Alan Agresti and Maria Kateri

MATLAB-Web-Appendix of Foundations of Statistics for Data Scientists

Working draft to be expanded

Contents

0

CHAPTER 0: BASICS OF MATLAB

This Appendix provides implementation in MATLAB of some of the examples that are worked out in the chapters of this book in R.[1] We use the version MATLAB.R2021b.

MATLAB has been designed for engineers and scientists to analyze and visualize data, develop algorithms, and create models and applications. Basic functions for data visualization and statistical data analysis are included in the main MATLAB software while the statistical analysis tools have been enriched in the Statistics and Machine Learning Toolbox (https://mathworks.com/products/statistics.html) and the Simulink environment (https://mathworks.com/products/simulink.html). A familiarity with MATLAB is assumed. The MATLAB support (https://mathworks.com/support.html) and the help center of the Statistics and Machine Learning Toolbox (https://mathworks.com/help/stats/) offer a great help with overview and documentation of available functions, examples and tutorials. Furthermore, there is a variety of introductions to MATLAB language available, e.g., Martinez and Cho (2015, 2016), Metcalfe et. al. (2019) and Rogers and Girolami (2017). The MATLAB support provides a rich list of books on Data Science and Statistics with implementation in MATLAB (https://mathworks.com/academia/books.html).

C0.1 MATLAB Preliminaries

MATLAB has a very user-friendly desktop environment that includes the menu bar with three tabs (HOME, PLOTS, APS) and the following main windows: Command Window, Current Folder, and Workspace (see Figure C0.1). Commands are entered in the Command Window after the MATLAB prompt >>. All variables that are in the current workspace are listed in the Workspace window. Double-clicking one of them opens it a spreadsheet in the Variables window. The Current Folder shows all the files in the folder given in the red-framed area in Figure C0.1. The current folder can be selected by the Browse for folder on its left. Double-clicking on a data file in the Current Folder window activates for this file the Import Data button of the HOME tab (red dashed framed in Figure C0.1), to be discussed in Section C0.2.

The most straight-forward way to get help and access the documentation of functions in MATLAB is through the Help button (green dashed framed in Figure C0.1). Typing help in the Command Window followed by the name of a function, provides brief information on this function, as shown in Figure C0.2 for the bar function that constructs a bar chart.

Comments in MATLAB follow the % symbol while, as in R, a command followed by a ";" does not print the result of the assignment or calculation. The code to be typed in the Command Window for a simple example defining a row vector and dividing its entries by a scalar (here 2) follows.

```
>> % create a row vector without printing it:
```

[1]This is working draft that will be gradually expanded.

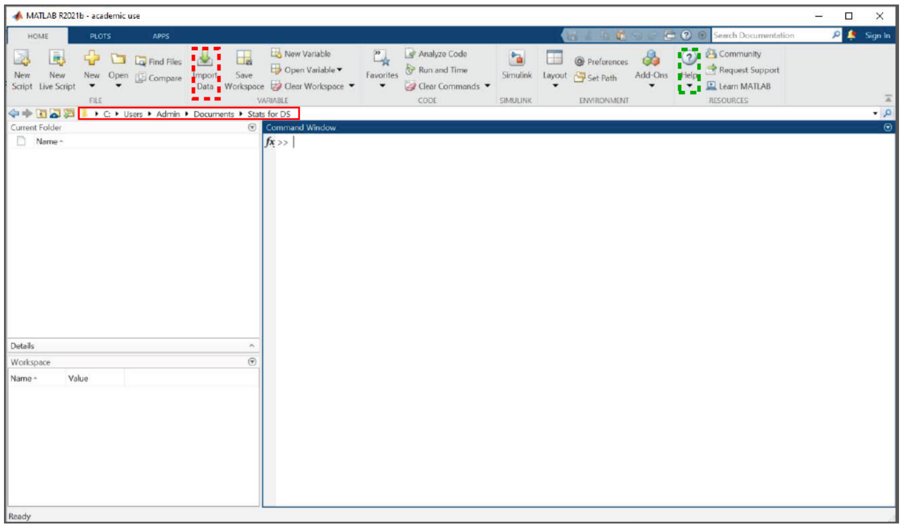


FIGURE C0.1: MATLAB–desktop screen–shot.

```
>> a=[1 2 3 4];  % equivalent to a=[1,2,3,4]; or a=1:4;

>> b=a/2         % devide all elements of a by 2
b =
    0.5000    1.0000    1.5000    2.0000
```

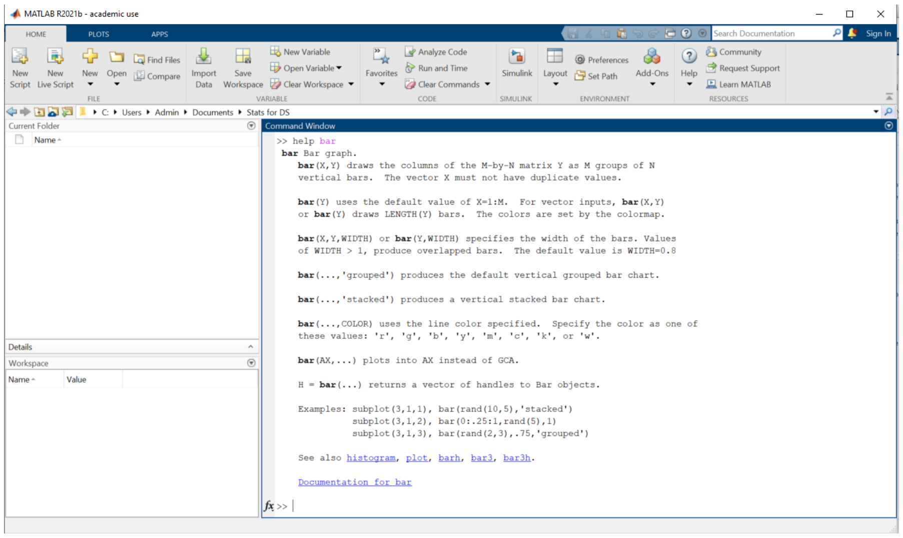
The code is shown in this appendix as it appears in the Command Window, with the MATLAB prompt, which of course has to be omitted when typing the code.

