Psychophysics data

Table of Contents

`A quick analysis`

A simple analysis script for looking at psychophysical data and fitting a cumulative gaussian curve.

ds 2017-12-04

Load the data

```
% An example data set is stored inside the following mat file
load('sampleData')

% Data are organised thus:
% |orientationByTrial|  -3, -2, 0, 5, 1 ...
% |responseByTrial|     1..2..1..2..1..1..1 etc
```

Sort and summarise the data

We want to have all the for a given orientation combined and responses summarised such that we know the proportion of times people said "2" (right)

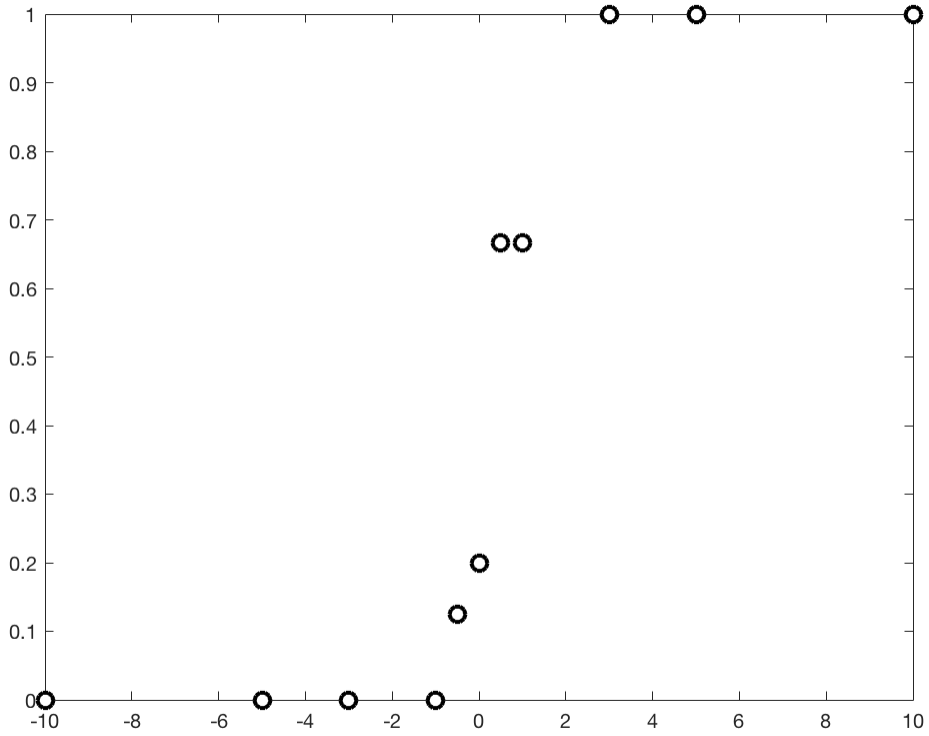
```
% the |sortData()| function does that... (you can use mine!)
data = sortData(orientationByTrial, responseByTrial)
```

```
data =

  -10.0000         0
   -5.0000         0
   -3.0000         0
   -1.0000         0
   -0.5000    0.1250
         0    0.2000
    0.5000    0.6667
    1.0000    0.6667
    3.0000    1.0000
    5.0000    1.0000
```

```
    10.0000    1.0000
```

Now make a simple plot

```
% If we keep track of the figure handle, we can re-use
% this figure window later.

figHandle = figure;
plot(data(:,1), data(:,2), 'ko', ...
    'markerfacecolor','w', ...
    'markersize',8, ...
    'linewidth',2)
```



Now to the fitting:

```
% re-wiring the input arguments of a function
% EXPLANATION...

% just to make explicit what is what: data
xData = data(:,1);
yData = data(:,2);

%                 [a function] [starting    [x]    [y]
%                                guess]
```

```
bestP = lsqcurvefit(@myNormcdf,   [0, 1],   xData, yData)

% the @ in @myNormcdf is needed to make sure matlab
% knows you are handing it a function (handle).


Local minimum possible.

lsqcurvefit stopped because the final change in the sum of squares
 relative to
its initial value is less than the default value of the function
 tolerance.




bestP =

    0.4501    0.7789
```

now that we have the best parameters...

