますので 1 ピクセル以下の運動も表現できます。運動するエッジ部分が白/黒の 2 値で描画されている場合、1 ピクセル以下の運動を表現できません。実験で運動刺激を遅い運動速度で呈示する場合、自分の刺激で動かす部分はグレースケールで連続的に変化しているかどうか確認しましょう。

もし白黒 2 値の絵で、ピクセル以下の運動を表現する場合は、1 ピクセル以下の運動を想定してエッジの位置をずらし、画像全体をぼやかす（ローパスフィルタをかける）必要があります。

★ ***注意！！***

位相の表現は**角度（0°～360°）**と**ラジアン（0～2π）**両方で表現できます。一般的に数式内ではラジアンを用います。変数を角度で指定している場合には単位の変換を忘れないようにしてください。

ここでは 1 フレームに 1 回描画を更新することにして、60 枚分の画像を用意しましょう。

```
%========Prepare the image============
Gabor_sigma = 60/60 * Onedegpix;
arraysize = 7*Onedegpix;%Gabor array size
SF = 1 ;%cpd
Velocity = 1;% deg/s
Contrast = 1;
TF = SF * Velocity; %Hz

direc = 1; %Directin of Gabor
angle = pi/2; %Radian

movieDurationSecs = 1;%seconds
movieDurationFrames = movieDurationSecs * FRAME;
numFrames = round(FRAME*(1/TF));
movieFrameIndices = mod(0:(movieDurationFrames-1),numFrames)+1;


[x,y] = meshgrid(-arraysize/2:arraysize/2,-arraysize/2:arraysize/2);
for i = 1:numFrames
    phase = direc*(i/numFrames)*2*pi;
    f = (SF/Onedegpix)*2*pi;
    a = cos(angle)*f;b = sin(angle)*f;
    m = Contrast*cos(a*x+b*y+phase).* exp(-(x.^2/(2*Gabor_sigma.^2)+ y.^2/(2*Gabor_sigma.^2)));
    m = gray + inc*m';

    Stimulus_texture(i) = Screen('MakeTexture',wp,m);
end
```

用意したガボールパッチの画像を順次呈示してみます。

```
for ii = 1:Number_of_Presentation
    Screen('FillRect',wp,[127 127 127]);

    Onset_time = GetSecs;
    for k = 1:movieDurationFrames
        Screen('DrawTexture',wp,Stimulus_texture(movieFrameIndices(k)),stimulus_rect,stimulus_position_rect);
        Screen('Flip',wp);
          Screen('WaitBlanking',wp);
    end
    %Calculate the presentation time
    Offset_time = GetSecs;
    RealPresentationTime(ii) = Offset_time - Onset_time;

    Screen('FillRect',wp,[127 127 127]);
    Screen('Flip',wp);

    WaitSecs(1);
end
```

★　やってみよう！

　動画の呈示は、プログラムを組む前に自分がどのような刺激を出したいのかをきちんと頭にイメージして、どのタイミングでどの絵を出すのか考えておかないとコントロールできません。自分の任意の空間周波数・速度にガボールパッチを呈示するにはどうしたらいいのか考えてデモプログラムをいじってみましょう。

第 4 章　心理物理学的測定法を学ぼう

4.1.心理物理学って？

心理物理学とは、Psychophysics の現代訳で外的な刺激と内的な感覚の対応関係を測定する学問です。人間がもつ感覚というのはとても主観的なもので、例えばある明るさの白い光を出した時にも、どの程度明るく感じられるかというのは人によって違います。心理物理学的測定法は、人によって異なる感覚を測定しそれを数値化します。

測定法を理解するのに重要になるのが、**閾値**と**主観的等価点**の概念です。

閾値…刺激を検出するのに必要な刺激強度のこと。刺激自体や刺激特性を検出できる閾値を**絶対閾**と言い、ある刺激と刺激の違いを検出できる閾値のことを**弁別閾**といいます。

絶対閾の例... 刺激が動いているかどうかがわかる速度=運動検出閾。刺激がみえるようになるコントラスト=コントラスト検出閾。

弁別閾の例... 2 つの運動刺激が違う速度で動いているとわかる速度差=速度弁別閾。2 つの刺激の方位が違っているとわかる角度差=方位弁別閾。

主観的等価点…ある基準点と等しく感じる刺激強度のこと。ある基準点は、実験者が標準刺激として呈示することもあれば、「止まってみえる/垂直にみえる」など内的に定義されることもある。