The standard arithmetic operators (+, -, *) are used in MATLAB. Similarly to R, calculations apply on matrices. A column vector is constructed by separating its elements by semicolons, as shown below.

```
>> c=[1;2;3;4]    % c is equal to transpose(a)
c =
     1
     2
     3
     4

>> (a*c)^2        % product of (1x4) and (4x1) vectors to the power 2
ans =
    900

>> m1=c*a         % product of (4x1) and (1x4) vectors
m1 =
     1     2     3     4
     2     4     6     8
     3     6     9    12
     4     8    12    16
```



```
>> help bar
 bar Bar graph.
    bar(X,Y) draws the columns of the M-by-N matrix Y as M groups of N
    vertical bars.  The vector X must not have duplicate values.

    bar(Y) uses the default value of X=1:M.  For vector inputs, bar(X,Y)
    or bar(Y) draws LENGTH(Y) bars.  The colors are set by the colormap.

    bar(X,Y,WIDTH) or bar(Y,WIDTH) specifies the width of the bars. Values
    of WIDTH > 1, produce overlapped bars.  The default value is WIDTH=0.8

    bar(...,'grouped') produces the default vertical grouped bar chart.

    bar(...,'stacked') produces a vertical stacked bar chart.

    bar(...,COLOR) uses the line color specified.  Specify the color as one of
    these values: 'r', 'g', 'b', 'y', 'm', 'c', 'k', or 'w'.

    bar(AX,...) plots into AX instead of GCA.

    H = bar(...) returns a vector of handles to Bar objects.

    Examples: subplot(3,1,1), bar(rand(10,5),'stacked')
              subplot(3,1,2), bar(0:.25:1,rand(5),1)
              subplot(3,1,3), bar(rand(2,3),.75,'grouped')

    See also histogram, plot, barh, bar3, bar3h.

    Documentation for bar

fx >> |
```

FIGURE C0.2: Output of the `help bar` command in the `Command Window`.

C0.2 Data Structures and Data Input

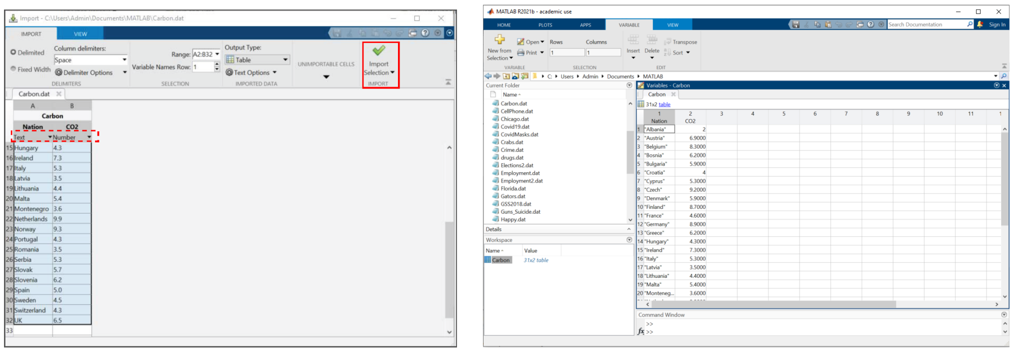
Analogously to `R`, `MATLAB` offers a variety of data types:

- *Arrays* for grouping objects of the same type (e.g., numeric, string, or categorical). Arrays can be scalars, vectors, matrices (i.e., two-dimensional arrays) or multi-dimensional arrays.
- *Cell Arrays* for grouping objects of different type. Different cells can have data of different type and different size but all rows of a cell array have the same number of cells. Otherwise, a cell array has the structure of a numeric or string array.
- *Structures* are arrays with named fields that can contain data of varying types and sizes.
- *Tables* are arrays in tabular form with named columns which can be of different type.
- *Dataset Arrays* for storing variables (in columns) with heterogeneous data types (e.g., combine numeric data, logical data, cell arrays of character vectors, and categorical arrays) measured over cases (given in rows). They are convenient for data sets and statistical data analysis (analogous to the *data frames* of `R`).

The first four are data types of the base `MATLAB` while Dataset Arrays is of the `Statistics and Machine Learning Toolbox`. For a detailed description of the powerful options of the data types and examples, see in https://mathworks.com/support.html.

Data files of various formats (text files, spreadsheets and other file formats) are easily read in `MATLAB`. Commonly, data files for statistical analysis are in spreadsheets (e.g., of `R` data frame type with rows corresponding to cases and columns to variables measured). Such files (`R .dat` files, .txt data files, excel .xls files, SAS data files) can be easily read through the *Import Tool* activated by clicking the `Import Data` button of the `HOME` tab (red dashed framed in Figure C0.1). The tool asks for the data file to be opened and the selected file appears in the `Import` window. Figure C0.3 shows the `Import window` for the `Carbon.dat` data file on per-capita carbon dioxide emissions for 31 European nations analyzed in Chapter 1 (to be downloaded from the book's webpage).

After selecting the variables and cases that should be read, and setting the type of each variable from the types offered (see dashed lined red frame in Figure C0.3, left), the data are imported in the workspace by selecting `Import Data` in the `Import Selection` button (see red frame in Figure C0.3, left). Clicking on `Carbon` in the *Workspace* window, the `Variables` window opens and the data spreadsheet is shown (see Figure C0.3, right).



