title

情報学群
1270328 佐藤謙成

2025 年 5 月 26 日

概要

1　全体の目的

1.1　第六回の目的

第六回では，地球に到達する光を分析することにより，恒星の動きを知ることを目的とする．恒星がどの程度の速度で地球から遠ざかっているのかを算出し，観測されたスペクトルを描画する．また，単一の恒星についての分析と複数の恒星についてどのよう分析を行い，〜〜する．

1.2　第七回の目的

第七回では，次回の眼球運動継続実験の準備として MatLab 及び Psychotoolbox を用いてプログラムを作成する．本課題では，被験者が左右に提示される顔画像の魅力を判断し，キー押しで反応する二者択一の選択課題を 2 試行行う実験環境を構築することである．

1.3　第八回の目的

1.4　第九回の目的

第八回では，EyeLink II によって過去に測定された眼球運動実験のデータをプログラムを用いて解析することである．被験者が左右の顔画像のどちらかを選択する際に，選択した顔画像んいどの程度視線を向けていたかの確率を時間経過とともにグラフかすることを目的とする．

1.5　第十回の目的

2　方法

2.1　第六回の方法

```
load starData
nObs = size(spectra,1);
```

??では，まず starData というデータファイルを読み込み，`size(spectra,1)` で各スペクトルに含まれる観測点の数を取得する．

次に，恒星 HD94028 のスペクトルを抽出する．それが以下のコード **??**である．

```
figure(2)
loglog(lambda, s, ".-")

xlabel("Wavelength")
ylabel("Intensity")

```

ここでは，両軸対数スケールで出力するために関数 loglog を用いている．また，恒星 HD94028 のスペクトルはベクトル s に格納する．

ベクトル s に格納された値を用いて水素アルファ線の波長を求める．コード **??** では， s の最小値が水素アルファ線であることを利用して `min(s)` で求めている．関数 min はここでは 2 つの値を出力することができる．一つ目の値が水素アルファ線の値で二つ目の値がそのインデックスとなる．

```
%task4
[sHa, idx] = min(s)
lambdaHa = lambda(idx)

```

特定した水素アルファ線に対して点を追加するためのコードがコード**??**である． ここでは， `hold on` を記述することで既存のグラフに点を追記することができ， `hold off` で追記を終了することができる．今回は，水素アルファ線の値の部分にマーカーサイズが 8 の赤い正方形をプロットする．

```
%task5
hold on
    plot(lambdaHa, sHa, 'rs', 'MarkerSize', 8)
hold off
```

赤方偏移係数と星が地球から遠ざかる速度を求める．赤方偏移係数は係数を z としたときに $z = \dfrac{\lambda Ha}{656.28} - 1$ で求めることができるため，これを実装する．速度に関しては，赤方偏移係数に光速の値をかけることで求めることができる．

```
%task6
z = (lambdaHa / 656.28) - 1
speed = z * 299792.458
```

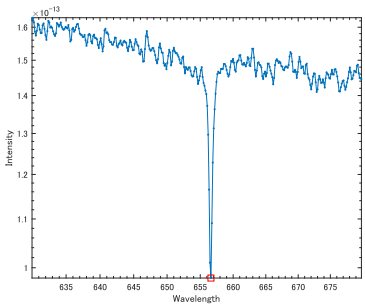

図 1: 恒星 HD94028 のスペクトルと水素アルファ線

ここから，各星の水素アルファ線を求める．行列の各行に対して最小値とそのインデックスを計算する．これは `min(spectra(:,:))` で求めることができる．また，各星の Ha の波長を `lambda(idx)` で求め，その値を用いて赤方偏移係数と速度を求める．

```
lambdaHa = lambda(idx);
z = lambdaHa/656.28 - 1;
speed = z*299792.458;
%task3,4,5
figure(3)
```

この各星で求めた値を 1 つの図にまとめるプログラムで出力する．全ての星のスペクトルを一つのグラフに目止めて描画し，青方偏移か赤方偏移を視覚的に区別し，どの線がどの星に対応するかを明確にする．各星について速度が 0 以下の場合は青方偏移，0 より大きい場合は赤方偏移と判断するため，if 文を用いて条件分岐を行う．また，その条件分岐を各星に対して行うために for 文を用いる．

```
for c = 1 : 1 : 7
    s = spectra(:,c);
    if(speed(c) <= 0)
```

```
        loglog(lambda, s, "--")
    else
        loglog(lambda, s, "LineWidth", 3)
    end
    hold on
end
hold off

%task6
```