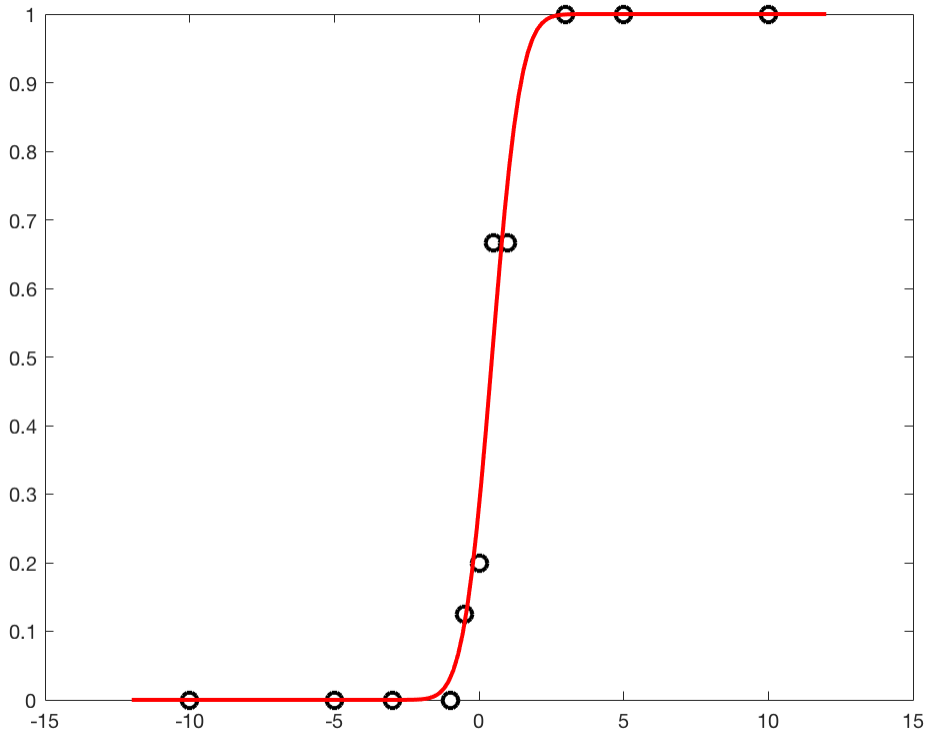
we can use the best parameters (bestP) to make a nice fitted curve. Because we have the parameters, we can calculate the y values for any x values have want (not just where the *data* were collected)

myNormcdf() is already set up to accept the parameters in this way

```
xForDisplay = linspace(-12, +12, 150);
yforDisplay = myNormcdf(bestP, xForDisplay);
```

now plot the fine grained function on top

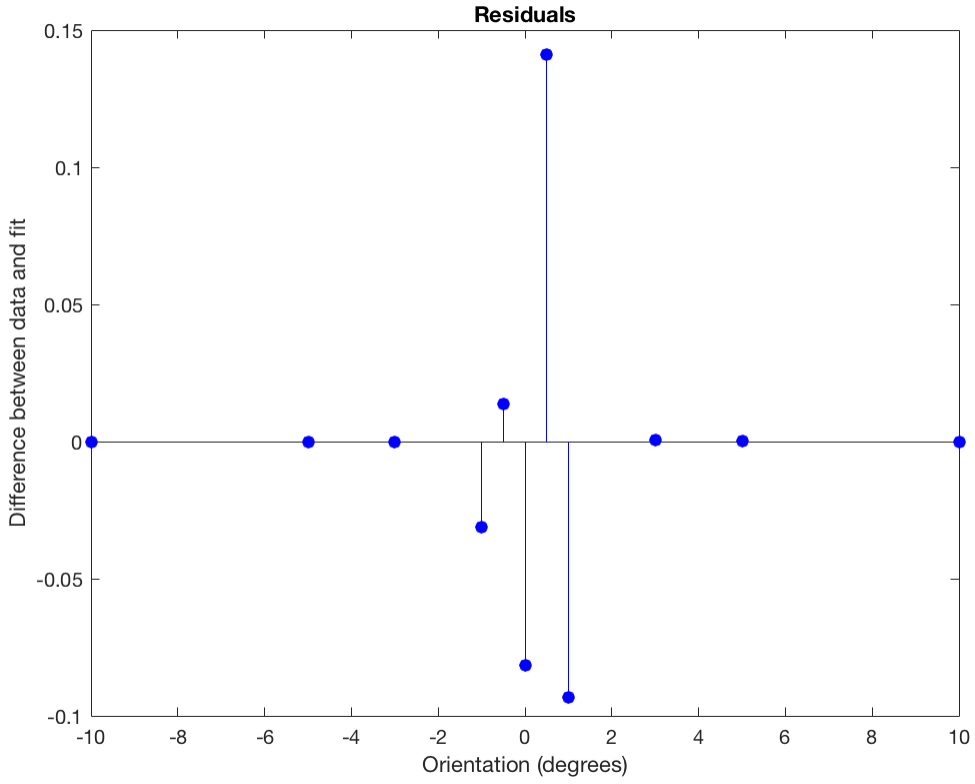
```
hold on
plot(xForDisplay, yforDisplay, 'r-', 'linewidth',2)
```