	A	B
	Carbon	
	Nation	CO2
	Text	Number
15	Hungary	4.3
16	Ireland	7.3
17	Italy	5.3
18	Latvia	3.5
19	Lithuania	4.4
20	Malta	5.4
21	Montenegro	3.6
22	Netherlands	9.9
23	Norway	9.3
24	Portugal	4.3
25	Romania	3.5
26	Serbia	5.3
27	Slovak	5.7
28	Slovenia	6.2
29	Spain	5.0
30	Sweden	4.5
31	Switzerland	4.3
32	UK	6.5
33		

Name	Value
Carbon	31x2 table

Carbon

31x2 table

	1	2
	Nation	CO2
1	"Albania"	2
2	"Austria"	6.9000
3	"Belgium"	8.3000
4	"Bosnia"	6.2000
5	"Bulgaria"	5.9000
6	"Croatia"	4
7	"Cyprus"	5.3000
8	"Czech"	9.2000
9	"Denmark"	5.9000
10	"Finland"	8.7000
11	"France"	4.6000
12	"Germany"	8.9000
13	"Greece"	6.2000
14	"Hungary"	4.3000
15	"Ireland"	7.3000
16	"Italy"	5.3000
17	"Latvia"	3.5000
18	"Lithuania"	4.4000
19	"Malta"	5.4000
20	"Monteneg...	3.6000

```
>>
fx >>
```

FIGURE C0.3: `Import` window (left) and `Variables` window (right) for `Carbon.dat` data set.

1

CHAPTER 1: MATLAB FOR DESCRIPTIVE STATISTICS

C1.1 Random Number Generation

The example of random number generation in Section 1.3.1 can be implemented in `MATLAB` using the `randsample` function of the `Statistics and Machine Learning Toolbox`. The content of the `Command Window` for the example follows.

```
>> y = randsample(1:60,5,'false') % sample 5 integrers from 1 to 60 without repacement
y =
    43     6     3     2    17

>> y = randsample(1:60,5,'true') % sample with repacement
y =
    58    33     9     9    16
```

Examples of random number generation from various disrtibutions in `MATLAB` are shown in Sections C1.5, C2.1 and C2.3.

C1.2 Summary Statistics and Graphs for Quantitative Variables

C1.2.1 Descriptive statistics for carbon dioxide emissions

The following code finds some descriptive statistics for the data in file `Carbon.dat`, imported in the Workspace in Section C0.2.

```
>> summary = groupsummary(Carbon,[],{"mean","std","min","max"},"CO2")
summary =
  1×5 table

    GroupCount    mean_CO2    std_CO2    min_CO2    max_CO2
    __________    ________    _______    _______    _______

        31         5.8194     1.9649        2         9.9
>> % alternatively, you can try:
>> [mean,std,min,max] = grpstats(Carbon.CO2,[],{"mean","std","min","max"}) % output not shown
>> summary = groupsummary(Carbon,[],"all","CO2") % all available statistics (output not shown)
>> Q = quantile(Carbon.CO2, [0.25,0.5,0.75])     % computes the quantiles of CO2
Q =
    4.3250    5.4000    6.8000
```

The histogram of CO2 with 8 bins of equal length can be derived in `Python` (see Figure C1.1), as shown below:

```
>> histogram(Carbon.CO2, 8)
xlabel('CO2 emissions')
ylabel('Count')
```

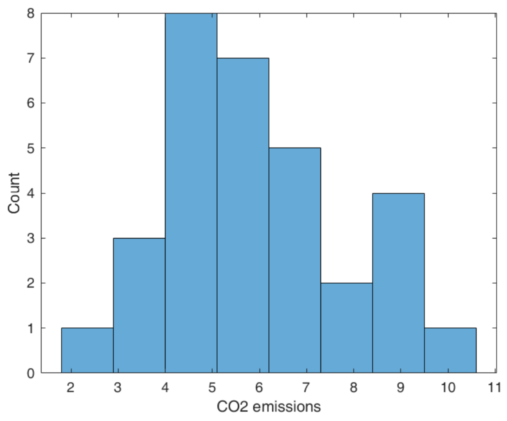


FIGURE C1.1: Histogram for frequency distribution of European CO2 values.

The box plot of CO2 (see Figure C1.2) is produced as follows.

```
>>  boxplot(Carbon.CO2)
xlabel('31 European nations')
ylabel('CO2 emissions')
```

Replacing the `boxplot` function call above with `boxplot(Carbon.CO2, 'Whisker',1)` produces a boc plot with the whisker length specified as 1.0 times the interquartile range. Data points beyond the whisker are displayed with +.

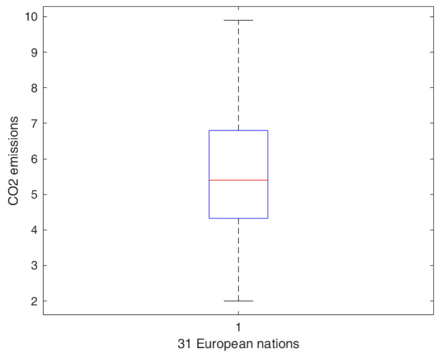


FIGURE C1.2: Box plot of CO2 emission values for European nations.

C1.2.2 Side-by-side box plots for U.S. and Canadian murder rates

Side-by-side box plots are illustrated in Section 1.4.5 for comparing the murder rates in U.S. and Canada. After importing the data in the `Workspace`, the code that follows constructs the side-by-side box plot for this example in `MATLAB`, shown in Figure C1.3.