基本的に、評定法以外の、刺激強度が数量的・物理的に操作できることを前提にした測定法を用いる場合は、閾値か主観的等価点のどちらかを測定することになります。

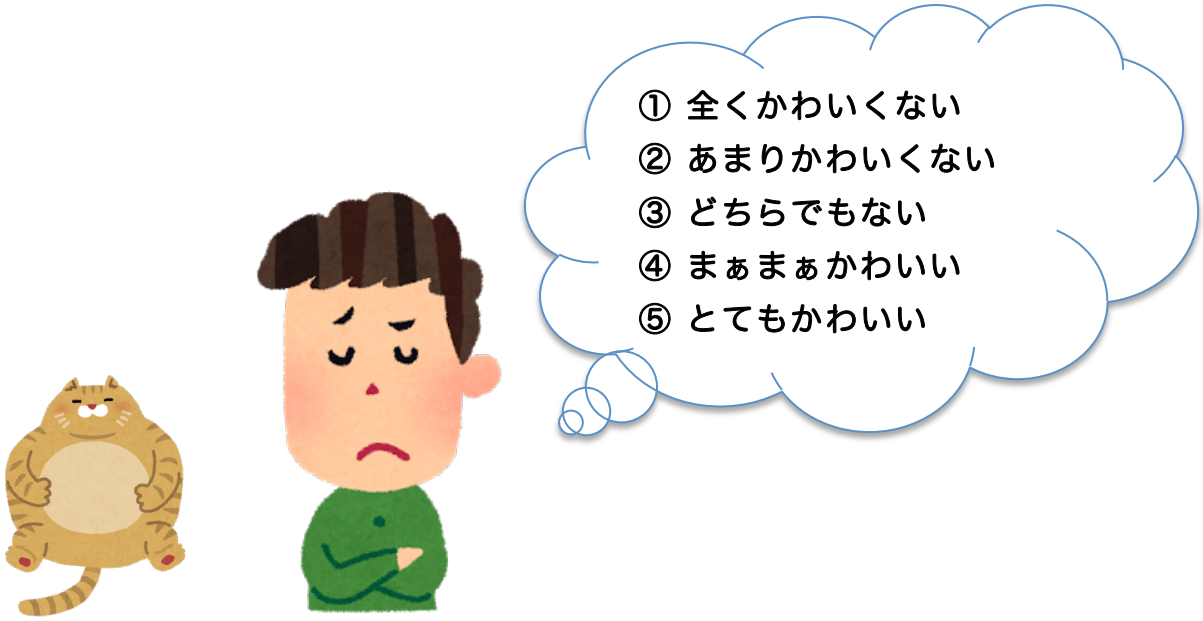
4.2.いろいろな知覚測定法

知覚を測定するにはさまざまな方法がありますが、ここでは主要なものとして**評定法、調整法、階段法（極限法）、恒常法**をとりあげます。

評定法

あらかじめ用意した評定尺度に従って、***参加者に刺激の強度を評定してもらう方法です。***

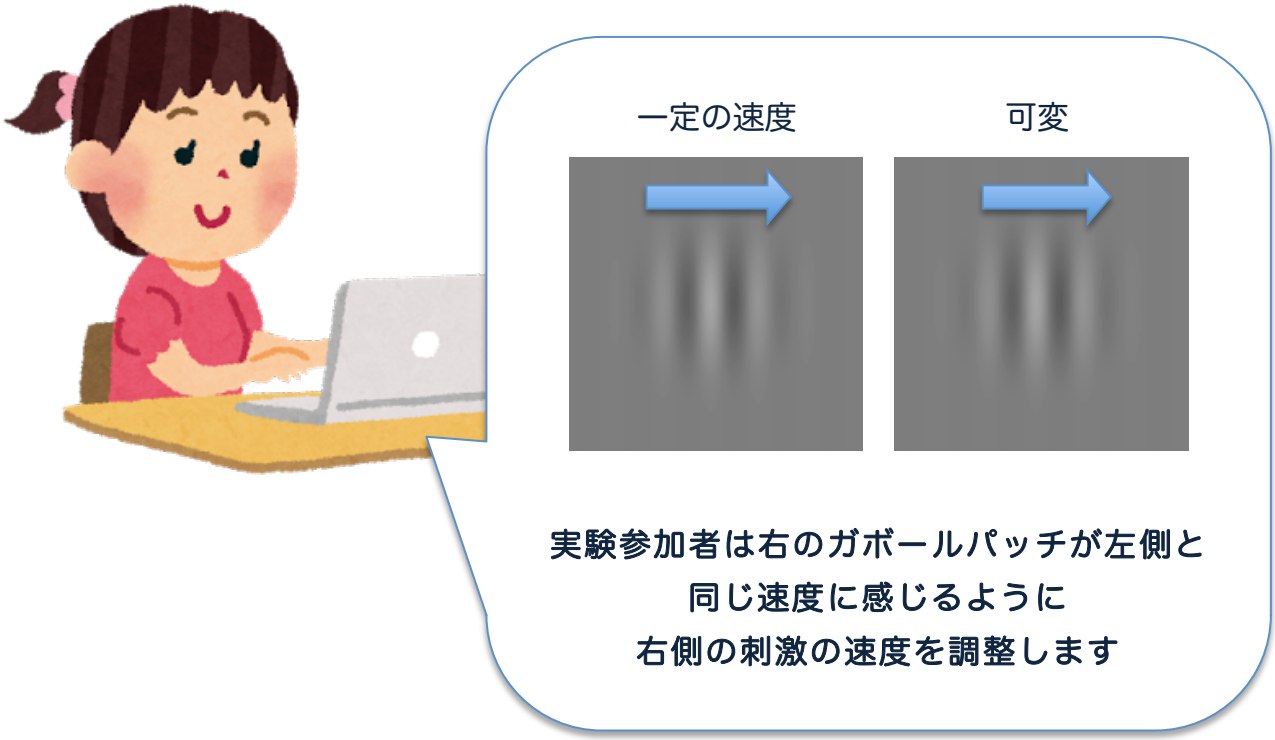
たとえば、刺激の運動印象を①全く動いていない～⑦はっきり動いている、の 7 段階で評定してもらうという手法が評定法になります。この方法は、刺激強度を数量的に操作できない場合や呈示刺激の印象などを聞きたい場合に用います。知覚測定だけでなく、さまざまな調査や実験などで用いられます。



