1 TP N°1 : Arduino Tutorial

2 TP N°2: Characterization of motors (Arduino + Matlab)

The aim of this work is to characterize a DC motor in order to control its speed. The project stages are as follows:

- Connect the motors to the boards.
- Test manually motor behavior
- Automatically test the motor to obtain a speed response curve
- Model the drive-speed relationship of the motor
- Write a direct motor control program

We assume that at this point, the operation of Arduino boards is known, we will now move on to a practical aspect.

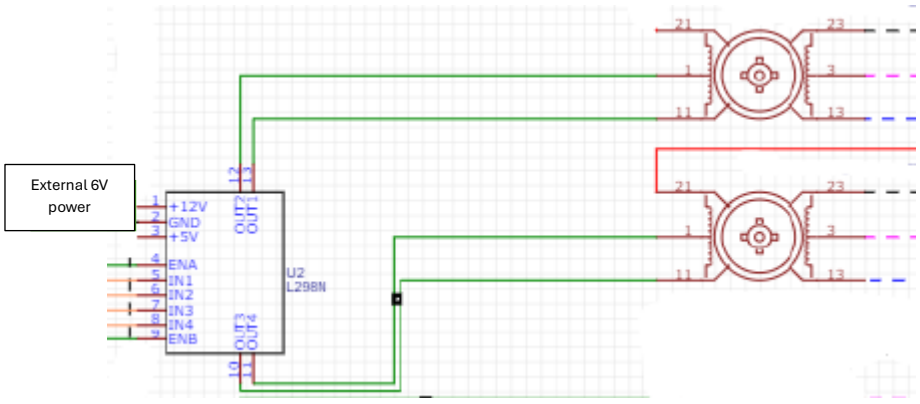
2.1 Motor connections

2.1.1 Presentation

The project can be done in 2 versions:

2 motors with Arduino board Uno (for example for a car with 2 wheels) or 4 motors (for example for an off-road vehicle with 4 wheels) with an Arduino board Mega 2560. We need one interrupt input per motor and Arduino uno boards have only 2 interrupt inputs. Therefore, larger cards are needed for the 4-motor version.

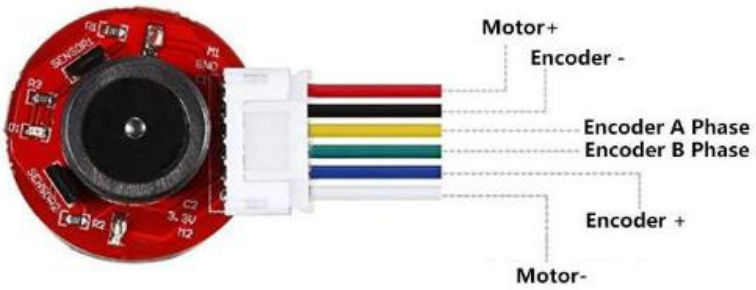
Each motor is fed via a L298N reference H-bridge for example. Each circuit is capable of controlling 2 motors according to the following scheme



The driver receives from the microcontroller the control signals on the pins ENA, IN1, IN2 for the first motor and ENB, IN3, IN4 for the second motor.

The +12V and GND pins must be connected to an external power supply to provide power to the motors (6V or 12V depending on the type of motor)

On the motors side, the motor is powered from the L298N via the signals Motor+ and Motor -. The position encoders of the motor must be powered by the 5V and the ground of the Arduino wires Encoder+ and Encoder-. They provide 2 signals Encoder A and Encoder B which will be connected to the Arduino board.