In order to illustrate data import in `MATLAB` from .txt files separating the columns by a "," (instead of a tab), we read the data from the Murder2.txt file (to be downloaded from the book's webpage). Such files are not opened in a spreadsheet when double clicking on them in the `Current Folder` window but need to be imported by the `Import Data` button.

```
>> boxplot(Murder2.murder,Murder2.nation)
xlabel('Nation')
ylabel('Murder rates')
```

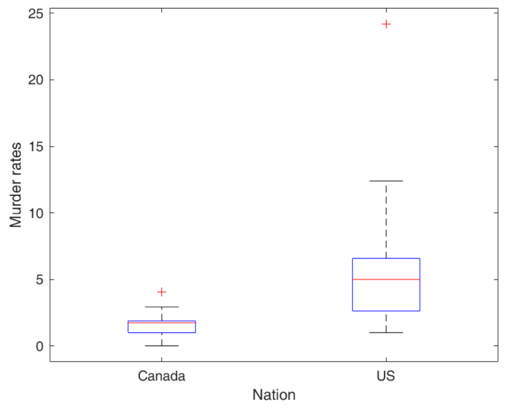


FIGURE C1.3: Side-by-side box plots for U.S. and Canadian murder rates.

For computing the quantiles per nation we need to define the functions `quant1`, `quant2` and `quant3` in `MATLAB` code files (.m). Indicatively we provide below one of them:

```
function [q1] = quant1(m)
    q1=quantile(m,0.25);
end
```

The code given below reports summary statistics by nation:

```
>> summary2 = groupsummary(Murder2,{"nation"},{"mean","std","min","max"},"murder")
summary2 =
  2×6 table

    nation    GroupCount    mean_murder    std_murder    min_murder    max_murder
    ______    __________    ___________    __________    __________    __________

    Canada        10           1.673         1.1844          0             4.07
    US            51          5.2529         3.7254          1             24.2

>> [Group,Nation] = findgroups(Murder2.nation);
>> Group = int8(Group);
>> Q1 = splitapply(@quant1,Murder2.murder,Group);  % functions quant1, quant2 and quant3
>> Q2 = splitapply(@quant2,Murder2.murder,Group);  % are defined in MATLAB-code files (.m)
>> Q3 = splitapply(@quant3,Murder2.murder,Group);  % as explained above
>> T=table(Nation,Q1,Q2,Q3)
T =
  2×4 table

    Nation     Q1       Q2       Q3
    ______    _____    _____    _____

    Canada     0.99    1.735     1.88
    US        2.625        5    6.575
```

C1.3 Descriptive Statistics for Bivariate Quantitative Data

Fr the example in Section 1.5.1 relating statewide suicide rates in U.S. to the percentage of people who own guns, we show next code to construct the scatter plot in Figure **??** (left) as well as the scatter plot with the fitted regression line (see Figure **??**, right). The associated Pearson's correlation coefficient and the simple linear regression model fit are also derived. The data file `Guns_Suicide.dat` is imported in the `MATLAB Workspace`, following the procedure described in Section C0.2 for the `Carbon.dat` data file, and saved under `GunsSuicide`.

```
% print just the first 3 lines of the data file:
>> head(GunsSuicide,3)
ans =
  3×3 table

    state    guns    suicide
    -----    ----    -------

    "AK"     60.6     23.3
    "AL"     57.2     14.9
    "AR"     58.3     17.4

% scatterplot:
>> scatter(GunsSuicide,'guns','suicide')
% addition of the regression line:
>> reg1=lsline; reg1.Color=[0 0.4470 0.7410];reg1.LineWidth=1.5;

>> Guns = GunsSuicide.guns; Suicide = GunsSuicide.suicide;
>> R=corrcoef(Guns,Suicide)      # correlation matrix between
                                 # variables Guns and Suicide
R =
    1.0000    0.7387             # corr. = 0.739 between guns and suicide
    0.7387    1.0000

% Linear regression fit:
>> model1 = fitlm(Guns,Suicide)

model1 =
Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
                   Estimate       SE       tStat       pValue
                   --------    --------    ------    ----------

    (Intercept)     7.3901       1.0372    7.1253    4.2432e-09
    x1             0.19356     0.025234    7.6708    6.1083e-10

Number of observations: 51, Error degrees of freedom: 49
Root Mean Squared Error: 2.65
R-squared: 0.546,  Adjusted R-Squared: 0.536
F-statistic vs. constant model: 58.8, p-value = 6.11e-10
```

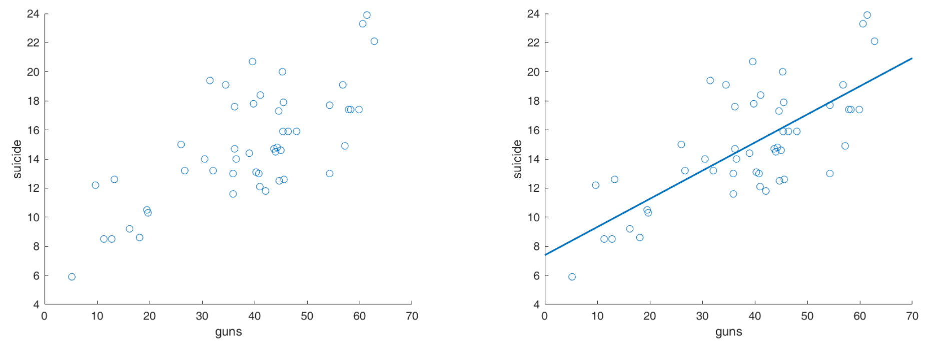


FIGURE C1.4: Scatterplot relating state-level data in the U.S. on percent gun ownership and suicide rate

C1.4 Descriptive Statistics for Bivariate Categorical Data

We next show code for forming a contingency table in `MATLAB` for the example in Section 1.5.2 of cross-classifying race and political party identification for data from the 2018 General Subject Survey, after importing the data in the `MATLAB Workspace`. We also compute the Chi-squared test of independence (see Section 5.4.4) and the associated p-value.

```
>> [conttab,chi2,p,labels] = crosstab(PartyID.race,PartyID.id)

conttab =
   281    65    30
   124    77    52
   633   272   704

chi2 =
  233.0352

p =
   2.9320e-49

labels =
  3×2 cell array
    {'black'}    {'Democrat' }
    {'other'}    {'Independe'}
    {'white'}    {'Republica'}

>> colmarg = sum(conttab,1)        % find column marginals

ans =
        1038         414         786
% derivation of joint probability table:
>> probtable = conttab/sum(colmarg)

probtable =
    0.1256    0.0290    0.0134
    0.0554    0.0344    0.0232
    0.2828    0.1215    0.3146

# derivation of conditional column probabilities (within column) tables:
>> conttab./repmat(sum(conttab,1),[3 1])
```

```
ans =
    0.2707    0.1570    0.0382
    0.1195    0.1860    0.0662
    0.6098    0.6570    0.8957
```

There is no direct option in MATLAB to construct a mosaic plot.