調整法

刺激強度を数量的・物理的に操作できる場合に用います。***調整法では、参加者に刺激を直接操作させます。***

たとえば、左視野と右視野にガボールパッチを呈示します。左視野のガボールパッチは一定速度で動いており、参加者は右視野のガボールパッチの速度を自由に操作できます。参加者には、左視野のガボールパッチと右視野のガボールパッチが同じ速度になるように右視野のガボールパッチの速度を操作させます。このようにして得られた速度は“**左のガボールパッチと同速に感じる主観的等価速度**”と言えます。



このようにして、ある基準点（この場合は左視野のガボールパッチの速度）と一致するまで刺激の強度を参加者に調整させる方法を調整法といいます。

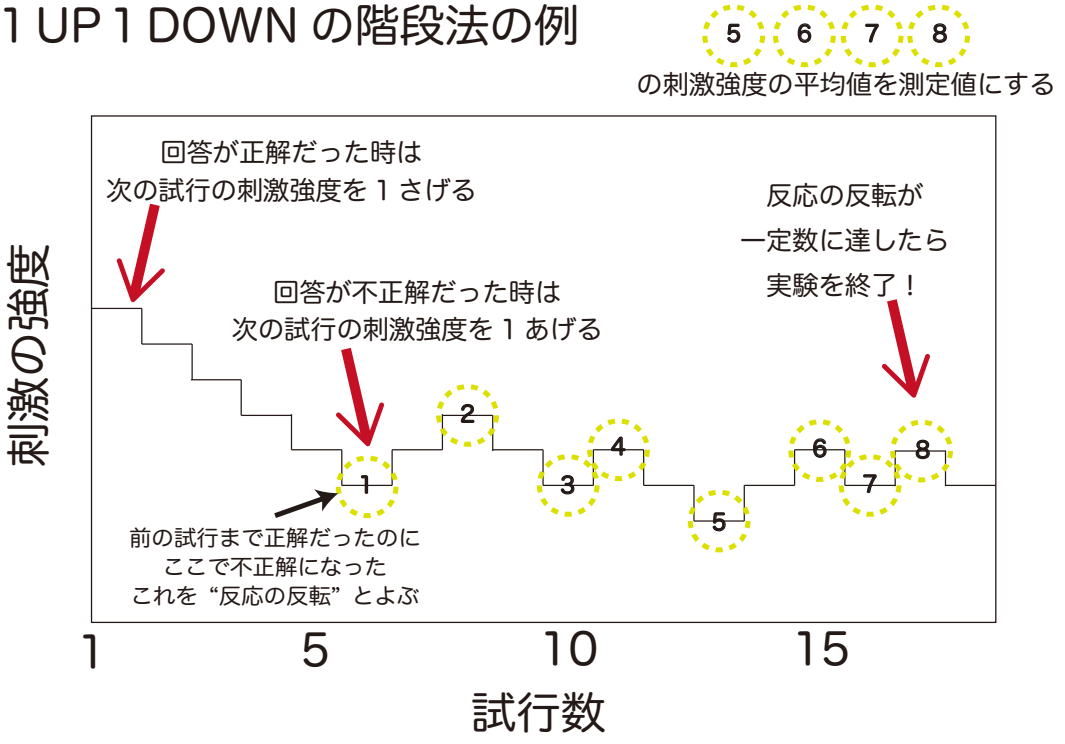
時間もかからず手軽に知覚が測定できる方法ですが、試行毎・参加者毎のばらつきが大きくなる特徴があります。

階段法

刺激強度を数量的・物理的に操作できる場合に用います。より厳密に知覚を測定したい場合、階段法（適応法）を用います。階段法は極限法の一種で、極限法には他に上下法などがあります。

階段法は、***実験者が刺激強度を一定の間隔で変化させて参加者の知覚を測定する方法***です。参加者が直前の試行と同じ反応をしている間は刺激強度を弱くしていき、反応が変化した場合は刺激強度を大きくします。あらかじめ決めておいた数だけ反応が反転する点が得られたら実験を終了します。

1 UP 1 DOWN の階段法の例



① 刺激強度の変化幅は 1

② 1 回正解したら刺激強度を 1 さげて、1 回不正解だったら刺激強度を 1 あげる

③ 8 回、反応の反転があったら終了する

④ 階段法の測定値は、最後の 4 回の反応の反転の刺激強度の平均値をとる

呈示した刺激強度と試行番号でプロット図を作るとそれが階段のようにみえることから階段法と呼ばれます。呈示する刺激強度を直前の参加者の反応に応じて適応的に変化させるため、**適応法**とも呼ばれます。

