Report 5

Maximilian Fernaldy - C2TB1702

Exercises 5.1

- The table to the right shows the number of Nobel laureates per capita (i.e., divided by population) and chocolate consumption per capita for different countries
- It has been discovered that there is a strong link between these two cultural traits (Nobel laureates and chocolate consumption)
  - Franz H. Messerli, Chocolate Consumption, Cognitive Function, and Nobel Laureates, the New England Journal of Medicine, 367, 1562-1564, 2012
- Fit a line to the data and plot the results
  - You can download the file ('Nobel_vs_choco.txt') from Google Class CAPS05 assigment.
- Add an imaginary „CAPS Kingdom", which has 10×(A+B) Nobel laureates per capita and consumes 0.5×(C+D) kg/y/head of chocolate, then show and plot how the fitted line changes. A, B, C and D are the last 4 digits from your student number (see Exercise 4.1).

	Nobel laureates per capita	Chocolate consumption per capita (kg/y/head)
Sweden	31.855	6.6
Switzerland	31.544	10.8
Denmark	25.255	8.6
Austria	24.332	7.9
Norway	23.368	9.8
UK	18.875	10.3
Ireland	12.706	8.8
Germany	12.668	11.4
USA	10.706	5.1
Hungary	9.038	3.5
France	8.99	7.4
Belgium	8.622	6.8
Finland	7.6	7
Australia	5.451	6
Italy	3.265	3.3
Poland	3.124	4.5
Lithuania	2.836	6.1
Greece	1.857	4.5
Portugal	1.855	4.5
Spain	1.701	3.3
Japan	1.492	2.2
Bulgaria	1.421	2.2
Brazil	0.05	2.5

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3743834/

9

Problem 1

In the first problem, we are asked to fit a line to the data and plot the results. To do this, we can use the least square method. It dictates that in this linear equation:

$$Xp = y$$

$\hat{p}$ is an array $\begin{bmatrix} a \\ b \end{bmatrix}$ which contains values that correspond to the linear equation $y = ax + b$. X is an array containing our x-coordinates in the first column and all ones in the second value:

$$X \equiv \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix}$$

pseudoinverting X will make it possible to find p in this manner:

$$\hat{p} \equiv X^{\dagger} y$$

Loading the data into Octave

Using the `load()` function, we can load the array stored in the (provided) *Nobel_vs_choco_CAPS05_assignment.txt* and store them in a variable `table`.

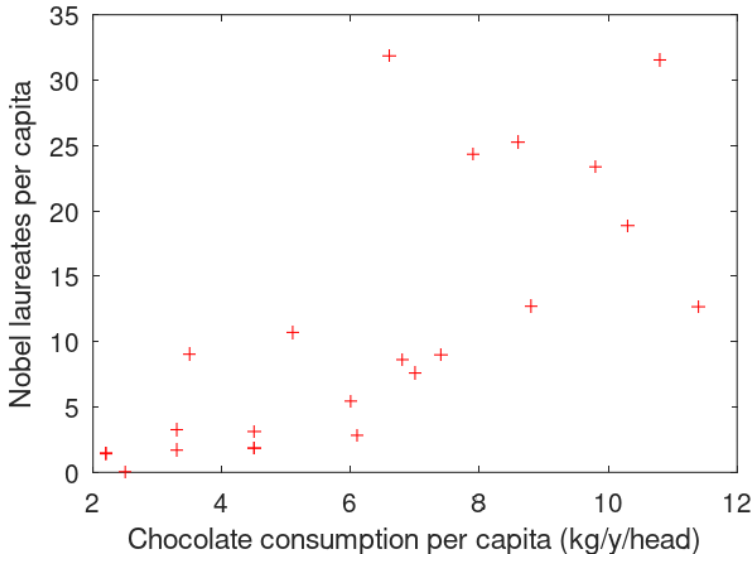
```
% Data loading
table = load("Nobel_vs_choco_CAPS05_assignment.txt") % Load data from .txt file
nobel = table(:,1) % column 1 is populated with data of nobel laureates
choco = table(:,2) % column 2 is populated with data of chocolate consumption
```

We store the number of nobel laureates per capita in `nobel` and the amount of chocolate consumption in `choco`. This will make it possible for us to use them separately as axes in a graph.

Plotting the data set

Since the relation between the two variables cannot be described by a perfect line or curve, it's better to use a scatterplot. I will use plus signs to indicate the data points.

```
% Scatterplot
plot(choco, nobel,'+','markersize',5,"color","r") % plot a scatterplot of the relation
xlabel("Chocolate consumption per capita (kg/y/head)") % Label the axes
ylabel("Nobel laureates per capita")
```



We are now ready to fit a line to the data points.