C1.5 Simulating Samples from a Bell-Shaped Population

The simulation example in Section 1.5.3 took two random samples of size $n = 30$ each from a bell-shaped population (specifically, the normal distribution introduced in Section 2.5.1) with a mean of 100 and a standard deviation of 16. The following code performs the simulation, saves the simulated numbers in a column vector, finds sample means and standard deviations, and constructs histograms:

```
% simulates a column vector (30x1) from N(100, 16\^2):
>> y1 = normrnd(100,16,30,1); mean(y1)
ans =
  102.4127

>> std(y1)
ans =
   15.5943

>> hist(y1)
```

You may simulate other samples and compare the sample means, the standard deviations and the histograms that are derived each time. Repeat the procedure simulating samples of larger size, say 200.

This example illustrates that descriptive statistics such as the sample mean can themselves be regarded as variables, their values varying from sample to sample. Chapter 3 provides results about the nature of that variation.

2

CHAPTER 2: MATLAB FOR PROBABILITY DISTRIBUTIONS

C2.1 Simulating a Probability as a Long-Run Relative Frequency

We illustrate the definition of the probability of an outcome as the long-run relative frequency of that outcome in n observations, with n taking values 100, 1000, 10000, 100000, 1000000, and with probability 0.20 for each observation:

```
% proportion of "heads" in 100, 1000, 10000, 100000, 1000000 flips:
>> x1 = binornd(100,0.2); x1/100
ans =
    0.1700

>> x2 = binornd(1000,0.2); x2/1000
ans =
    0.1920

>> x3 = binornd(10000,0.2); x3/10000
ans =
    0.1993

>> x4 = binornd(100000,0.2); x4/100000
ans =
    0.1994

>> x5 = binornd(100000,0.2); x5/100000
ans =
    0.1996
```

C2.2 MATLAB Functions for Discrete Probability Distributions

Discrete probability distributions that are directly supported in MATLAB can be found in https://uk.mathworks.com/help/stats/discrete-distributions.html. We shall illustrate here for the binomial, Poisson and multinomial.

C2.2.1 Binomial Distribution

A binomial probability distribution object `BinomialDistribution` can be generated either by fitting the binomial distribution to a specific data set or by specifying its parameter values. Such an object has several options, including calculation of *pmf* or *cdf* values and random number generation. The following shows code for computing binomial probabilities and plotting a binomial *pmf* for the example in Section 2.4.2 about the Hispanic composition of a jury list, which has $n = 12$ and $\pi = 0.20$. The plot is provided in Figure C2.1.

```
>> binopdf(1,12,0.2)    % binomial P(Y=1) when n=12, pi=0.20
ans =
    0.2062
>> binocdf(1,12,0.2)   % binomial P(Y=0)+P(Y=1) when n=12, pi=0.20
ans =
    0.2749
>> y=0:12; p=binopdf(y,12,0.2); stem(y,p)  % alternatively as a bar-chart: bar(y,p)
>> xlabel('y');  ylabel('P(Y=y)')
```

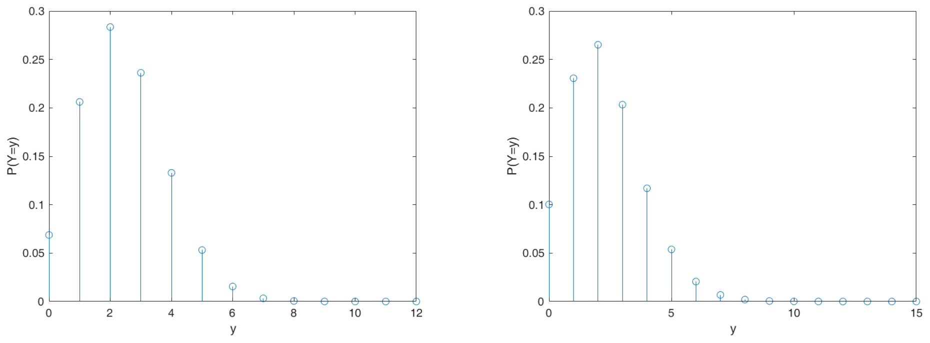


FIGURE C2.1: Probability mass function of (i) a binomial random variable with $n = 12$ and $\pi = 0.2$ (left) and a Poisson random variable with $\lambda = 2.3$.

C2.2.2 Poisson Distribution

For a Poisson distribution, probabilities of individual values using the *pmf* or of a range of values using the *cdf*, such as in the example in Section 2.4.7, can be found as follows:

```
>> poisspdf(0,2.3)  % P(Y=0) if Poisson mean = 2.3
ans =
    0.1003

% Difference of cdf values at 130 and 69 for Poisson with mean = 100:
>> poisscdf(130,100) - poisscdf(69,100)
ans =
    0.9976

% Probability within 2 stand. deviations of mean (from 80 to 120):
>> poisscdf(120,100) - poisscdf(80,100)
ans =
    0.9547
% Plot of the pmf of a Poisson(2.3), provided in Figure C2.1:
>> y = 0:15; p = poisspdf(y, 2.3)
>> stem(y,p); xlabel('y');  ylabel('P(Y=y)')
```