階段法を用いる場合、実験者は

① 刺激強度を変化させる幅
② 刺激強度の変化反転のルール
③ 試行終了のルール

の 3 点をあらかじめ決めておく必要があります。

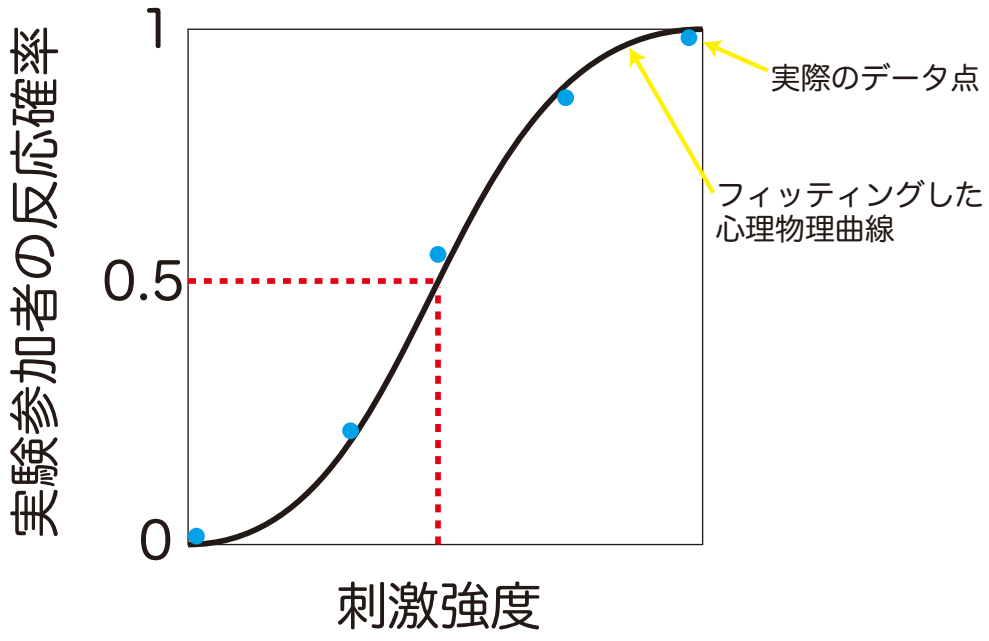
階段法には N-up-M-down（N 回誤反応の後に刺激強度を 1 段階強くし、M 回連続正反応の後に刺激強度を 1 段階弱くする。N と M に入る数値によって測定できる反応確率が異なる）と呼ばれる形式のシンプルなものから、PEST や QUEST と言った比較的複雑なルールを持つものや統計的推定を行いながら進める形式もあります。

恒常法

刺激強度を数量的・物理的に操作できる場合に用います。より詳細に厳密に知覚を測定したい場合、恒常法を用います。

恒常法では、実験者があらかじめ呈示する複数の刺激強度と呈示回数を決めておきます。***実験では各刺激強度をランダムで複数回呈示して、各刺激強度の反応確率を求めます***。この刺激強度の反応確率プロットから**心理物理曲線（psychometric function）**を求めます。

一般的に、反応確率のみならず、生体反応を刺激強度ごとにプロットすると S 字状のシグモイド曲線になることが知られています。この反応確率に S 字になる関数をフィットし得られた心理物理曲線から、閾値や主観的等価点を求めます。



シグモイド関数の傾きや中心は、心理物理曲線で求める重要なパラメータとなります。

一般的に、**評定法＜調整法＜階段法＜恒常法**の順に得られる情報量が多く精度も高くなり、参加者のもつ反応バイアスにデータが影響されにくくなります。しかし、かかる時間も情報量に比例します。

★ ***ちょっとポイント***

恒常法は各刺激強度の繰り返し呈示が多く必要なため時間がかかります。そのかわり、恒常法では心理物理曲線から、その刺激に対する感度や弁別閾と絶対閾を同時に測ることができたりします。評定法等は 1 人あたりにかかる時間は少なくなり負担も減りますが、個人の反応バイアスなどの影響を受けやすいため被験者数が多く必要になります。

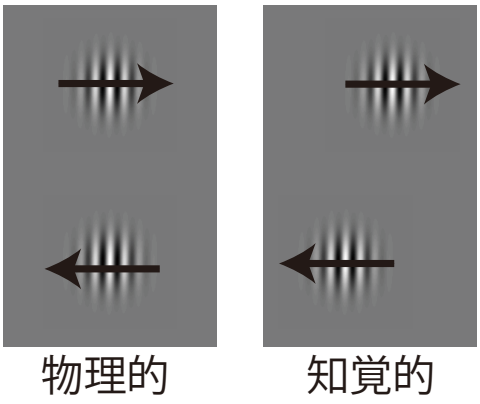
自分がどのような実験デザインを組み、最終的にどのようなデータを得たいのか、その後の解析のことまで考えながらどの測定法が最適なのかを考え選ぶようにしましょう。

