PSBC Project 3

May 10, 2019

1 Introduction

This report details the approach and implementation relating to the second coursework project of MATH36032. Please note that all programming and testing was carried out using GNU Octave rather than MATLAB.

2 Context

The goal of this project was to examine a dataset concerning the sales figures of three products; Bread, Salad and Lettuce, the first few lines of which are included below.

```
Date       MaxTemp     MinTemp     Bread     Salad     Lettuce
1/1/2015        13           8       748       424        1125
2/1/2015        11           2       733       410        1066
3/1/2015        5           -1       744       404        1085
```

Our main aim was to determine whether there exists a correlation between the maximum temperature of a given day, and the amount of each product sold. However, for this analysis, we will only consider dates whose maximum temperature was greater than or equal to ten degrees Celsius. In subsequent sections, we will describe our implementation using the "Bread" dataset for reference. Furthermore, for brevity, we will refer to the maximum temperature simply as the temperature.

3 Preprocessing the data

The first, simplest preprocessing action performed on the data was to avoid working with date strings since this involves working with cell arrays, which are more tiresome to manipulate than vectors and matrices in Octave. Instead, each date d was represented as the number of days between `31/12/2014` and d. For example, `01/01/2015` is represented by the number 1, from now on this is what we refer to as the "dates".

Let us plot a few weeks worth of data and examine the results (shown in Figure 1). We can immediately observe that the data exhibits a periodic trend, with each period being seven days long. Since we are aiming to discover a correlation between temperature and sales, we will need to filter out this periodic effect. For this reason, our analysis will focus heavily on these periods.

To simplify our data analysis, we apply a transformation to each relevant column from the initial data. These columns are the sales data, the maximum temperature and the dates. This

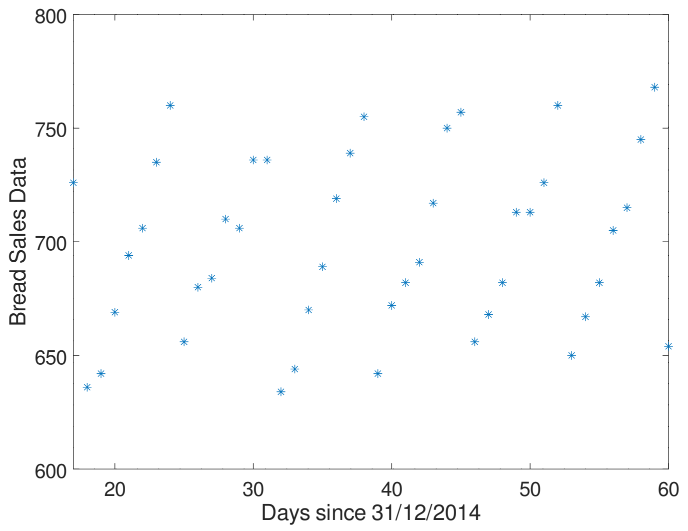

Figure 1: Sales data for bread during a part of 2015.

transform transforms the flat list of values into a matrix whose rows represent periods. Since our periods are seven days long, our matrices each have seven columns.

For example, consider a column vector

$$v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{7m} \end{pmatrix}. \tag{1}$$

Under this transform v becomes

$$\begin{bmatrix} v_1 & v_2 & v_3 & \cdots & v_7 \\ v_8 & v_9 & v_{10} & \cdots & v_{14} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ v_{7m-6} & v_{7m-5} & v_{7m-4} & \cdots & v_{7m} \end{bmatrix}.$$

Of course, this assumes that the number of data entries is a multiple of seven. In order to ensure this is the case, we cut off the leading and trailing data points from the original data set which are not part of a whole period. Listing 1 shows the practical implementation of this idea.

The benefit of this approach will become apparent when we need to perform operations on each period. With the data in this form, it is easy to iterate over rows of the matrix rather than calculate indexes for the periods in the flat data (which becomes tedious when deleting whole periods, something we will need to do shortly).