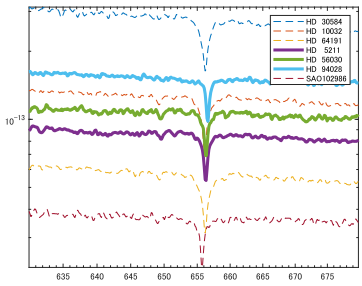

図 2: 各星のスペクトルと水素アルファ線

2.2 第七回の方法

まず，刺激呈示環境を構築する．まず，背景色を白色とするために `bgcolor = 255` と設定する．刺激画像の大きさは 640 × 480 とするので 高さは `h = 640` とし，幅は `w = 480` とする．

画面左右端と画像端との距離は 220 ピクセルとするため， `margin = 220` とする．固視点は 画像中央に幅 40，縦 4 及び 幅 4 縦 40 の黒色の十字とする．黒色は 0 で指定することができ，固視点の十字は長方形を二つで組み合わせることで表現できる．ここで長方形は `[left top right bottom]` でベクトルで定義できる．

```
trialnumber = 2;                                    % ★修正：試行数を 2 に設定
h=640;                                              % 刺激画像の横のサイズ（pix）
v=480;                                              % 刺激画像の縦のサイズ（pix）
textcolor=0;                                        % 文字色 （黒）
% 課題 bgcolor=という形で作成する。                 % 背景色
bgcolor = 255;                                      % ★修正：背景色を白に設定 （白は 255）
% 課題 fxcolor = という形で作成する。               % 固視点色
fxcolor = 0;                                        % ★修正：固視点色を黒に設定 （黒は 0）
```

```
    rectF1=[rectW(3)/2-rectF(1)/2 rectW(4)/2-rectF(2)/2 rectW(3)/2+rectF(1)/2
    ↪  rectW(4)/2+rectF(2)/2]; % 横棒
    % 課題 上の rectF1 を参考に rectF2 というのを作成する
    rectF_vertical = [rectF(2) rectF(1)]; % 縦棒用に幅と高さを入れ替え
    rectF2=[rectW(3)/2-rectF_vertical(1)/2 rectW(4)/2-rectF_vertical(2)/2
    ↪  rectW(3)/2+rectF_vertical(1)/2 rectW(4)/2+rectF_vertical(2)/2]; % ★修正：縦棒

```

Listing 2.1: 固視点の表示

固視点呈示後，準備された左右の顔画像を表示する．それはコード 2.2 で表示することができる．

Listing 2.2: 刺激の呈示

```
        % 刺激画像を呈示する。
        Screen(window,'FillRect',bgcolor);                  % 背景ウィンドウの描画
        Screen(window,'PutImage',fileL,rectIL);             % 左画像の描画
        Screen(window,'PutImage',fileR,rectIR);             % 右画像の描画
        [VBLTimestamp StimulusOnsetTime FlipTimestamp Missed Beampos] = Screen(window, 'Flip');
        ↪  % FlipTimestamp が刺激画像を呈示した時間になる
```

刺激を呈示した後の回答を格納するための関数は KbWait([], 3) を用いる．KbWait の返り値の一つである keyCode1 で押されたキーが確認できるが 1×256 の論理配列であり，各キーに対応した要素を真である．keycode1 で真になった要素を探索するため関数 Find を使用して行った．そのコードが 2.3 である．

Listing 2.3: 反応時間の取得

```
        % 回答キーが押されたらキーコードと反応時間を取得する。
        [secs, keyCode1, deltaSecs] = KbWait([], 3); %
        res = find(keyCode1);    %  キーコードの取得

         % 回答（キーコード）  (g はコード 71、j はコード 74)
         results(i,2) = res(1); % 最初に押されたキーを記録
```