Here are the connections proposed on an Arduino UNO board for 2 motors

Numéro de Broche	Branchement
0	réservé communication
1	réservé communication
2	Moteur 0 encodeur B
3	Moteur 1 encodeur B
4	Moteur 0 encodeur A
5	L298-ENA
6	L298-ENB
7	Moteur 1 encodeur A
8	L298-IN1
9	L298-IN2
10	L298-IN3
11	L298-IN4
12	libre
13	libre

And the connection on a MEGA 2560 board for 4 motors

N°	Branchement
0	réservé communication
1	réservé communication
2	Moteur 0 encodeur B
3	Moteur 1 encodeur B
10	Moteur 3 EN
11	Moteur 2 EN
12	Moteur 1 EN
13	Moteur 0 EN
18	Moteur 2 encodeur B
19	Moteur 3 encodeur B

34	Moteur 0 INA
35	Moteur 0 INB
36	Moteur 1 INA
37	Moteur 1 INB
38	Moteur 2 INA
39	Moteur 2 INB
40	Moteur 3 INA
41	Moteur 3 INB
44	Moteur 0 encodeur A
45	Moteur 1 encodeur A
46	Moteur 2 encodeur A
47	Moteur 3 encodeur A

It is possible to change the spindle if you follow the rules:

- the encoder signals B must be on interrupt inputs
- Motor x EN signals must be connected to PWM outputs

2.1.2 Work to be done

Connect the individual wires between the power supply, H-bridges and motors as described above

Power supply 6 or 12V -> H bridge

Arduino -> H-bridge (*ena* and *in* signals)

H bridge -> motors

Motors -> Arduino (position encoders)

2.2 Exercise #1: First manual test of a motor

2.2.1 Presentation

The purpose of this first exercise is to manually test the behavior of a motor. We will use for this a first Arduino program that we provide you in the directory «2.2 Test manuel du moteur», and which allows to control a motor from a potentiometer.

The motor is controlled by a variable PWM signal RC. The command can range from -255 (motor running at full speed in one direction), to 0 (motor stopped) and up to 255 (motor running at full speed in the other direction).

The average voltage received by the motor is therefore VDD*RC/255 where VDD is the supply voltage of the motors.

In the rest of the document we will only consider the RC command which is the cyclic ratio of the PWM signal. The average voltage received by the motor is given by the previous formula

2.2.2 General information about Arduino programs

Each Arduino program must be contained in a directory with the same name as the project.

When the Arduino environment is installed, just click on the .ino program to launch the development environment.

On the upper bar of the environment, 4 icons will help us

 «Verify» this first icon compiles the program without downloading it on the map (to check the syntax)

 «Upload» Second icon compiles and downloads the program on the map

 »serial plotter» This icon opens a graphical window and displays the data sent to the terminal

 »Serial monitor» This icon opens the serial monitor in which messages sent by the card are displayed

An «exportMatlab» constant defined at the beginning of each program indicates whether the results are sent in matlab format (to be imported later) (true) or arduino format for a display as a curve (false).

All programs contain at least 2 functions:

- setup(): this function is started at the beginning of the program. It is usually used to configure all devices
- loop(): this function is a loop that runs continuously once the «setup» is done

As mentioned before the programs can run in 2 or 4 motors mode with an Arduino uno or Arduino Mega board. A mega2560 flag at the beginning of each program allows you to choose the card used

if we leave `#define mega2560` we are in 4 motors mode by a mega2560 card, if we comment the line we pass in Arduino uno with 2 motors. This is valid for all the following programs.

2.2.3 Description of program functions

`void setSpeed(int speed);` this function defines the motor control voltage. The argument is 0 to stop the motor, 255 to turn it in one direction at maximum speed, and -255 to turn it in the other

int readSpeed(): This function returns the estimated motor speed. The estimate is made between 2 measurements as the difference of position divided by the difference of time. It is therefore necessary to run the function 2 times in a row spaced for example by one second to have the estimation of the speed of the motor.

void calculateControl(): This routine defines the motor control from the voltage supplied by the potentiometer

void writeText(): This routine displays measurement information on the console

void pulse1(): This routine is special. It's an interruption. It is called each time the motor position encoder sends a pulse. It increments or decrements the position meter according to the direction of rotation of the motor.

