In [1]:
```
options(jupyter.rich_display = FALSE)
```

Assignment 4

Due June 20, 2021, 23:59

General notice about submissions

In every problem we ask you to create one or more functions with specific names. Suppose you are asked to create functions named `foo()` and `bar()`. When you submit your code, the code must have ONLY the definitions of the functions, and not any CALLS to them. Also, you must not include any `set.seed()` statements in your submissions.

Good example:

```
# mysubmission.r
foo <- function(x){
    return 2*x
}
bar <- function(x, y){
    return x+y
}
baz <- function(N){
    runif(N)*2
}
```

Bad example:

```
# mysubmission.r
foo <- function(x){
    return 2*x
}
foo(c(1,2,3)) # don't call the function here
bar <- function(x, y){
    return x+y
}
mysum <- bar(3,2) # don't make an assignment outside a function block

set.seed(111) # don't set the random seed here
baz <- function(N){
    runif(N)*2
}
```

The reason is that when we run your code for testing purposes, such calls may create some unwanted side effects and interfere with the grading.

When you work on your programs, feel free to call your functions to check that they give correct results. However, in the final submission keep only the function definitions. Your submission should

not do anything except for defining the functions.

Keep in mind that we are going to test your programs with new data that has the same structure but different content. Write your programs to be as general as possible within the description.

I suggest that you come up with further test cases on your own, and check that they give the expected output. Make your final tests after restarting R, so that existing variables do not confound your program.

You must work on the assignment by yourself. **Giving or receiving any help will not be tolerated.**

Notice on data files: When writing your code, you should assume that the given data files are in the same directory as the source files. Do NOT call `setwd()` in your code. You can use the `getwd()` command to find your current working directory and copy your data files there.

Problem 1: Gas mileage prediction

The text file `auto-mp.data` stores a table of data about different car models. For each model, it provides the miles-per-gallon (mpg), number of cylinders, the displacement, engine horsepower, car weight, the acceleration, the model year, origin code, and the name of the model.

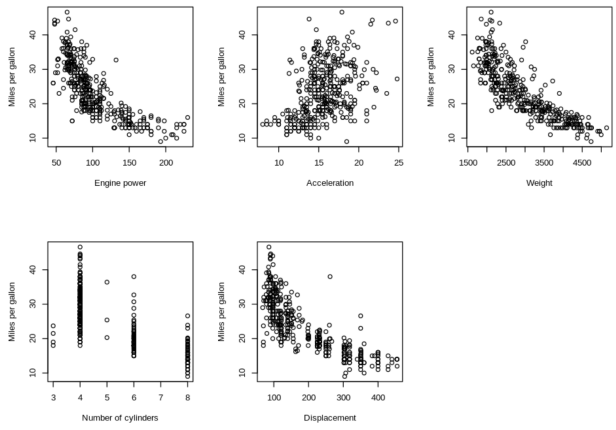
In the following, you can use the `read.table()` function in order to read the data into a data frame.

(A) *(15 points)* Create a function named `plot_car_data(filename)` that generates a plot of the `mpg` variable versus the other variables, as shown below. The function should take the data file name as its single parameter. It should not return any value.

Hint: Use the `par()` command to create a plot array as shown.

Example

```
> plot_car_data("auto-mpg.data")
```



(B) *(15 points)* Assuming that the fuel consumption (mpg) of a car depends linearly on engine power, on acceleration, and on the weight of the car, we propose the following linear model:

$$\text{mpg} = a \cdot \text{horsepower} + b \cdot \text{acceleration} + c \cdot \text{weight} + d$$

The coefficients a, b, c, and d are unknown, and will be determined by fitting a linear model to the dataset given in the file `auto-mp.data`. (We don't need the coefficients themselves in this problem.)

Write a function named `predict_mpg(p, a, w)` that returns an estimate for the miles-per-gallon for the car, given the horsepower `p`, acceleration `a`, and weight `w`. The function should use the `lm()` function to set up a linear model as given above, and fit it with the data given in `auto-mp.data` (read the file inside the function; don't use a global variable).

The function should return the model's prediction for the given `p`, `a`, and `w`, which can be vectors.

Note: To make a prediction inside the function, convert the passed parameters to a data frame with columns named `horsepower`, `acceleration`, and `weight`. This should be then passed to `predict.lm()` as a parameter.