Listing 1: A function to transform a flat data column into a matrix whose rows represent periods.

```
function [datesPeriods, ...
          salesPeriods, ...
          maxTempPeriods] = preprocessData(data, salesData)

  % Trim the start and end of the data set
  dates = data.Days(4:end-1);
  sales = salesData(4:end-1);
  maxTemp = data.MaxTemp(4:end-1);

  % Transform the data into a matrix who's rows are periods
  datesPeriods = reshape(dates, 7, [])';
  salesPeriods = reshape(sales, 7, [])';
  maxTempPeriods = reshape(maxTemp, 7, [])';

end
```

4 Examining the period slope

The first step towards computing the correlation we seek is to examine the slope of the periods in the data. For this analysis, we will assume that the slope of each period is equal (for each product), though the average in each period may vary. In order to compute the common slope, we will compute the slope of each period and take their average after eliminating any outliers.

For each product, we have a matrix representing its sales data which we will denote P, a result of our preprocessing step. It is defined as follows.

$$v = \begin{pmatrix} \underline{p_1} \\ \underline{p_2} \\ \vdots \\ \underline{p_7} \end{pmatrix}. \tag{2}$$

Where $\underline{p_i}$ is a vector of the seven sales data points for period i. Similarly, we have a matrix D, where each row $\underline{d_i}$ contains date values for the period i. Least square fitting was used to compute the slope of the data for each period. However, there are certain days throughout the dataset whose sales figures are zero; these must be omitted from our calculations. For ease of notation let $p = [p_1 \dots p_k]$ be the product sales data and $d = [d_1 \dots d_k]$ the corresponding dates for an arbitrary period after dates whose sales data is zero have been removed (so $k \leq 7$). We can now use the following two equations to compute the slope m and intercept c of a line of best fit for the period data.

$$m\sum_{j=1}^{k} d_j^2 + c\sum_{j=1}^{k} d_j = m\sum_{j=1}^{k} d_j \times p_j$$

$$m\sum_{j=1}^{k} d_j + c\sum_{j=1}^{k} 1 = m\sum_{j=1}^{k} p_j$$

These equations are derived from equations described by S.J Miller [**?**]. Notice that to compute of m and c we can express the above equation in matrix form as follows. The system can now be solved easily in Octave.

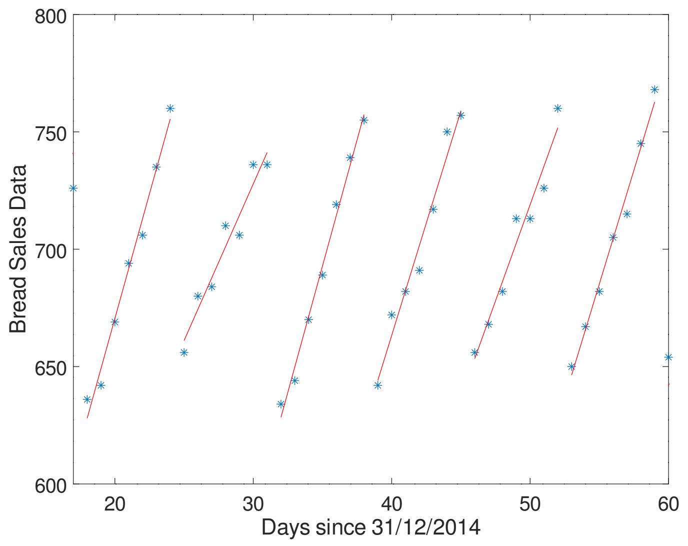

Figure 2: Sales data for bread during a part of 2015 showing the lines of best fit.

$$\begin{bmatrix} \sum_{j=1}^{7} d_j^2 & \sum_{j=1}^{7} d_j \\ \sum_{j=1}^{7} d_j & \sum_{j=1}^{7} 1 \end{bmatrix} \begin{bmatrix} m \\ c \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{7} d_j \times p_j \\ \sum_{j=1}^{7} p_j \end{bmatrix}.$$

Figure 2 shows the results of this computation for the Bread data, performed on the first few periods which shows promising results for estimating the slopes. The implementation of this idea is included in Listings 2 and 3.