2.2.4 Work to be done

- Connect the potentiometer between the Arduino supply voltage and ground. The center pin will be connected to the A0 input of the Arduino board.
- check that exportMatlab = false at program start
- Launch the program "Test_Moteur_Manuel.ino" and download it to the board. Open the integrated serial monitor window. Choose the speed of 115200 bauds. You can see the value of the control and the speed of rotation of the motor.
- Make a hand reading by moving the potentiometer to roughly plot the characteristics of the motor response

- What are the two phenomena that can be noticed on the curve? Provide a probable explanation for these phenomena

Answer: hysteresis: between -50 and 50 the motor is stopped

A saturation for large values due to resistance inside the windings

2.3 Exercise #2: Automate the variation of the control voltage and recovery of the speed of a motor

2.3.1 Presentation

We will now transform the previous program in such a way that the control of the motor is automatically varied periodically and draw the characteristic curve of it

2.3.2 Work to be done

- Open the Test_1M_Auto.ino program which is currently a copy of the previous program
- Add definitions at the beginning of the program:

```
int cmd = -250;
int deltaspeed0=5 ;
int deltaspeed=-deltaspeed0 ;
```

- Modify the calculateControl function as follows

```
void calculateControl() {
 if (cmd+ deltaspeed >=255) deltaspeed=-deltaspeed0 ;
 if (cmd+deltaspeed <=-255) deltaspeed=deltaspeed0 ;
 cmd=cmd+deltaspeed ;
 setSpeed(cmd); }
```

- Change the timing of the main loop so that the motors have time to change gears and have a better precision on the calculation of the rotation speed.

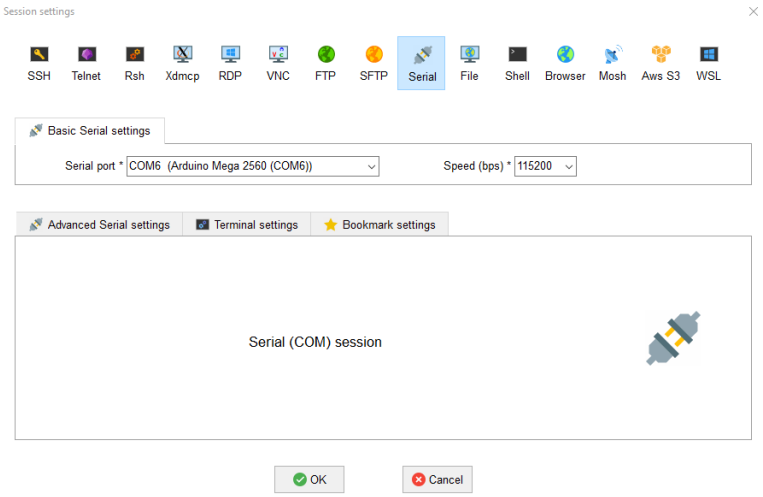
```
void loop() {
        calculateControl();
        delay(500);
        rpm=readSpeed();
        delay(1000);
        rpm=readSpeed();
        writeText();
}
```

- Run the program and check on the terminal that there is a periodic change of the command.
- We must now retrieve the values obtained in a text file to export them to matlab. We will start by setting the exportMatlab variable to true in the program:

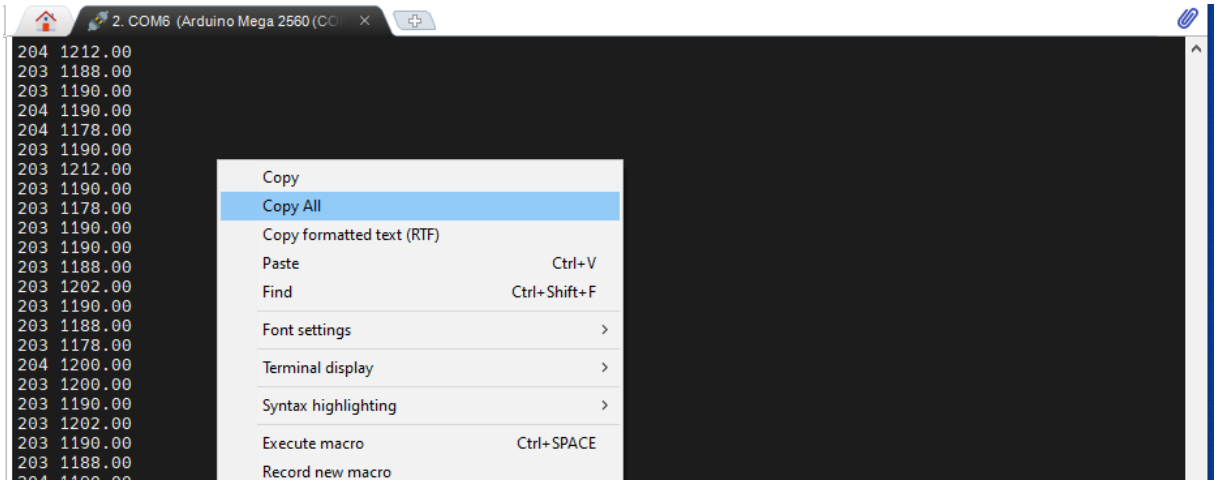
```
const bool exportMatlab = true ;
```