C2.2.3 Multinomial Distribution

Next we provide the MATLAB code for calculating the *pmf* of a multinomial distribution and for plotting it (see Figure C2.2).

```
>> p = [0.3 0.5 0.2];
% generate one random vector mult(1,p):
>> rng('default')  % For reproducibility
>> r = mnrnd(1,p,1)
>> r =
     0     0     1
% generate 3 random vectors from mult(n,p) for n=10:
>> n=10; r = mnrnd(n,p,3)
r =
     1     6     3
     3     2     5
     3     4     3
% compute the PMF of mult(n,p):
>> count1 = 0:n; count2 = 0:n;
>> LP=length(count1);
>> [x1,x2] = meshgrid(count1,count2);
>> x3 = n-(x1+x2);
>> y = mnpdf([x1(:),x2(:),x3(:)],repmat(p,(LP)^2,1));

% Plot the PMF (see Figure C2.3):
>> y = reshape(y,LP,LP);   bar3(y)
>> set(gca,'XTickLabel',0:n);  set(gca,'YTickLabel',0:n);
>> xlabel('x_1');  ylabel('x_2');  zlabel('PMF')
```

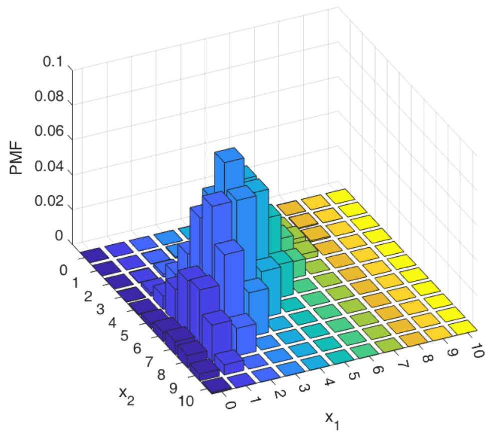


FIGURE C2.2: Probability mass function of a multinomial random variable with $n = 10$ and probability vector $(0.3, 0.5, 0.2)$.

C2.3 Python Functions for Continuous Probability Distributions

Many continuous probability distributions are available in available in MATLAB, as can be verified in https://uk.mathworks.com/help/stats/continuous-distributions.html.

C2.3.1 Exponential and Gamma Distributions

We provide next code for generating random numbers from an exponential distribution and for plotting the *pdf* of an exponential and a gamma distributed random variable.

```
% generation of random numbers from Exp(1):
>> y1 = exprnd(1)  % a random number
y1 =
    3.4473

>> y2 = exprnd(1,1,4)  % a random row vector of dimension 1x4
y2 =
    1.2840    3.0754    2.3317    0.1942

% Plot of the probability density function of an Exp(1):
>> y = 0:0.1:10; f = exppdf(y,1);
>> plot(y,f);  xlabel('y');  ylabel('f(y)')
```

The following code also shows how to find the 0.05 and 0.95 quantiles of an exponential distribution, such as done with `R` in Section 2.5.6:

```
>> lambda = 1;
>> pd = makedist('exponential','mu',lambda);  % mu is the lambda parameter
>> p = [0.05,0.95];                           % of an exponential distribution
>> x = icdf(pd,p)
x =
    0.0513    2.9957
```

Figure 2.12 in the book portrays gamma distributions with $\mu = 10$ and shape parameters $k = 1, 2$ and 10. Such a plot can be derived in `MATLAB` as shown below:

```
>> y = 0:0.1:40;
>> f1 = gampdf(y,1,10);   f2 = gampdf(y,2,5);  f3 = gampdf(y,10,1);
>> plot(y,f1);  hold on
   plot(y,f2);  plot(y,f3);  hold off
   xlabel('y'); ylabel('f(y)')
   legend('k = 1   (mean=10)','k = 2   (mean=10)','k = 10 (mean=10)')
```

C2.3.2 Normal Distribution

We use the *cdf* of a normal distribution to find tail probabilities or central probabilities. Next, using the *cdf* of the standard normal, we find the probabilities falling within 1, 2, and 3 standard deviations of the mean, as in the `R` code in Section 2.5.2:

```
>> cumprob = normcdf(-3:3) % cdf-values of a N(0,1) at the components of the vector -3:3
cumprob =
    0.0013    0.0228    0.1587    0.5000    0.8413    0.9772    0.9987

>> cumprob(5)-cumprob(3)   % probability within 2 standard deviation of mean
ans =
    0.6827

>> cumprob(6)-cumprob(2)   % probability within 2 standard deviation of mean
ans =
    0.9545

>> cumprob(7)-cumprob(1)   % probability within 3 standard deviation of mean
ans =
    0.9973
```

Next we use `MATLAB` for the Section 2.5.3 examples of finding probabilities and quantiles, such as finding the proportion of the self-employed who work between 50 and 70 hours a week, when the times have a $N(45, 15^2)$ distribution. We can apply normal distributions other than the standard normal by specifying μ and σ:

```
>> mu = 45;  sigma = 15
>> normcdf(70,mu,sigma) - normcdf(50,mu,sigma)
ans =                           % probability between 50 and 70
    0.3217

>> norminv(0.99)                % 0.99 quantile of standard normal
ans =
    2.3263

>> norminv(0.99,100,16)         % 0.99 normal quantile for IQ's
ans =                           % when mean = 100, standard deviation = 16
  137.2216

>> normcdf(550,500,100)         % SAT = 550 is 69th percentile
ans =                           % when SAT mean = 500, standard deviation = 100
    0.6915

>> normcdf(30,18,6)             % ACT = 30 is 97.7 percentile
ans =                           % when ACT mean = 18, standard deviation = 6
    0.9772
```