第５章　知覚を測ってみよう！～運動による位置ずれ錯視の実験～

　測定法がそれぞれ具体的にどういったものなのか、説明だけではわかりにくいかもしれません。ここでは、ある錯覚現象の実験を実際にやってみてそれぞれの測定法の違いを体験してみましょう。

5.1. 運動による位置ずれ錯視

　デモ実験のテーマとして、「運動による位置ずれ錯視」を取り上げます。ガボールパッチの内部の正弦波が動いている時、ガボールパッチ全体の位置が運動方向側にずれてみえる錯覚です。



　錯視というのは、知覚上にしか現れないものなので「どれくらい錯視が起こるのか？」というのは条件と個人によって違います。実際に錯視がどれくらい感じられたかを数値化するためには、心理物理測定法を用いて個人の知覚を測ってみるしかありません。

　ここでは、錯覚とは逆方向の刺激を物理的に呈示し、錯覚を相殺する刺激量を測定することによって錯覚量を測る、**相殺法**というやり方を試してみましょう。

運動による位置ずれ錯視の場合、錯覚は”知覚位置“が変化することになるので、相殺するのに操作する刺激強度は“**刺激の呈示位置**”ということになります。物体位置を測定する時には、単体刺激のみの呈示ではずれているかどうかわからないので、位置のリファレンスとなる刺激も用意します。リファレンスからどれくらいずれてみえるのか？が”ずれ量”となります。

相殺法では、運動刺激の位置を錯視が起こる方向とは逆に呈示し、「**錯視がみえなくなる（リファレンスと同じ垂直位置になる）刺激呈示位置**」をもって錯視を測ります。つまり、**リファレンスと同位置にみえる主観的等価点**を測ることになります。

5.2. 運動による位置ずれ錯視を各測定法で測る

デモプログラムは 3 種類あります。

調整法で位置ずれを測るプログラム

`MotionShiftIllusion_withAdjustment.m`

fを押すと左に、jを押すと右に刺激がずれていくので、垂直にみえる位置になるように調整し、調整が終わったらspaceキーをおす

階段法で位置ずれを測るプログラム

`MotionShiftIllusion_withUpDown.m`

刺激が呈示され終わったら、f(左)か j(右)キーを押して、運動しているガボールパッチが上下の刺激に比べてどちらにずれていたかを回答する

恒常法で位置ずれを測るプログラム

`MotionShiftIllusion_withConstant.m`

刺激が呈示され終わったら、f(左)か j(右)キーを押して、運動しているガボールパッチが上下の刺激に比べてどちらにずれていたかを回答する

です。それぞれ 3 分〜5 分で終わりますので、まずは自分の錯視量をそれぞれの方法で測定してみましょう。

条件は右運動、左運動、静止の 3 条件あります。階段法と恒常法の 1 試行の刺激呈示時間は 1 秒です。調整法では被験者の調整が終わるまで刺激は呈示され続けます。

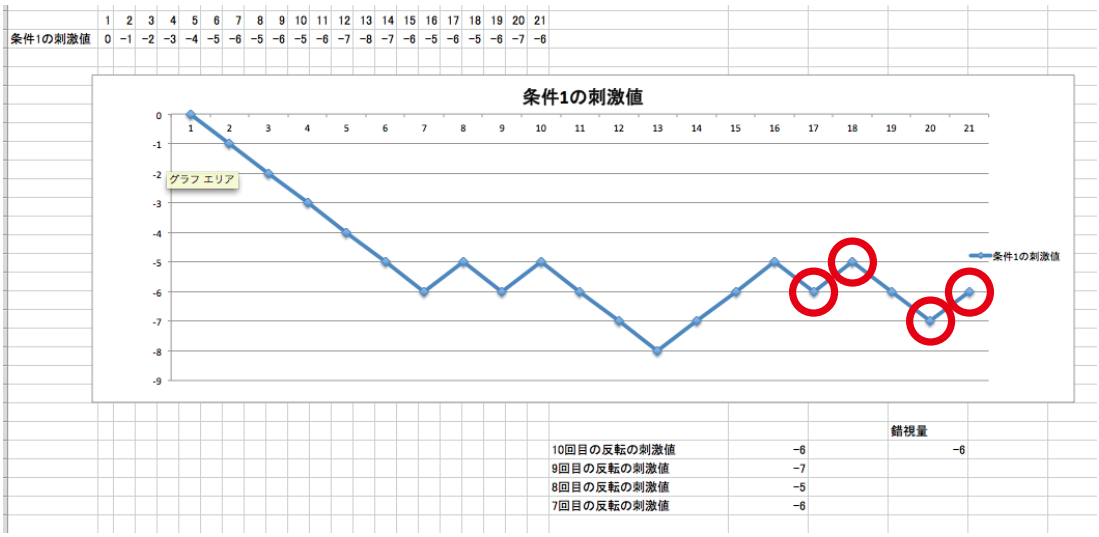
5.2.1 調整法で位置ずれを測る

調整法では各条件10回繰り返しました。データは行列のData_rawの3行目、もしくは構造体Dataの`AnswerSheet`に入っています。条件毎に10回の平均値を算出し、それをそれぞれの主観的等価点とします。