Listing 2: A function to compute the slope and intercept for a given period.

```
function [slope, intercept] = slopeAndIntercept(x, y)

  % Solve the system of equations for least square fitting
  A = [sum(x.^2) sum(x);...
       sum(x)     numel(x)];
  lambda = [sum(x .* y); sum(y)];

  solved = A \ lambda;
  slope = solved(1);
  intercept = solved(2);
end
```

Listing 3: A function to compute the slopes of each period.

```
function [slopes] = computePeriodSlopes(datesPeriods, salesPeriods)

  for periodIndex = 1:size(datesPeriods, 1)
    datesPeriod = datesPeriods(periodIndex, :);
    salesPeriod = salesPeriods(periodIndex, :);

    % Ignore the zero sales data
    validIndexes = find(salesPeriod > 0);

    [slope, ~] = slopeAndIntercept(datesPeriod(validIndexes), ...
                                           salesPeriod(validIndexes));

    slopes(periodIndex) = slope;
  end

end
```

4.1 Identifying and eliminating outliers

It is clear that some periods have very different slopes to the others. Figure 3 shows all the slopes and reveals evident outliers highlighted in red. To identify these outliers Tukey fences [?] were used. This approach relies on the computation of quartiles in the data, (the data here being the list of slopes).

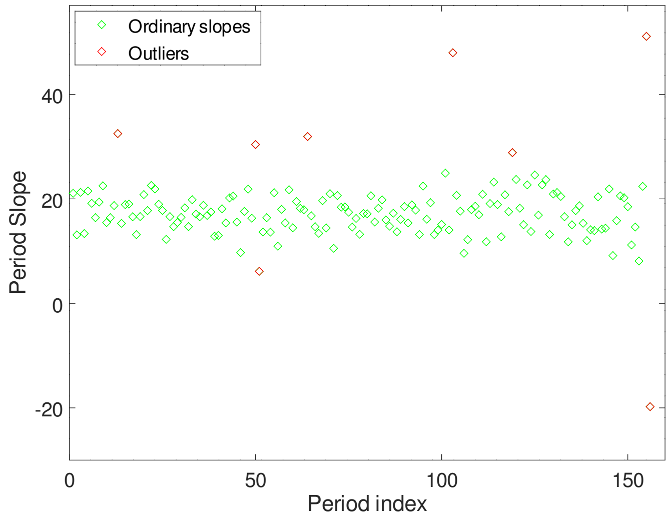

Figure 3: The slopes for each period with the outliers highlighted in red.

This is a reasonably simple method; we work out the first and third quartiles in the data and denote them Q_1 and Q_3 respectively. Denote the interquartile range $(Q_3 - Q_1)$ as IQR and the slope for period i as s_i. Now we define outliers to be all the slopes s_i such that either of the following holds:

$$s_i < Q_1 - 1.5 \times IQR,$$

$$s_i > Q_3 + 1.5 \times IQR.$$

The number 1.5 was a number used initially in this context by John Tukey [?] since it was effective in many applications. In our case, the figure 1.5 works well, evident when we plot the results as shown in Figure 3 (this is similarly effective for the other two products). Outliers are identified correctly. Now that this is done, we can take the mean of the remaining values to obtain the value we need, from here on we will denote this value μ. Listing 4 shows the code used to identify outliers. Note, the preprocessing of the data now makes it very easy to eliminate entire periods of data (those with slopes that are outliers) in the next part of our analysis by removing the corresponding rows of the data matrices which is a simple operation.

Listing 4: A function to identify outliers in a column of data

```
function outlierIndexes = findOutliers(rawData)

  % First calculate the quartiles in the data
  q1 = quantile(rawData, 0.25);
  q3 = quantile(rawData, 0.75);

  % Interquartile range
  iqr = q3 - q1;

  outlierIndexes = union(find(rawData < q1 - (iqr * 1.5)), ...
                         find(rawData > q3 + (iqr * 1.5)));

end
```

5 Computing the correlation coefficient

Now that we have a value for the period slope μ, of a given product we can use it to filter out the periodic effect in order to gain insight into the correlation between temperature and sales. Note that from now on we will be working with data that omits the periods whose slopes are outliers. We will also omit data for days whose temperature is below 10 degrees Celsius to improve results.