Examples

```
> predict_mpg(100, 15, 3500) # single car model with horsepower 100,
acceleration 15 and weight 3500
       1
20.63484
> predict_mpg(c(100, 110, 120, 150), c(10,15,20,25), c(2000, 2200, 3500,
4000))  # four different car models
       1        2        3        4
29.32927 27.68610 19.67460 15.34470
```

Problem 2: Fitting cosines to a wave

The file `dataxy.csv` stores data in two columns, `"x"` and `"y"`. Your task is to fit several different models to the data, with an increasing number of linear terms.

The data appears to be periodic and symmetric about the y-axis (see the plot in the example), so we try to fit a sequence of *cosine* functions to it, in the form:

$$y = a + b\cos(x) \tag{1}$$
$$y = a + b\cos(x) + c\cos(3x) \tag{2}$$
$$y = a + b\cos(x) + c\cos(3x) + d\cos(5x) \tag{3}$$

Just as we have done in polynomial fitting, we can define new variables $x_1 \equiv \cos(x)$, $x_2 \equiv \cos(3x)$, $x_3 \equiv \cos(5x)$, etc., in order to convert this problem in to a linear regression problem.

(A) *(10 points)* First, try to fit a model with the first term. That is, assume $y = a + b\cos(x)$.

Write a function named `first_model(filename)` that reads data from file named `filename`. The file is assumed to have two columns, named `"x"` and `"y"`. The function should return a vector of model coefficients, as shown below.

Example:

```
> first_model("dataxy.csv")
(Intercept)        cosx
  0.2700142   0.8059648
```

(B) *(10 points)* Now try the second-order model; that is, assume $y = a + b\cos(x) + c\cos(3x)$.

Write a function named `second_model(filename)` that reads data from file named `filename`. The file is assumed to have two columns, named `"x"` and `"y"`. The function should return a vector of model coefficients, as shown below.

Example:

```
> second_model("dataxy.csv")
(Intercept)        cosx       cos3x
 0.27160566  0.81217534  0.08404965
```

(C) *(10 points)* Finally, fit a third-order model; that is, assume $y = a + b\cos(x) + c\cos(3x) + d\cos(5x)$.

Write a function named `third_model(filename)` that reads data from file named `filename`. The file is assumed to have two columns, named `"x"` and `"y"`. ThThe function should return a vector of model coefficients, as shown below.

Example:

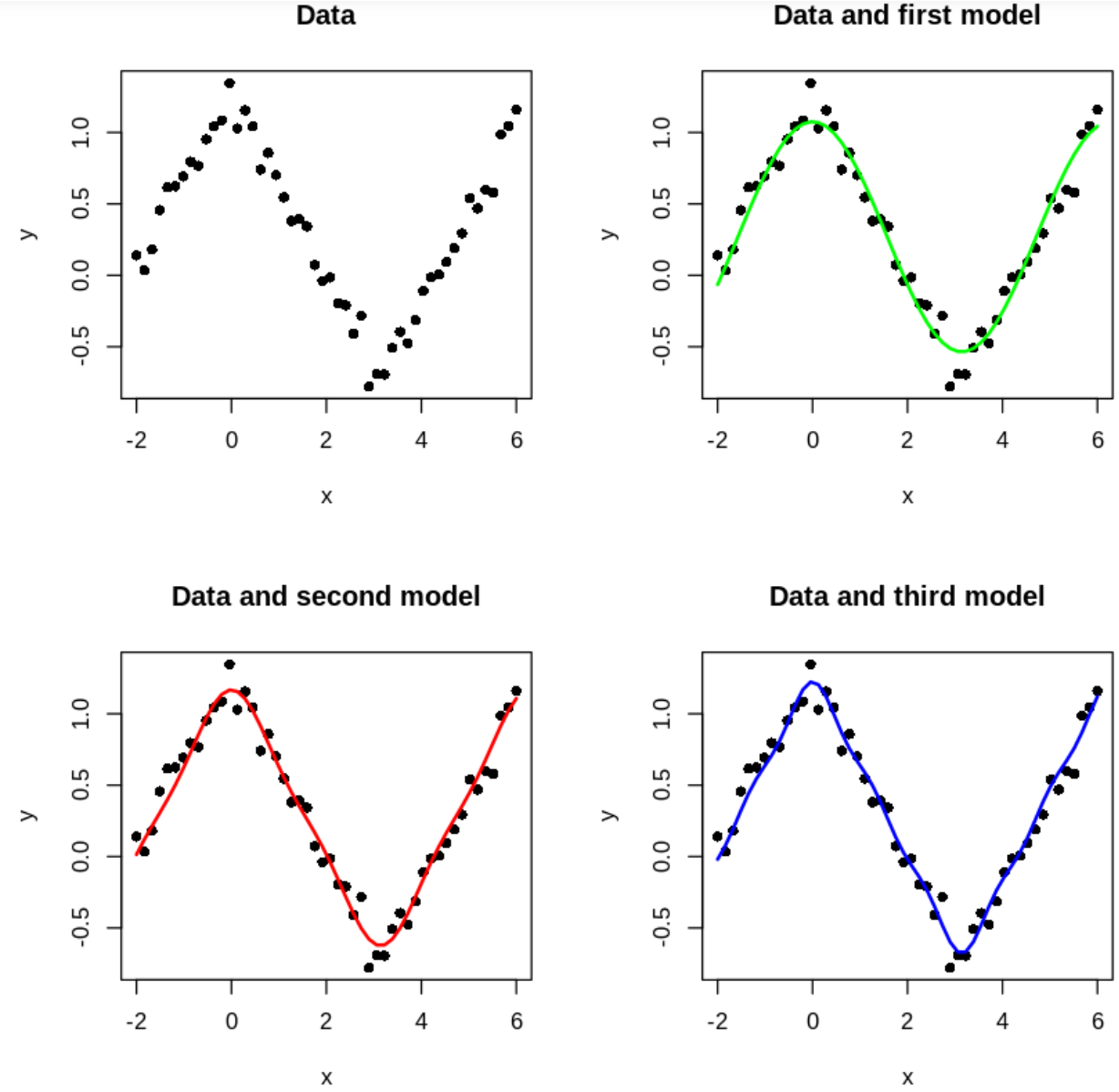
```
> third_model("dataxy.csv")
(Intercept)        cosx       cos3x       cos5x
 0.27372830  0.81307666  0.08952228  0.04872744
```

(D) *(10 points)* Generate a plot comparing the models' predictions with the data points.

Write a function `plotmodels(filename)` which takes the name of the data file with columns `"x"` and `"y"`, and generates a 2-by-2 plot as shown below. In each panel, the black dots are the data points and the colored lines are *predictions of the model* generated at the same x values.

Note: The predictions should be generated with `predict.lm()` function calls, not by manually copying the output of previous parts, otherwise the automated tests will fail.

```
> plotmodels("dataxy.csv")
```



Replicate the plot with correct axis labels, colors, grid and legend. The aspect ratio (height/width) does not have to be the same. Set the line widths to 2.

Problem 3: Workforce transfers

The small but proud Duchy of Grand Fenwick has three main economic sectors: Agriculture, Industry, and Services (they had banished the finance sector 300 years ago).

Although people mostly stay in their own sector of expertise, switching between sectors is not uncommon. Based on historical records, the fraction of people that move between sectors each year is summarized in the following table.

		From		
		Agriculture	Industry	Services
To	Agriculture	0.75	0.2	0.12
	Industry	0.1	0.55	0.22
	Services	0.15	0.25	0.66

The first column of the table shows that, among people who work in agriculture, 75% keep working in the same sector in the following year, 10% switch to a job in industry, and 15% switch to a job in services. The second and the third columns show the transfer rates from industry and services, respectively, to other sectors.

At a given year t, let A_t, I_t, S_t be the number of people in agriculture, industry, and the service sectors, respectively. Then, according to the table above, the number of people in each sector at year $t+1$ can be expressed as

$$A_{t+1} = 0.75A_t + 0.2I_t + 0.12S_t \quad (4)$$
$$I_{t+1} = 0.1A_t + 0.55I_t + 0.22S_t \quad (5)$$
$$S_{t+1} = 0.15A_t + 0.25I_t + 0.66S_t \quad (6)$$

With these equations, if we know the initial number of people working in each sector in one year, we can estimate the number for all the following years.

(We do not take births and deaths into account. We assume that they are "magically" balanced so that the total workforce is 100,000 people at all times.)