5.2.2 階段法で位置ずれを測る

ここでは、条件毎に 1 つの階段を求めました。ステップサイズは 6arcmin としています。10 回反応の反転があったら終了するルールでした。主観的等価点は、最後の 4 回の反応反転の刺激呈示位置の平均をとることにします。

階段法では、きちんとデータがとれているかを確認するためにそれぞれの条件の階段をグラフに描いて確認しましょう。



図は、ある条件の刺激強度の変化を試行ごとにエクセルでプロットしたものです。最後の 4 回の反応の反転を取る場合は、このように赤で囲んだところの刺激強度の平均値をとります。

デモプログラムでは、各試行の刺激強度は構造体Dataの`Stimulus`の中に入っています。

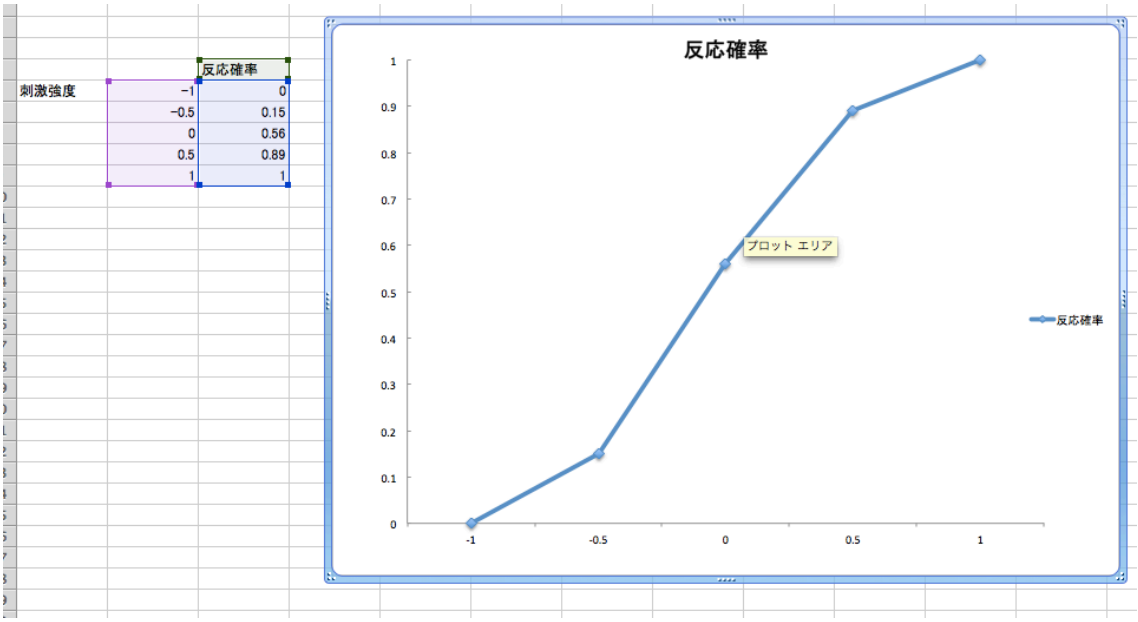
慣れてきたら、構造体の中に、「反応が反転した試行での刺激強度」が保存されるように書き換えてみたらよいと思います。

5.2.3 恒常法で位置ずれを測る

恒常法は、あらかじめ定めた 5 点の刺激強度をそれぞれ 10 回ずつ呈示しました。各条件につき 1 本の心理物理曲線を描きます。本来なら各ポイント最低 30 回は繰り返しが欲しいところですが、ここでは時間がないので 10 回繰り返しで反応確率を求めます（時間があれば休憩をいれて、このプログラムを 3 回繰り返してみてください。その際はくれぐれもファイル名を変えるのをわすれずに！）。

反応確率は「リファレンスよりも運動刺激が右側にあると回答した確率」です。各運動方向の心理物理曲線はどのように変化しているでしょうか？

心理物理曲線を求める前に、エクセルで刺激強度ごとの反応確率を折れ線グラフでプロットしてみましょう。どのようになっているでしょうか？



ある反応確率プロットの例

この反応確率のデータに心理物理曲線を任意の関数でフィッティングし、その関数の 50%点が主観的等価点となります。

心理物理曲線のフィッティングには Psignifit や Palamedes というツールもありますのでそれを使用してもよいでしょう（ただし Psignifit は Python 推奨なので Matlab のバージョンが対応していない場合もあります）。

Psignifit
http://psignifit.sourceforge.net/

Palamedes
http://www.palamedestoolbox.org/

5.2.4 それぞれの手法で測った錯視量と標準偏差を比較する

それぞれの手法で得られた錯視量を比較するために、グラフを作成してみましょう。

☆☆☆☆ヘルプの参照の仕方☆☆☆☆

基本的に、コマンドラインに
>>help 関数名

と打ち込んでください。その関数に関する説明が出てきます。

Screen(～で始まるものは、PTB の Screen 関数（群）と呼ばれるものです。Screen 関数にはさまざまな使用方法がありますが、もし Screen 関数内の特定の関数について使い方を調べたい場合は

>>Screen 関数名?