- Unfortunately the terminal of the Arduino environment makes it difficult to simply retrieve text. So we will close the serial monitor and open a moba-xterm. This program is free and can be downloaded from the internet at: better suited program as https://mobaxterm.mobatek.net/download-home-edition.html

- A new session is created by choosing a serial type and taking the serial port corresponding to the Arduino and speed 115200 bauds.



- The text from the Arduino is written well on the terminal. Just right click in the window to copy all the text (Command Copy all) and paste it into a text file (here called 'tm.txt').



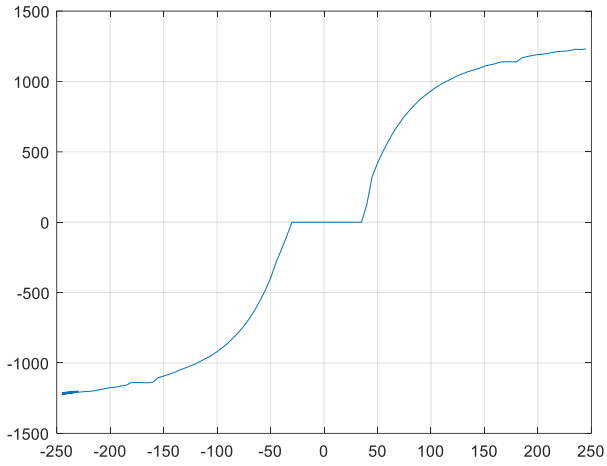
- Warning: before modifying a program or launching another project, you must close the session on mobaxterm otherwise it will not be possible for the arduino program to access the board

- Manually edit the resulting text file so that it only keeps a complete cycle from -255 to +255. You must be careful to only have complete lines, otherwise the file will not be readable by Matlab.
- Load the file on Matlab and display the response command <> rotation speed.

The 2-dimensional table has the N° of measurement as its first index. The second index is 1 for control and 2 for speed

```
data=load('tm.txt');
plot (data( :,1),data( :,2)) ;
grid on
```

We need to get a figure that looks like this



- Check that the resulting curve is consistent with the hand-drawn reading. Accurately record the minimum and maximum rotational speed as well as the minimum control values required to run the motor
- Repeat the whole process (using another measurement file) during a descending phase and not an ascending phase of the control voltages and look for any variations in characteristics. How can differences be explained if they exist?

2.4 Exercise #3: Modelling the continuous response of motors

2.4.1 Presentation

We now seek to obtain a mathematical model that allows, on the one hand, to model the motor and, on the other hand, to find the correct control law.

Therefore, we must obtain a set of equations that allows us to obtain

- The direct function: speed as a function of the command (to make later a simulink model of the motor behavior)
- The inverse function: The command needed to get a certain speed (to implement a direct control law on Arduino)

We will treat each half of the curve separately for positive and negative orders. It can be noted that for the implementation on Arduino of the inverse function, we will provide integer coefficients so as not to use floating point functions.

2.4.2 Work to be done

We will start from the previous measurement file which starts from the minimum negative value to the maximum positive value. If the file contains additional samples, you must delete them by hand otherwise the Matlab programs will not work.