(A) *(20 points)* Write a function named `fenwick_workforce(years, initial, rates)`. The function parameters are:

- `years` : the number of years (steps) to calculate
- `initial` : a 3-element vector containing the initial numbers (A_0, I_0, S_0) in each sector.
- `rates` : a 3-by-3 matrix contaning the change rates as given in the table.

The function should return **a data frame** containing the number of people in each sector every year. Each row corresponds to one year, starting with the initial value. Numbers must be rounded to the nearest integer.

Use a loop that starts with the initial population values, and calculates the next year's populations using the relations above, for `years` steps. The output must have `years` +1 rows.

Because of the rounding, numbers in some rows may not add up exactly to 100000. This is acceptable.

Important: The calculations in the function should use the matrix `rates` given as a parameter. There must be no "hard-coded" numeric values in the function code.

(Alternatively, you can use the `%*%` operation to make matrix-vector calculations, but it is not required.)

Example:

```
> M <- matrix(c(0.75, 0.2, 0.12,
              0.1, 0.55, 0.22,
              0.15, 0.25, 0.66),
    ncol = 3, byrow = T)
> init <- c(30000, 40000, 30000) # initial number of workers
> fenwick_workforce(10, init, M)
```

```
   Agro  Inds  Serv
1  30000 40000 30000
2  34100 31600 34300
3  36011 28336 35653
4  36954 27030 36017
5  37444 26486 36072
6  37709 26248 36046
7  37857 26137 36009
8  37941 26083 35979
9  37990 26055 35958
10 38018 26040 35945
11 38035 26032 35936
```

Another example with different rates of movement between sectors:

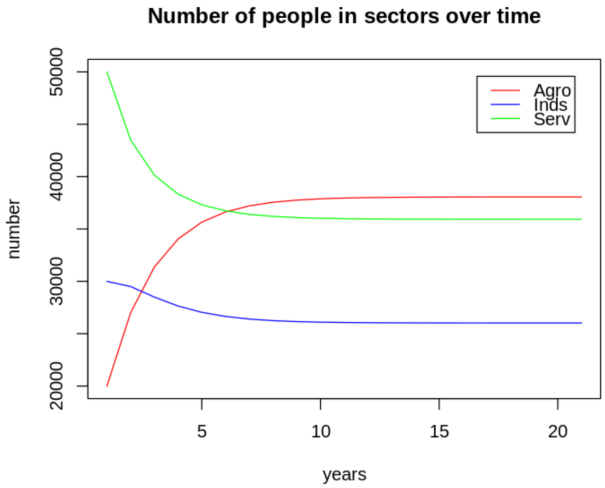
```
> M <- matrix(c(0.70, 0.25, 0.15,
              0.2, 0.60, 0.20,
              0.1, 0.15, 0.65),
            ncol = 3, byrow = T)
> init <- c(50000, 20000, 30000)
> fenwick_workforce(15, init, M)
   Agro  Inds  Serv
1  50000 20000 30000
2  44500 28000 27500
3  42275 31200 26525
4  41371 32480 26149
5  41002 32992 26006
6  40850 33197 25953
7  40787 33279 25934
8  40761 33311 25928
9  40750 33325 25926
10 40745 33330 25925
11 40743 33332 25925
12 40742 33333 25926
13 40741 33333 25926
14 40741 33333 25926
15 40741 33333 25926
16 40741 33333 25926
```

(B) *(10 points)* Write a function named `plot_workforce_time(years, initial, rates)` that plots the evolution of the number of people in each sector, as shown. You can call `fenwick_workforce()` from within this function, and plot the columns separately.

The vertical limits of the plot should be set using the minimum and maximum of all values.

Example

```
> M <- matrix(c(0.75, 0.2, 0.12,
              0.1, 0.55, 0.22,
              0.15, 0.25, 0.66),
    ncol = 3, byrow = T)
> init <- c(20000, 30000, 50000)
> plot_workforce_time(20, init, M)
```



Submission template

The function definitions should be submitted as a single source file named *assignment4.R*, with the following contents:

```
plot_car_data <- function(filename){
    # your code here
}

predict_mpg <- function(p,a,w){
    # your code here
}

first_model <- function(filename){
    # your code here
}

second_model <- function(filename){
    # your code here
}

third_model <- function(filename){
    # your code here
}

plotmodels <- function(filename){
    # your code here
}

fenwick_workforce <- function(years, initial, rates){
    # your code here
}

plot_workforce_time <- function(years, initial, rates){
    # your code here
}
```