これで，全ての試行が完了した後は，results 配列にデータを格納するコードをコード 2.4 に示す．

Listing 2.4: データの保存

```
         % 回答（キーコード）  (g はコード 71、j はコード 74)
         results(i,2) = res(1); % 最初に押されたキーを記録

         % 反応時間
         % 課題     %回答を参考にして、反応時間を results の 3 列目に入れる。
         results(i,3) = tKey - stimOnset; % ★修正: 反応時間を計算して記録
```

最後に，results 配列の値を csv ファイルに書き出した保存する．ここでは writematrix 関数を使用して `writematrix(results, filename)` と記述することで書き出すことができる．ここでの filename は，exp3_7_05.csv のことであり，拡張子に csv がついているため csv ファイルで保存される．

Listing 2.5: データの書き出し

```
    % writematrix を使ってデータをカンマ区切りで保存する。ファイル名は exp3_7_**.csv とする。**には班の数字
    ↪  を入れる。
    group_number = '05'; % ★修正: ここに班の数字（例: '01'）を入れる
    filename = ['exp3_7_' group_number '.csv'];
    writematrix(results, filename);                    % ★修正: 結果を CSV ファイルに保存
```

2.3 第七回の結果

第七回では眼球運動計測実験で使用するプログラムを作成した．このプログラムを実行した後に書き出される csv ファイルには 1 列目に 試行番号， 2 列目に 被験者が押したキーに対応するキーコード， 3 列目に反応時間が入力される．これらのデータを用いて眼球運動に関してのデータ分析をすることが期待できる．

2.4 第八回の方法

2.5 第九回の方法

結果を格納するための行列を作成する．行数はサンプリング周波数に対応する 500 行と列数は時間データ列と試行数となる．時間軸のデータについてはキー押しの 1 秒前からのデータを用いる．キーを押した時を 0msec として -998 までの 2msec おきの配列を作成する．これらを実装したコードがコード 2.6 である．

Listing 2.6: 結果格納用の準備

```
% 初期設定
clear all;
filename = 'exp4i_g0310.csv';
data = csvread(filename);

choice = data(:,2,:);
num_trials = length(choice);
sampling_freq = 500;
data_matrix = zeros(500, num_trials + 1);
time = -998:2:0;
data_matrix(:,1) = time;
```

ここから各試行ごとにデータ処理を行っていく．キー押しの瞬間を基準とした相対的な時間軸を用いるため，キーを押した時刻を 0 の基準として，元々格納している時間である絶対時間を用いて相対時間を求める．それが以下のコード 2.7 である．

Listing 2.7: 注視データの整形

```
    % 必要な情報を抽出して行列 expos を作成
    % 行はサンプル数を示す
    % 列は 1. 時間，2. 画面の状態，3. 左右どちらを見ているか，4. 選択した方を見ているか，5.X 座標

    end_time = eye_data(end, 1); % 最終行の時間 (終了時)
    adjusted_eye_time = eye_data(:, 1) - end_time; % 終了時を 0 とする
    times_to_zero = -2 * ((rows - 1) : -1 : 0); % キー押し時を 0 として時間を格納
    expos = zeros(rows, 5);
    expos(:, 1) = times_to_zero;
    expos(:, 2:4) = -1;
```

ここで，欠損値についても考えていく．欠損値とは，眼球運動の測定データにおいて，特定の時点でデータが取得されなかった場合や解析するにあたって無効なデータの場合は欠損として扱われる． EyeLink の測定では，まばたきやセンサの誤作動によってデータが欠けることがあるため， MatLab 上では欠損値として扱う．欠損データは -1 で初期化する．また，有効な行のみを対象に画面状態や X 座標を代入する．有効な行の

識別は ismember(expos(:, 1), eye_data(:, 1) - eye_data(end, 1)) でき， expos の各時間点で実際の眼球データが存在するかを確認する．その結果から有効な行のインデックスを取得する．expos の各列は以下の形式で入力を行う．