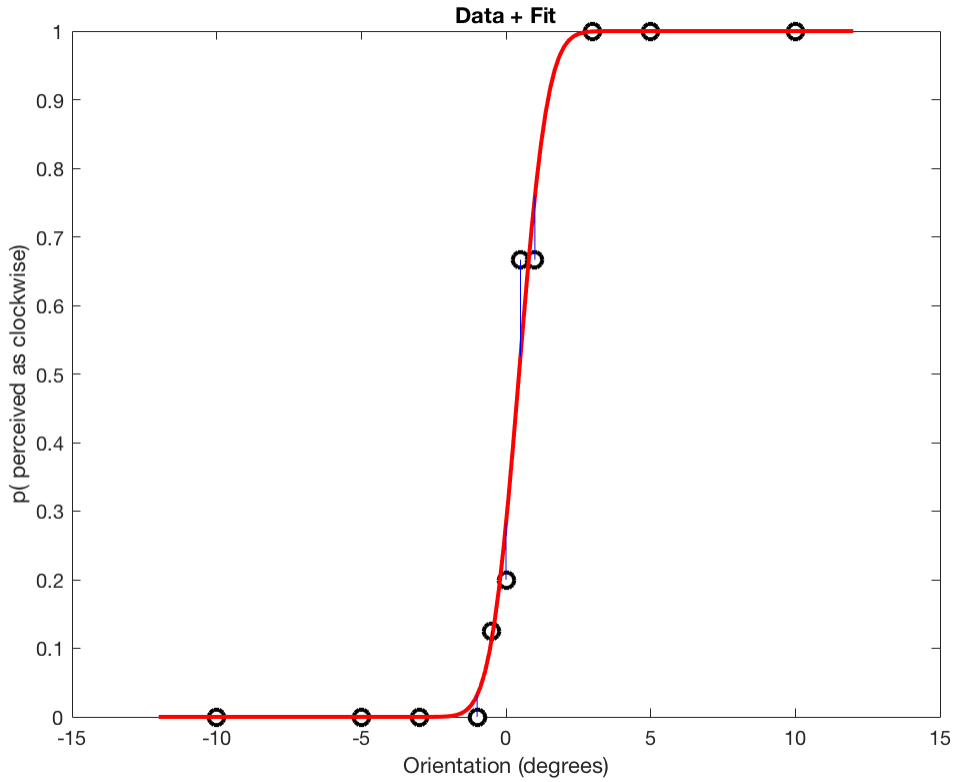


bonus:

calculate the residuals. we need the yData and y-values of the fit at the xData values

```
yData; % we already have
yFit = myNormcdf(bestP, xData); % y values on the fit curve
residuals = yData - yFit;

figure
stem(xData, residuals, 'b', 'filled')
xlabel('Orientation (degrees)')
ylabel('Difference between data and fit')
title('Residuals')

% now plot this info on top of data + fit plot
figure(figHandle)
hold on

% make many line segments
xLines = repmat(xData(:),1,2);
yLines = [yData(:), yFit(:)];
line(xLines', yLines', 'color', 'b')

xlabel('Orientation (degrees)')
ylabel('p( perceived as clockwise)')
title('Data + Fit')
```


Residuals
Difference between data and fit
Orientation (degrees)
0.15
0.1
0.05
0
-0.05
-0.1
-10
-8
-6
-4
-2
0
2
4
6
8
10


Data + Fit
p(perceived as clockwise)
Orientation (degrees)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-15
-10
-5
0
5
10
15