The code for plotting the *pmf* of a Poisson distribution with $\lambda = 100$ along with the *pdf* of a normal with $\mu = 100$ and $\sigma = 10$ (see Figure C2.3) is shown next:

```
>> lambda = 100;
>> y1 = 60:141;             % y values between 60 and 140 with increment of 1
>> f1 = poisspdf(y1,lambda);
>> mu = lambda; sigma = sqrt(lambda);
>> y2 = 60:0.1:141;         % y values between 60 and 140 with increment of 0.1
>> f2 = normpdf(y2,mu,sigma);
>> figure
   bar(y1,f1,1); hold on
   plot(y2,f2,'LineWidth',2); xlabel('y'); ylabel('Probability')
   title('Poisson pmf and Normal pdf'); legend('Poisson','Normal','location','northeast')
   hold off
```

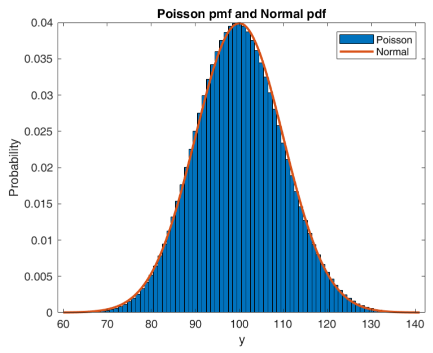


FIGURE C2.3: Probability mass function of a Poisson random variable when $\mu = 100$ and pdf of a $N(100, 10^2)$ in red.

C2.3.3 *Q-Q* Plots and the Normal Quantile Plot

Exercise 2.67 in Chapter 2 and Section A.2.2 in the R Appendix introduced the *Q-Q plot* (*quantile-quantile plot*), which compares graphically the quantiles of an observed sample data distribution with those of a theoretical distribution. If the theoretical distribution considered is the standard normal, then the Q-Q plot is called a *normal quantile plot.*

To illustrate how textitQ-Q plots are constructed in MATLAB, we construct one for the carbon dioxide emissions values for the 31 European nations in the Carbon.dat data file and for the Carbon_West.dat data file at the book's website that adds four Western nations to the data file for Europe. In MATLAB they are imported as Carbon and CarbonWest. Figure C2.4 shows these plots. For a discussion and interpretation of these plots we refer to Section B.2.3 of the book's Appendix.

```
% Normal quantile plot for the CO2 variable of data file Carbon:
>> CO2 = Carbon.CO2;      % a vector containing values of variable CO2 of Carbon
>> qqplot(Carbon.CO2)

>> qqplot(CarbonWest.CO2) % qqplot for the CO2 variable in CarbonWest
```

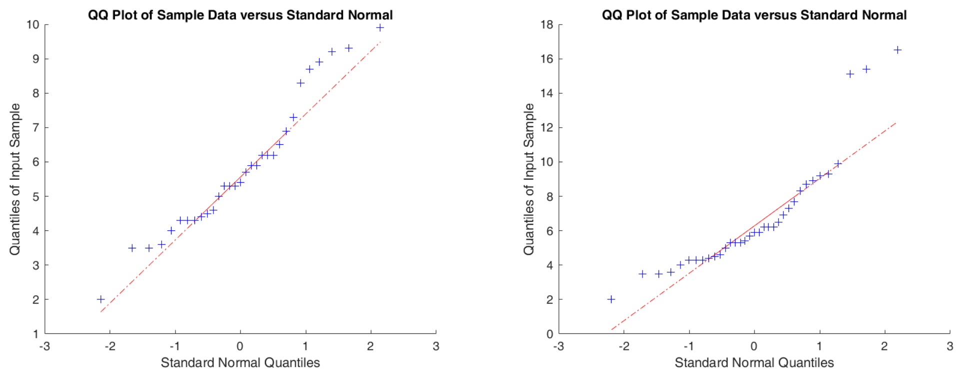


FIGURE C2.4: Normal quantile plots for carbon dioxide emissions, for 31 European nations (left) and also including four other Western nations (right).

C2.4 Expectations of Random Variables

C2.4.1 Binomial distribution

For a sufficiently large number of simulations, the sample mean of a random sample from a binomial distribution is close to its expected value. This was illustrated in Section 2.3.1 by an example, which is shown here in MATLAB:

```
% randomly generate a random sample with 10000000 bin(3,0.5):
>> y = binornd(3, 0.5, 1, 10000000);      % row vector
>> y(1:7)            % first 7 of 10 million generated
ans =
     3     1     1     3     0     3     2
```

```
>> mean(y)     % sample mean of 10000000 binomial outcomes
ans =
    1.4998  % binomial expected value n(pi) = 3(0.5) = 1.5
```

For the example in Section 2.4.4 of gauging the popularity of a prime minister, using a sample survey with $n = 1500$ when $\pi = 0.60$, we use `Python` to find the mean and standard deviation of the relevant binomial distribution and find the probability within 2 and within 3 standard deviations of the mean:

```
>> n = 1500;  p = 0.60;
>> [mu, sigma2] = binostat(n,p);
>> mu
mu =                  % mean of binomial(1500, 0.60)
   900
>> sigma = sqrt(sigma2)
sigma =               % standard deviation of binomial(1500, 0.60)
   18.9737

>> binocdf(mu + 2*sigma, n, p) - binocdf(mu - 2*sigma, n, p)
ans =           %  probability within 2 standard deviations of mean
    0.9519
>> binocdf(mu + 3*sigma, n, p) - binocdf(mu - 3*sigma, n, p)
ans =           %  probability within 3 standard deviations of mean
    0.9971
```

Since this binomial distribution is approximately normal, the probabilities above are close to the normal probabilities of 0.9545 and 0.9973.