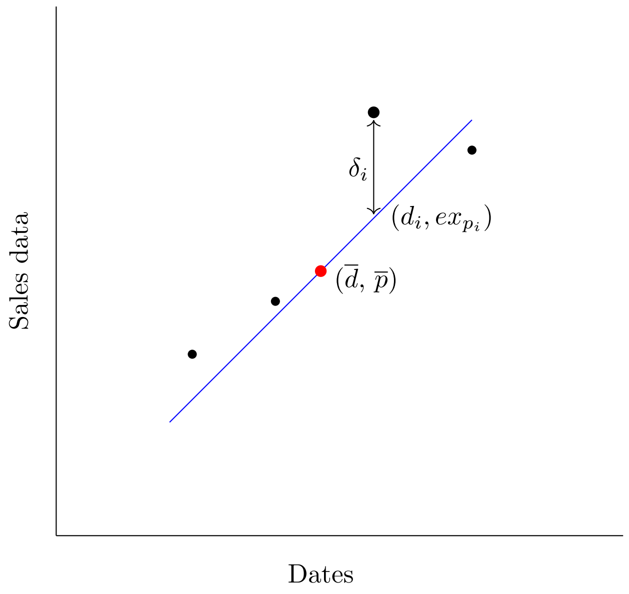

Figure 4: A visualisation of the values used in defining the transformation T.

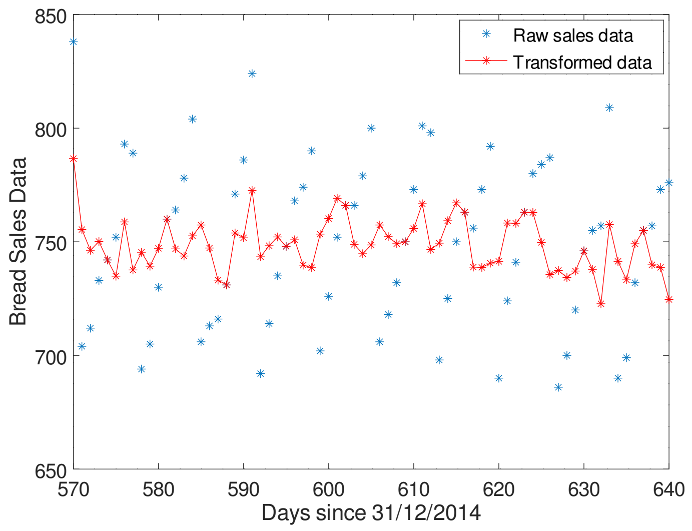

Figure 5: Sales data for bread during a part of 2015 showing the transformed sales data.

Our aim now is to define a transformation T on the sales data such that the transformed data exhibits trends unaffected by the day of the week, i.e. data which is independent of the periodic effect. To achieve this, we make the following assumption. For a given period, the deviation of a specific data point from the mean for that period is dependent on two things: temperature and day of the week.

Let us first define the mean for the period. Here our x-axis is the date, and the y-axis is the sales data. Define the sales data $\underline{p}$ and dates $\underline{d}$ as before. We can represent the mean point by $(\overline{d}, \overline{p})$. We can define a line of expected sales values as passing through this mean point with slope μ. This line is shown in blue in Figure 4. By computing the difference, δ_i between the expected sales value ex_{p_i} at date d_i and the actual value, we obtain the variation based on temperature, not the period. It follows from our assumption that the sum of the mean and these δ_i values will be independent of the period; hence we can define our transform as follows.

$$T(p_i) = \overline{p} + \delta_i.$$

Figure 5 shows the data for the first few periods as well as the transformed data. It is clear that the periodic effect is mostly eliminated. Applying the transformation T to each period and concatenating the results now gives a flat list of transformed values. By flattening the temperature matrix M we now have data of a form that can be processed by the `corrcoef` command to compute a correlation coefficient. Flattening the matrices consists of taking each row and forming a vector by concatenating them (essentially reversing the preprocessing step).

However, Figure 6 shows the results of plotting the transformed data against the temperature data. Clearly, there are two separate correlations, in investigating this, it appears sales data spiked in the third year, a trend not related to the temperature. Computing the correlation coefficient at this point would be ineffective for this reason. However, by splitting the data by year and normalising the data for the three years individually (after removing obvious outliers for each year), we eliminate the effect of the two separate correlations hinted at in Figure 6. With this done we can plot the temperature against the normalised data and see a