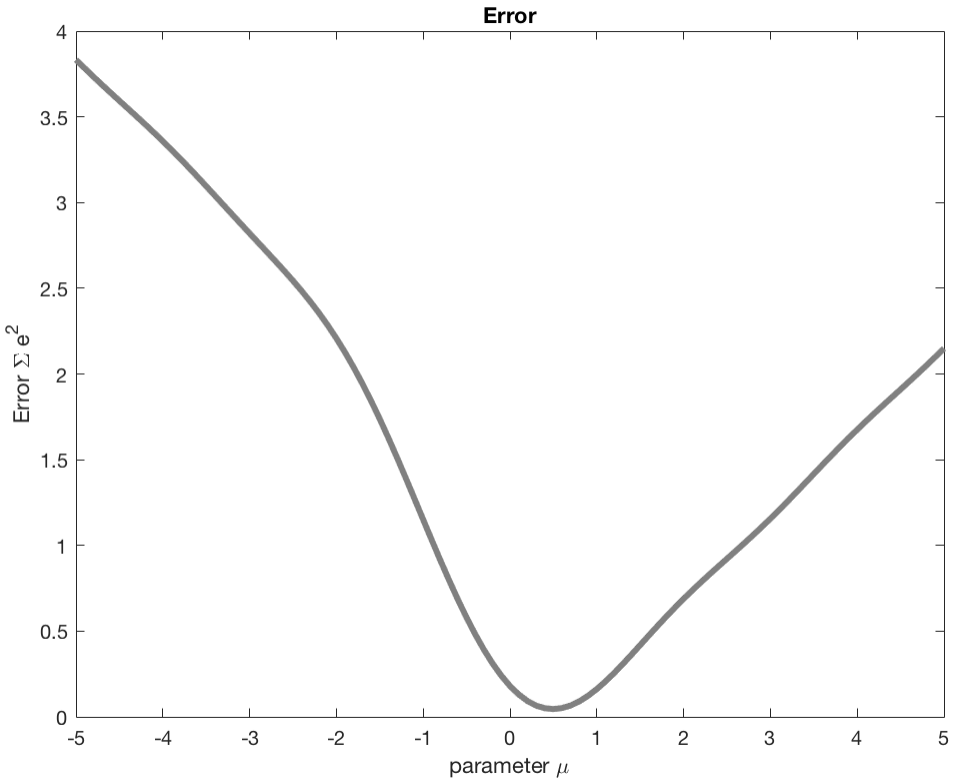
bonus 2:

Let's explore how well the data fits to curves with different choices of mu (and then sigma)

e = sqerror(p, xdata, ydata)

xData = data(:,1); yData = data(:,2);

```
% with a guess for one parameter
allMus = linspace(-5,5,101);
allE = nan(size(allMus));
sigmaGuess = 1.0;

for iMu = 1:numel(allMus)
    currentMu = allMus(iMu);
    allE(iMu) = sqerror([currentMu, sigmaGuess], xData, yData);
end

figure
plot(allMus, allE, 'color', [1,1,1].*0.5, 'linewidth',3)
xlabel('parameter \mu')
ylabel('Error \Sigma e^2')
title('Error')
```