Fitting a line

```
% Line fitting
X = ones(length(table),2); % Initialize the X array
X(:,1) = choco; % fill the first column of X with our x-axis values
```

To initialize the array `X`, we make an array of ones with 2 columns and rows the same amount as the array `table` has. Then we put in the values in the array `choco` and put it in the first column of `X`.

```
y = nobel; % fill the y array with our y-axis values
```

For y, we fill it with the values in the array `nobel`.

The equation from earlier, $\hat{p} = X^{\dagger}y$ can be translated into code:

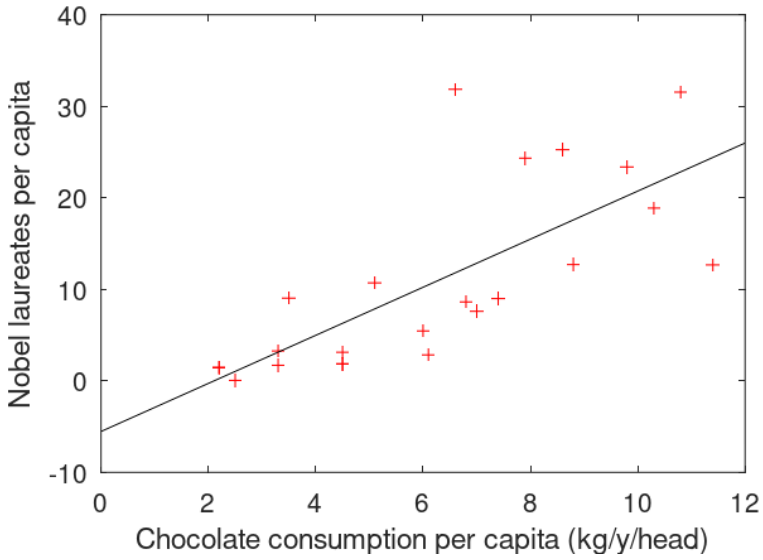
```
p = pinv(X)*y; % Define p as the pseudoinverse of X, times y.
```

Now we can plot the line with the values we have in `p`.

```
% Plot the fitted line
hold on
xx = 0:1:12; % we need to define x values for the straight line to iterate through
plot(xx,p(1)*xx+p(2),'b',"color","k") % plot y = ax+b against the xx we just defined
```

Keep in mind that we are just plotting $y = ax + b$, we just replace x with xx, a with p(1) and b with p(2).

The fitted line looks like this:



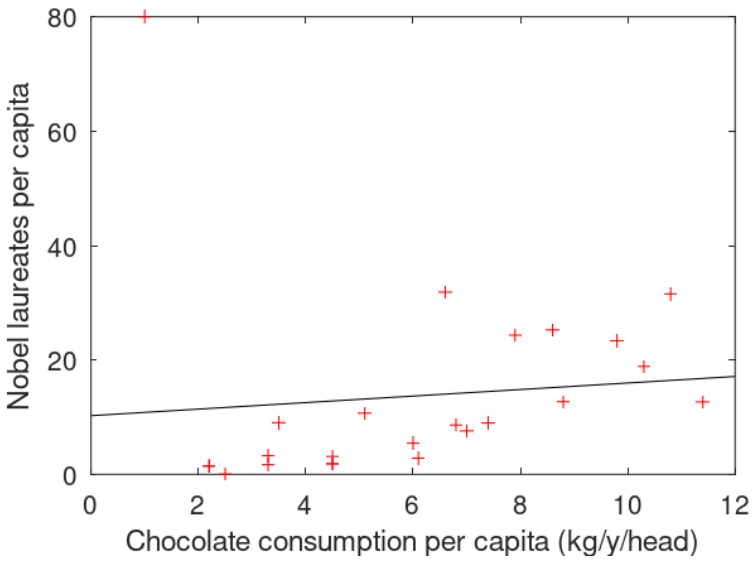
Problem 2

In the next problem, we are to add another data point, belonging to a fictional CAPS Kingdom. The people in CAPS Kingdom has $10 \times (1 + 7) = 80$ nobel laureates per capita and consumes $0.5 \times (0 + 2) = 1$ kg of chocolate per year per person.