C2.4.2 Uniform Distribution

Section 2.3.3 showed that a uniform random variable over the interval $[0, U]$ has $\mu = U/2$ and $\sigma = U/\sqrt{12}$. Next we find in `MATLAB` the mean and standard deviation of a simulated sample of 10 million random outcomes from a uniform [0, 100] distribution, for which $\mu = 50.0$ and $\sigma = 28.8675$:

```
>> a = 0;  b = 100;  n = 10000000;
% generate a column vector y with n random numbers in the interval (a,b):
>> y = a + (b-a).*rand(n,1);
>> y(1:3)    % first 5 simulated values
ans =
   50.4711
   24.1895
    8.0480

>> mean(y)  % mean of values in list y
ans =
   50.0115
>> std(y)
ans =   % standard deviation of values in list y
   28.8661
```

C2.4.3 Finding the Correlation For a Joint Probability Distribution

For a particular joint probability distribution, we can find the correlation using equation (2.16). We illustrate for the correlation between income and happiness for the joint distribution in Table 2.5.

```
>> prob = [0.2, 0.1, 0.0; 0.1, 0.2, 0.1; 0.0, 0.1, 0.2]; % probability table
>> x = [1;2;3];                              % column vector of row categories
>> y = [1 2 3];                              % row vector of column categories
% expected value of XY, E(XY):
>> EXY = sum((x*y).*prob, 'all')  % elementwise multiplication of matrices: A.*B
ans =                             % or: times(A,B)
    4.4000
>> rowp =sum(prob,2);             % marginal row probabilities (column vector)
>> colp =sum(prob,1)              % marginal column probabilities (row vector)
colp =
    0.3000    0.4000    0.3000
>> EX = sum(rowp.*x)
EX =
     2
>> EY = sum(colp.*y);
>> colp =sum(prob,2);              % marginal column probabilities
>> varX = sum(rowp.*x.^2) - EX^2   % elementwise matrix powers: .^
varX =
    0.6000
>> varY = sum(colp.*y.^2) - EY^2;
>> corrXY = (EXY-EX*EY)/sqrt(varX*varY)
corrXY =
    0.6667
```

3

CHAPTER 3: MATLAB FOR SAMPLING DISTRIBUTIONS

C3.1 Simulation to Illustrate a Sampling Distribution

To explain the concept of a sampling distribution, Section 3.1.1 used simulation to illustrate results of an exit poll in a U.S. Presidential election, when the probability is $\pi = 0.50$ of voting for Joe Biden. Here this is done in `MATLAB` using a random sample of 2271 voters:

```
n = 2271;  p = 0.5;         % values for binomial n, pi
y = binornd(n, p, 1)        % 1 binomial experiment
y =                         % binomial random variable = 1106 Biden votes
        1106
>> y/n                      % simulated proportion of Biden votes = 0.487
```

The above process is repeated 100000 times next, aiming at investigating the variability in the results of the simulated proportion voting for Biden, when half of the population voted for him. Also shown is the code for deriving the histogram of the 100000 simulated proportions. The histogram is pictured in Figure C3.1 (compare to Figure 3.1).

```
sim = 100000;
prop = binornd(n, p, sim, 1)/n;  % vector with the sample proportion values
mean(prop)    % mean of 100000 sample proportion values
ans =
    0.5000
std(prop)     % standard deviation of 100000 sample proportions
ans =
    0.0105

hist(prop, 18);  % histogram of the sampe proprtions with 18 bins
xlabel('Sample Proportion');  ylabel('Frequency')
```

C3.2 Law of Large Numbers

The simulation discussed in Section 3.2.5, to illustrate the law of large numbers, is performed here in `MATLAB`, using the code for uniform random number generation already seen in Section C2.4.2:

```
>> a = 0;  b = 100;
>> n1 = 10;  n2 = 1000;  n3 = 10000000;
% generate a column vector y with n random numbers in the interval (a,b):
>> y1 = a + (b-a).*rand(n1,1);
>> format long              % for output with more than 4 decimal places
>> mean(y1)                           % sample mean for random sample of
```

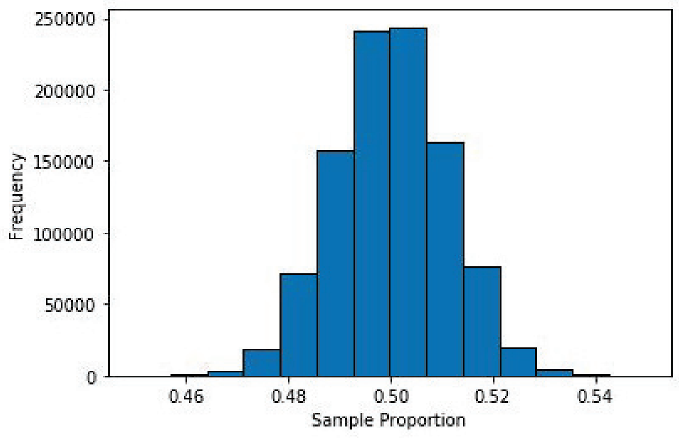


FIGURE C3.1: Histogram of 100000 simulations of the sample proportion favoring Biden, for simple random samples of 2271 subjects from a population in which exactly half voted for Biden.

```
ans =                                  % n=10  from uniform [0,100]
  64.283843396916197                   % (population mean = 50.0)
>> y2 = a + (b-a).*rand(n2,1); mean(y2)
ans =                                  % sample mean for n=1000
  49.045016155333144
>> y3 = a + (b-a).*rand(n3,1); mean(y3)
ans =                                  % sample mean for n=10000000
  50.009942307981206
```

Bibliography

Martinez, W. L., and Cho, M. (2015). *Statistics in MATLAB: A Primer*. CRC Press.

Martinez, W. L., and Cho, M. (2016). *Computational Statistics Handbook with MATLAB*. 3rd edition, Chapman & Hall/CRC.

Metcalfe, A., Green, D., Greenfield, T., Mansor, M., Smith, A., and Tuke, J. (2019). *Statistics in Engineering: With Examples in MATLAB and R*. 2nd edition, Chapman & Hall/CRC.

Rogers, S., and Girolami, M. (2017). *A First Course in Machine Learnig*. 2nd edition, Chapman & Hall/CRC.