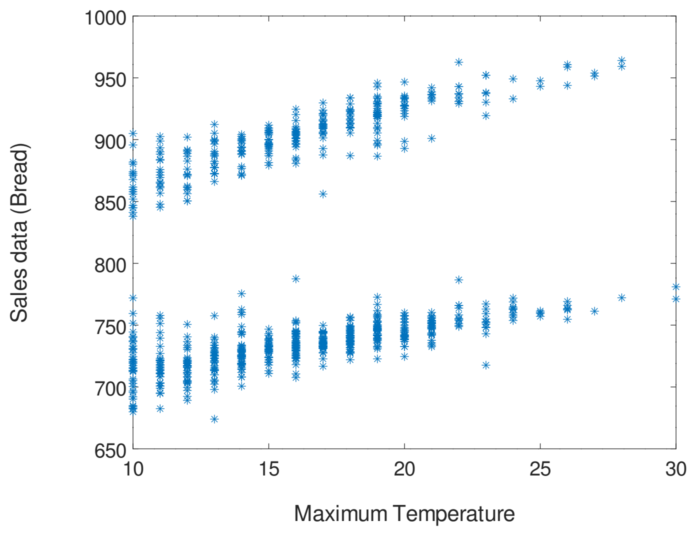

Figure 6: The temperature plotted against the normalised and transformed sales data.

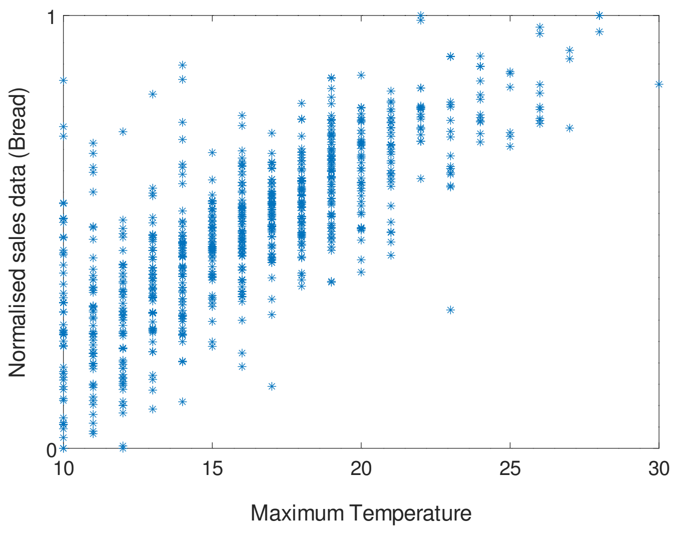

Figure 7: The temperature plotted against the normalised and transformed sales data.

much clearer correlation as shown in Figure 7.

Finally, the `corrcoef` command can be used on the normalised data as shown in Listing 5. The results of our analysis are below (Figure 8) having run the analysis on the data for each product. The glue code which calls these functions for each product is all included in the Listings in Appendix A.

```
>> main
  Bread: Slope = 17.1303; Correlation Coefficient = 0.774382
Lettuce: Slope = 23.0191; Correlation Coefficient = 0.804459
  Salad: Slope = 11.8209; Correlation Coefficient = 0.748916
```

Figure 8: The overall results of the analysis.