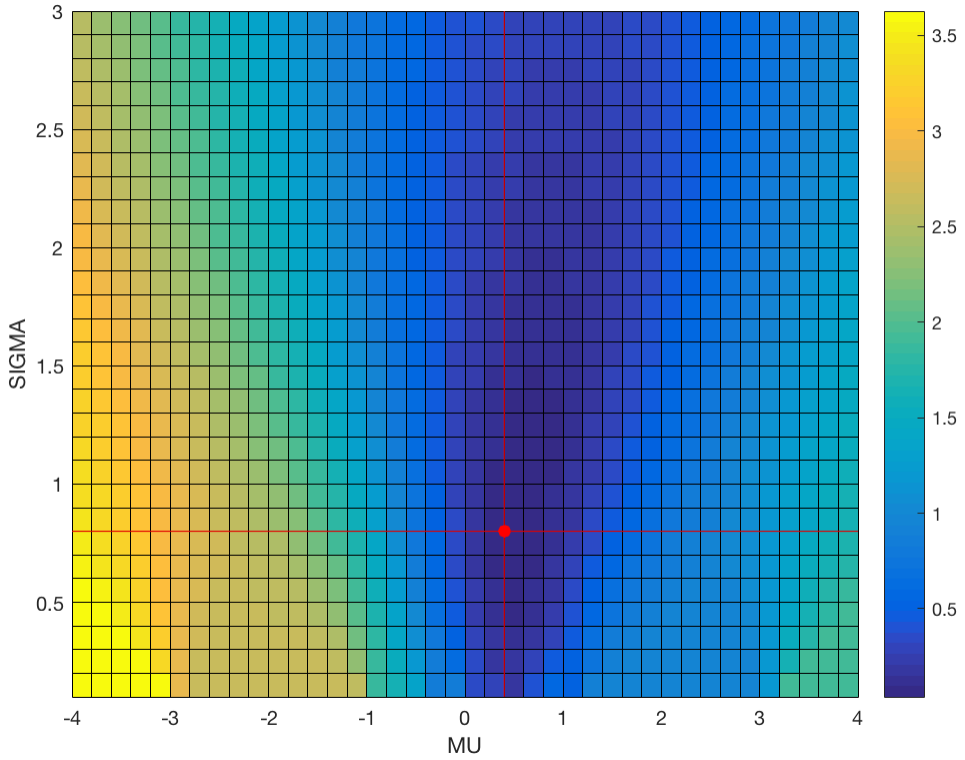


error surface

```
[M,S] = meshgrid(-4:0.2:4, 0.1:0.1:3);
% provide the function with inputs, element-wise from M and S.
% |arrayfun()| let's you do that, but it's a bit advanced, so maybe
% don't worry about this. The important point here is that we
 calculate
% an error |E| for each combination of MU and SIGMA values.

E = arrayfun(@(x,y) sqerror([x,y], xData, yData), M, S);
```

Make a 2d plot

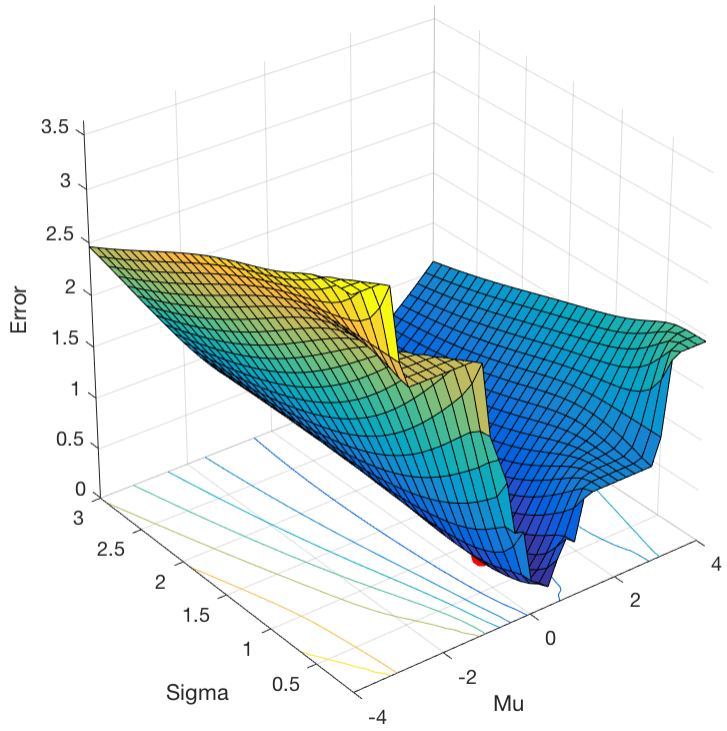
```
figure
pcolor(M, S, E)
% ginput(1)  % you can use this to query by clicking the mouse

% find the min value... use one long vector so you get the linear
 index.
[mval, mindx] = min(E(:));

% the values of M, S and E at that point
Mmin = M(mindx);
Smin = S(mindx);
Emin = E(mindx);

% som helper lines
vline(Mmin, 'r');
hline(Smin, 'r');

% and some labels
xlabel('MU')
ylabel('SIGMA')

colorbar()

plot(Mmin, Smin, 'ro', 'Markerfacecolor', 'r')
```



Make a surface plot

To get a more fancy-looking (but not necessarily easier to understand) 3d plot, you can do the following.

```
figure

surfc(M, S, E)

hold on
plot3(Mmin, Smin, Emin, 'ro', ...
    'markerfacecolor', 'r', 'markersize', 10)

xlabel('Mu');
ylabel('Sigma');
zlabel('Error');

% change the following figure properties to make it look nice
% when you interact with the plot
axis vis3d
camproj('perspective')
```



Published with MATLAB® R2016b