もしくは

>>Screen('関数名?')

と打ち込んでください。

例）Screen('OpenWindow',…)について調べたい場合

>>Screen OpenWindow?

もしくは

>>Screen('OpenWindow?')

とコマンドラインに打ち込む

補足① MATLAB のプログラミングについて

①変数

変数とは，データを一定期間記憶し必要なときに利用できるようにするために名前をつけたものです。変数の名前はアルファベット，数字，アンダーバー「_」を使って自由に決めてください。最初の文字はアルファベットであること，大文字と小文字は区別されること，認識される最大文字数があること（nemelengthmax で知ることが出来ます）に注意してください。

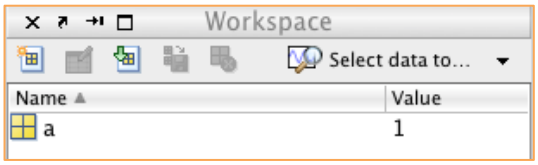
変数「a」

```
>> a=1

a =

     1
```

Workspace には以下のように表示されます。



②行ベクトル

MATLAB では，1 つの変数に複数の値を代入することも出来ます。1 行からなる変数のことを「行ベクトル」と呼びます。大事な点は，括弧 [ ]をつけることです。

```
>> a=[1 2 3 4 5 6]

a =

     1     2     3     4     5     6
```

コンマを入れても同じです。

```
>> a=[1, 2, 3, 4, 5, 6]

a =

     1     2     3     4     5     6
```

③列ベクトル

縦１列のベクトルを入力したい場合は，数値の後に「；」を入れてください。

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

a =

     1
     2
     3
     4
     5
     6
```

④行ベクトルを列ベクトルに変える

「行ベクトルから列ベクトルに変える場合は「‘」をつけてください。同様に列ベクトルを行ベクトルに変えることも出来ます。

```
>> a=[2 5 7 9]

a =

     2     5     7     9

>> a'

ans =

     2
     5
     7
     9
```

⑤配列

エクセルの表のように行と列の両方を持つ変数は配列変数と呼ばれます。

```
>> a=[1 2 3; 4 5 6; 7 8 9;]

a =

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

⑥ 組み込み関数

配列の大きさを知る関数「size」

変数の大きさを知るためには関数「size」を使ってください。

```
>> a=[1 2 ; 4 5 ; 7 8 ;]

a =

     1     2
     4     5
     7     8
```

サイズを調べてみましょう。

```
>> size(a)

ans =

     3     2
```

「nrows」 「ncols」に列と行の数を代入出来ます。

```
>> [nrows,ncols] = size(a)

nrows =

      3


ncols =

      4
```

「列だけのサイズ」 と「行だけのサイズ」を表示することも出来ます。

```
>> size(a,1)

ans =

     3
```

```
>> size(a,2)

ans =

     2
```

配列を初期化する関数

変数の初期化は関数「ones」や「zeros」を使います。

「zeros 関数」を使うことで「0」が代入された配列を作ることができます。

```
>> a=zeros(2,3)

a =

     0     0     0
     0     0     0
```

「ones 関数」を使うことで「1」が代入された配列を作ることができます。

```
>> a=ones(2,3)

a =

     1     1     1
     1     1     1
```

配列に「255」をかけると，全ての値が「255」になります。これは白い四角形を描くときに便利です。

```
>> a=ones(2,3)*255

a =

   255   255   255
   255   255   255
```

特殊文字列

詳細な情報が知りたい場合は, help ops とタイプしてください。

①コメント　パーセントサイン(%)

後ろの文章をコメントとして扱います。必要な情報は，後で見た人が分かるようにコメントとして残しておきましょう。

※Octave で PTB を使う場合は日本語を使うと文字化けします。

②セミコロン(;)

「セミコロン(;)」 はコマンドライン上に表示されるのを防ぎます。

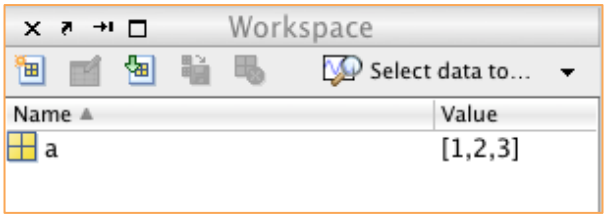
```
>> a=[1 2 3]

a =

     1     2     3

>> a=[1 2 3];
```

Workspace には表示されます。



③コロン（ : ）

配列を範囲で指定する際に使います。具体的には以下の例を見てください。２つの書き方で出力される結果は同じです。

```
>> a=[1 2 3]

a =

     1     2     3
```

```
>> a=1:3

a =

     1     2     3
```