Listing 5: A function to calculate the correlation coefficients

```
function coef = computeCorrelationCoef(datesPeriods, salesPeriods, ...
    maxTempPeriods, slopeAvg)
  % Each sales data point transformed based on the slope and the average of
  % its period

  filteredTemps = [];
  filteredDates = [];
  transformedSales = [];

  for periodIndex = 1:size(datesPeriods, 1)

    % Get data for current period
    salesPeriod = salesPeriods(periodIndex, :);
    tempPeriod = maxTempPeriods(periodIndex, :);
    validIndexes = intersect(find(salesPeriod > 0), find(tempPeriod >= 10));

    % Nothing to do if none of the data is valid
    if length(validIndexes) == 0
      continue;
    end

    % Filter invalid data
    datesPeriod = datesPeriods(periodIndex, :)(validIndexes);
    salesPeriod = salesPeriod(validIndexes);
    tempPeriod = tempPeriod(validIndexes);

    deltaSalesPlusMean = salesPeriod - slopeAvg .* (datesPeriod - mean(datesPeriod));

    filteredDates = [filteredDates datesPeriod];
    filteredTemps = [filteredTemps tempPeriod];
    transformedSales = [transformedSales deltaSalesPlusMean];

  end

  % Analyse the last year seperately
  firstDayOfYr3 = 732;
  lastDayYr1 = 365;

  % The 1st year data excluding outliers
  indexesYr1 = find(filteredDates <= lastDayYr1);
  [dates1, sales1, temp1] = removeOutliers(indexesYr1, filteredDates, ...
      transformedSales, filteredTemps);

  % The 2rd year data excluding outliers
  indexesYr2 = intersect(find(filteredDates > lastDayYr1), find(filteredDates < ...
      firstDayOfYr3));
  [dates2, sales2, temp2] = removeOutliers(indexesYr2, filteredDates, ...
      transformedSales, filteredTemps);

  % The 3rd year data excluding outliers
  indexesYr3 = find(filteredDates >= firstDayOfYr3);
  [dates3, sales3, temp3] = removeOutliers(indexesYr3, filteredDates, ...
      transformedSales, filteredTemps);

  % Normalise the data before putting it back together
  normalise = @(data) (data - min(data)) / (max(data) - min(data));

  sales1 = normalise(sales1);
  sales2 = normalise(sales2);
  sales3 = normalise(sales3);

  % Compute coefficient
  coef = corrcoef([temp1, temp2, temp3], [sales1, sales2, sales3])(1,2);

endfunction
```

Appendix A Code

Listing 6: The code that runs the analysis for the seperate products and prints results.

```
data = readSalesFigures();

printRes = @(type, slope, coef) printf("%7s: Slope = %d; Correlation Coefficient ...
    = %d\n", type, slope, coef);

[slope, coef] = runAnalysis(data, data.Bread);
printRes("Bread", slope, coef);

[slope, coef] = runAnalysis(data, data.Lettuce);
printRes("Lettuce", slope, coef);

[slope, coef] = runAnalysis(data, data.Salad);
printRes("Salad", slope, coef);
```

Listing 7: A function to run the analysis on the data for a single product.

```
function [slope, coef] = runAnalysis(data, salesData)

  % Turn data into matricies who's rows are periods
  [datesPeriods, salesPeriods, maxTempPeriods] = preprocessData(data, salesData);

  periodSlopes = computePeriodSlopes(datesPeriods, salesPeriods);

  % Remove periods who's slopes are outliers
  outlierIndexes = findOutliers(periodSlopes);

  datesPeriods(outlierIndexes, :) = [];
  salesPeriods(outlierIndexes, :) = [];
  maxTempPeriods(outlierIndexes, :) = [];
  periodSlopes(outlierIndexes) = [];

  slope = mean(periodSlopes);
  coef = computeCorrelationCoef(datesPeriods, salesPeriods, maxTempPeriods, slope);
endfunction
```

Listing 8: A function to read the data from the csv file.

```
function figs = readSalesFigures()

  fid = fopen('salesfig.csv', 'r'); % open the file

  data = textscan ( fid , '%s,%f,%f,%f,%f,%f', -1,'Delimiter',',');
  fclose (fid);

  figs = struct();

  figs.MaxTemp = cell2mat(data(2));
  figs.MinTemp = cell2mat(data(3));
  figs.Bread   = cell2mat(data(4));
  figs.Salad   = cell2mat(data(5));
  figs.Lettuce = cell2mat(data(6));
  figs.Days = [1:length(figs.Lettuce)];

end
```

Listing 9: A function to remove outliers from the data.

```
function [dates, sales, temps] = removeOutliers(indexes,...,
                                                filteredDates, ...
                                                transformedSales, ...
                                                filteredTemps)

  sales = transformedSales(indexes);
  dates = filteredDates(indexes);
  temps = filteredTemps(indexes);

  outliers = findOutliers(sales);
  sales(outliers) = [];
  dates(outliers) = [];
  temps(outliers) = [];
end
```

References

[1] Staven J. Miller. *The Method of Least Squares*. `https://web.williams.edu/Mathematics/sjmiller/public_html/BrownClasses/54/handouts/MethodLeastSquares.pdf`. Accessed: 09-05-2019.

[2] Songwon Seo. *A Review and Comparison of Methods for Detecting Outliers in Univariate Data Sets*. `http://d-scholarship.pitt.edu/7948/1/Seo.pdf`. Accessed: 09-05-2019.