Adding this new data point into the existing set is simple enough; we can just add a new row to the array called `table` containing the new values:

```
% Data loading
table = load("Nobel_vs_choco_CAPS05_assignment.txt"); % Load data from .txt file
table = [table; 80, 1]; % Add new data point for CAPS Kingdom to the dataset
nobel = table(:,1); % column 1 is populated with data of nobel laureates
choco = table(:,2); % column 2 is populated with data of chocolate consumption
```

The rest of the script stays the same. Running this script reveals the altered fitted line:



To more clearly show the difference between the two data sets, we can display both lines in a single plot window. To do this, we can just run the scripts one after the other, but a cleaner way would be to hold the first line in view, then add the last dataset before displaying the altered fitted line.

To make it easier for us to fit different datasets, we should define a function that takes data of countries with their respective nobel laureates and chocolate consumption, and outputs the array $\hat{p}$ containing the values a and b, as explained earlier.

```
% Line fitting
function fittedLine = lineFitter(table,choco,nobel)
  X = ones(length(table),2); % Initialize the X array
  X(:,1) = choco; % fill the first column of X with our x-axis values
  y = nobel; % fill the y array with our y-axis values
  fittedLine = pinv(X)*y; % Define fittedLine as the pseudoinverse of X, times y.
end
```

Now everytime we call `lineFitter(table,choco,nobel)`, we pass in our dataset and get `p` as an output.

To add the new data point, we access the existing array and add a new row to it containing our data, which is `80, 1`:

```
% Add new data point to the existing set
table = [table; 80, 1]; % Add new data point for CAPS Kingdom to the dataset
nobel = table(:,1);
choco = table(:,2);
```

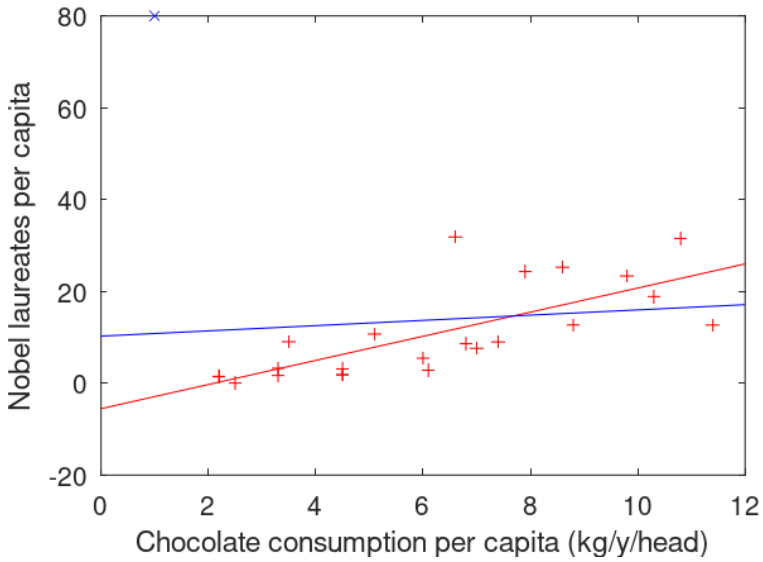
We also want to add to our scatterplot with the new data point. To make it stand out, we make the marker for the point different from the others:

```
% Update the scatterplot with new data
% Mark the new dataset with a blue cross so it's more visible
plot(choco(end),nobel(end),'x','markersize',4,"color","b")
```

And finally, we fit a new line to the updated dataset. Recall that we have `hold` on, which means we won't lose any of our previously plotted points or lines, so we can properly see the difference.

```
% Fit the new line
p = lineFitter(table,choco,nobel)
xx = 0:1:12;
plot(xx,p(1)*xx+p(2),'b',"color","b")
```

Running the completed code of `CAPS_05_C2TB1702_comparison.m` produces this plot:



with the new line in blue and the old one in red. Now we can clearly see that after adding in the new data point, the fitted line significantly becomes less steep. This is because the people in CAPS Kingdom have a considerable amount of nobel laureates, while also consuming very little chocolate. It's visually confirmed by the new data point being very high in the graph while also being very close to the left in chocolate consumption. This tips the balance of the data significantly, as it is an outlier, i.e., it does not fit well at all into the pattern created by the other data points. This is why it makes such a difference even though it's only one data point.

Footnotes - what I've learned

1. The least square method is a very useful way to fit a line into scattered data. However, it should be noted that some data might be deceiving, i.e., it does not always show causation, or even correlation. Some things can be a statistical coincidence, or a logical fallacy. A statistical relation does not always mean causation, and it is very easy to lie or fall prey to lies with data.
2. `hold` is a useful way to keep previous plots in the plot window while adding new things.
3. It's better to define a function once than copy and paste code again and again for doing things repeatedly.