1 列目 時間
2 列目 測定する目の情報
3 列目 両目の状態
4 列目 左目注視位置 X 座標
5 列目 左目注視位置 Y 座標
6 列目 左目瞳孔径

この列に対応した情報を入れていくが有効な行のみを対象とするため， ismember で取得したインデックスを用いて行う．これらのコードをコード 2.8 に示す．

Listing 2.8: 欠損値の処理

```
29      %expos の時間列を「終了時 0 基準にした」eye_data の時間と比較
30      [is_valid_time, idx_in_eye_data] = ismember(expos(:, 1), eye_data(:, 1) - eye_data(end,
        ↪  1));
31
32      % is_valid_time は expos の時間が eye_data の時間に存在するかどうかを示す論理配列
33      % idx_in_eye_data は expos の時間が eye_data の時間に対応するインデックスを示す
34      valid_rows = is_valid_time;
35
36      % 欠損でない行に画面状態を代入に画面状態を代入
37      expos(valid_rows, 2) = eye_data(idx_in_eye_data(valid_rows),3);
38
39      % 欠損でない行に X 座標を代入
40      % 少数の値になるため 10 倍して整数に変換
41      expos(valid_rows, 5) = round(eye_data(idx_in_eye_data(valid_rows), 4)*10);
42
43      % 画面の状態が 1 の時の X 座標を抽出
44      filation_rows =  expos(:, 2) == 1 & expos(:, 5) ~= -1;
45      fixation_x_values = expos(filation_rows, 5);
46
47      % X 座標の平均を算出し，中心座標とする
48      center_x = round(mean(fixation_x_values));
```

これから，注視位置の判定を行う．画像提示時のうち，x 座標を取得し，これらの平均値を計算する．この平均値の四捨五入したものがその試行における画面の中心 x 座標とする．この中心よりも左であるならば左側の画像を選択したと判断し，右であれば右側の画像を選択したと判断する．ここで，左側を見ていた場合を 100, 右側を見ていたなら 102 とする．画像提示時は expos の 3 列目に格納されており，その時の値は 2 であれば刺激画像を提示してる時である．また，左目の注視位置は， 5 列目に格納されており，中心よりも左であるか右であるかの論理積を取ることでそれぞれ求めることができる．これらのコードをコード 2.9 に示す．

Listing 2.9: 注視位置の判定

```
    % 画像提示時のうち，X 座標が center_x より小さい行を 100，それより大きい行を 102 とする
    left_rows = expos(:, 2) == 2 & expos(:,5) < center_x;
    expos(left_rows, 3) = 100;

    right_rows = expos(:, 2) == 2 & expos(:,5) > center_x;
    expos(right_rows, 3) = 102;

    % どちらの画像が魅力的だったかを取得
    selected_side(i) = data(i,2);
```

注視方向と選択の位置判定はキー押しの選択と視線が一致していたら 1, 一致していなければ 0 とする．これを行うために expos の 3 列目の値が 2 の時に選択位置と注視位置の判定を行う．また，選択位置は 100, 102 のどちらかであり，注視位置は -1, 1 のどちらかであるため，それらの論理積を取ることで一致しているかを判定する． どちらの顔画像が魅力的かを判断したデータは data という行列の 2 行目に格納されているため，そのデータとキーコードが一致しているかを確認する．また，キー押し直線の一秒間に限定して，被験者が最終的に魅力的だと判断した画像を見ていたかどうかをのデータを格納する．これらを実装したコードをコード 2.10 に示す．

Listing 2.10: 注視位置と選択位置の一致

```
    % 欠損値でないデータを探す
    not_missing = expos(:, 3) ~= -1;

    % キー出力データファイルと比較して，一致していれば 1 を出力
    expos(not_missing & expos(:, 3) == selected_side(i), 4)  = 1;

    % 一致していない場合と欠損値の場合は 0 を出力
    expos(not_missing & expos(:, 3) ~= selected_side(i), 4) = 0;

    % 魅力的だと判断した画像の方向と見ている方向が一致しているかのデータを
    % キー押し時の 1 秒前からキー押し時まで取得し，対応している試行の行に値を代入
    % ここではキー押し時を 0 として，-998 ms から 0 ms の間のデータを取得
    time_window = expos(:,1) >= -998 & expos(:,1) <= 0;

    % 該当する expos 4 列目を抽出して代入
    selected_segment = expos(time_window, 4);
    data_matrix(:, i + 1) = selected_segment;
```