以下のような書き方をすれば，「2」ずつ増えていくベクトルを作れます。

```
>> a=[2 4 6 8]
a =
     2     4     6     8
>> a=2:2:8
a =
     2     4     6     8
```

ちなみに以下のような書き方をすると，何も入力されません。

```
>> a=8:2
a =
   Empty matrix: 1-by-0
```

「8」から「2」に減らしていく場合は以下のように書いてください。

```
>> a=8:-1:2
a =
     8     7     6     5     4     3     2
```

④「括弧（()」何番目か？

「括弧内の数字」は，配列の「何**番**目か？」という意味です。例えば，「a(3)」は「a」の３番目の値を指定していることになります。

まず a に４つの数字を代入します。

```
>> a=[5 6 7 8]
a =
     5     6     7     8
```

a の１番目から４番目までを指定します。

```
>> a([1 2 3 4])
ans =
     5     6     7     8
```

```
>> a(1:4)
ans =
     5     6     7     8
```

別の配列の値を使う事もできます。配列「b」には「１～３」が入っており，その値を使う事で，「a」の「b 番目」の数値を出力します。

```
>> b=1:3
b =
     1     2     3
>> a(b)
ans =
     5     6     7
```

順番をランダムに指定すると並び替えをすることが出来ます。

```
>> a([1 4 3 2])

ans =

     5     8     7     6
```

配列の大きさを超えてしまった場合にはエラーが返されます。配列「a」は４つの値しか保存していませんので，５番目以降の数値は無いと返されます。

```
>> a([5 6 7 8])
??? Index exceeds matrix dimensions.
```

２次元以上の配列も同じように扱う事が出来ます。

```
>> a=[9 8 7 6; 5 4 3 2]

a =

     9     8     7     6
     5     4     3     2
```

以下のように，１列目の１行目，１列目の２行目，２列目の１行目，２列目の２行目といった感じで数値が読み出されます。９番目を指定するとエラーになります。

```
>> a(1)

ans =

     9

>> a(2)

ans =

     5

>> a(3)

ans =

     8

>> a(4)

ans =

     4

>> a(5)

ans =

     7

>> a(6)

ans =

     3

>> a(7)

ans =

     6

>> a(8)

ans =

     2

>> a(9)
??? Index exceeds matrix dimensions.
```

行や列全部を指定したい場合にはコロン（：）を使います。

```
a =

     9     8     7     6
     5     4     3     2

>> a(:)

ans =

     9
     5
     8
     4
     7
     3
     6
     2
```

「１列目」の値のみ取り出したい時は以下のように指定してください。

```
>> a(:,1)

ans =

     9
     5
```

「１行目」の値のみを取り出したい時は以下のように指定してください。

```
>> a(1,:)

ans =

     9     8     7     6
```

以下のようにすると「１列目」と「３列目」を取り出す事が出来ます。

```
>> a(:,[1 3])

ans =

     9     7
     5     3
```

以下のように１列目と３列目の値に「０」を代入すると，特定の列の値が「０」になります。

```
>> a(:,[1 3])=0

a =

     0     8     0     6
     0     4     0     2
```

式の左辺と右辺が一致していないとエラーになります。エラーが出た場合には右辺と左辺のそれぞれを「size 関数」で調べてみると問題が明らかになったりします。

四則演算

help arith で詳しいヘルプが出ます。

① 変数の演算

足し算

```
>> 5+2
ans =
     7
```

引き算

```
>> 5-2
ans =
     3
```

かけ算

```
>> 5*2
ans =
    10
```

割り算

```
>> 5/2
ans =
    2.5000
```

累乗

```
>> 5^2
ans =
    25
```

足し算とかけ算の組み合わせ

```
>> 2*3+1
ans =
     7
>> 2*(3+1)
ans =
     8
```

② ベクトルと変数の演算

以下のように「２」は，ベクトル全体の値にかけられます。

```
>> a=[4 5 6];
>> 2*a

ans =

     8    10    12
```

同じようにベクトルの全ての値に「５」が足されます。

```
>> a+5

ans =

     9    10    11
```

参考文献

Matlab について

Matlab 入門　高井信勝　2002 工学社

Matlab プログラミング入門　上坂吉則　2011 牧野書店

Psychtoolbox について

はじめよう実験心理学: MATLAB と Psychtoolbox を使って 実吉 綾子・前原 吾朗 2015 勁草書房

ガボールパッチについての文献

ガボール刺激と空間定位　蘆田宏　2006 VISION vol.18 pp.23-27.

心理物理学的測定法について

心理学研究法 1　感覚・知覚　大山正　2011　誠信書房

イラスト素材：

かわいいイラスト素材いらすとや

http://www.irasutoya.com/p/terms.html

Illust AC

http://www.ac-illust.com/

VSJ　Programming Workshop　2015

2015年10月17日　初版発行

著者・発行者
　視覚心理実験プログラミングワークショップ運営委員会
　URL：
http://visitope.org/workshop/ProgrammingWorkshop/index.html
　E-mail（連絡先）： programmingws2015open@googlegroups.com
共催
　日本視覚学会若手の会
後援
　NIJC・Visiome platform