In a first part, we will directly launch Matlab commands by hand. In a second part we will complete a script that automatically determines the different models

- We get 2 vectors that contain command and speed

```
data=load('tm.txt');
cmd=data(:,1)
vit=data(:,2);
```

- Then we recover the indices of the 'dead zone', where speed is zero.

```
ind1=find(vit==0,1,"first");
ind2=find(vit==0,1,"last");
```

- Then the positive and negative parts of the curves

```
cmdp=cmd(ind2+1:end);
vitp=vit(ind2+1:end);
cmdm=cmd(1:ind1-1);
vitm=vit(1:ind1-1);
```

- We start by working on the negative part of the curve. We try to fit the curve by a polynomial of order 2

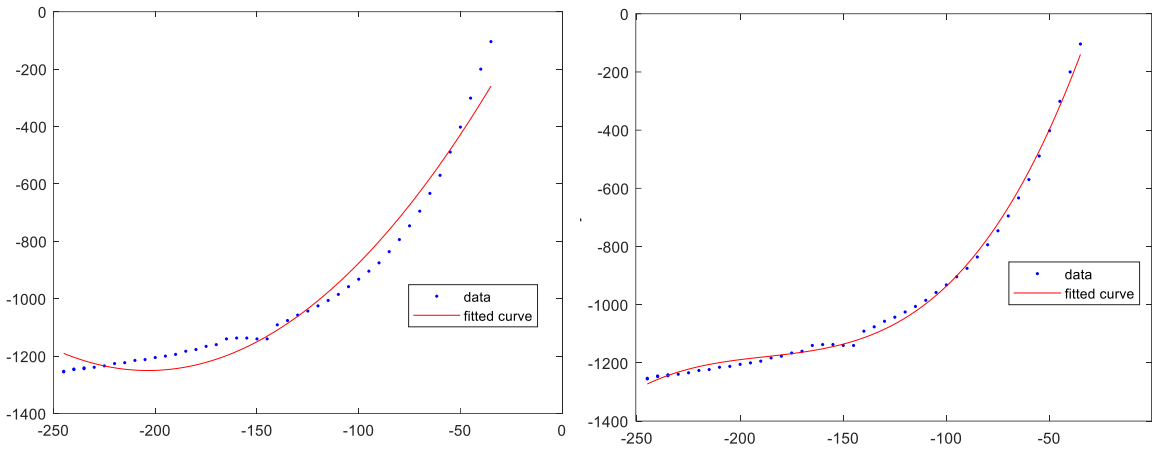
```
f=fit(cmdm,vitm,'poly2')
plot(f,cmdm,vitm);
```

Attention : this function uses the curve fitting toolbox that may need to be installed to run the program

- Then order 3

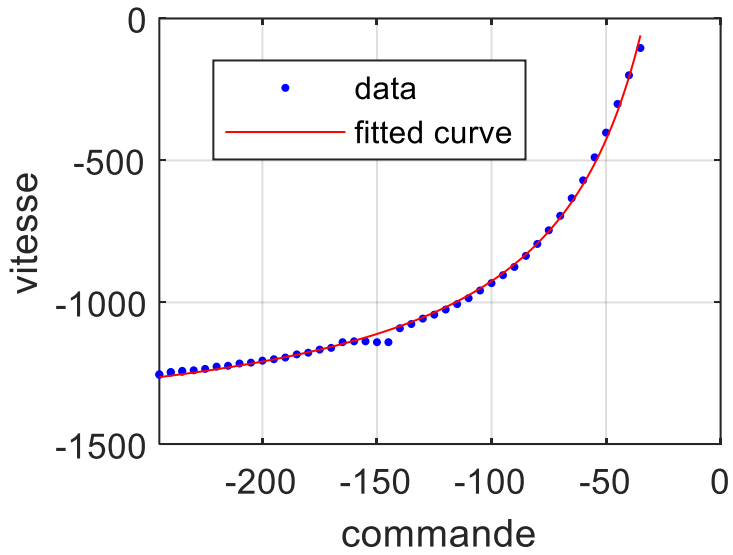
```
f=fit(cmdm,vitm,'poly3')
plot(f,cmdm,vitm);
```

As can be seen from the following curves, a polynomial function is not the right way to model behavior. The measurements are relatively far from the model.



- We now choose a model as a rational function

```
f=fit(cmdm,vitm,'rat11')
plot(f,cmdm,vitm);
p1i=fi.p1; p2i=fi.p2; q1i=fi.q1;
```

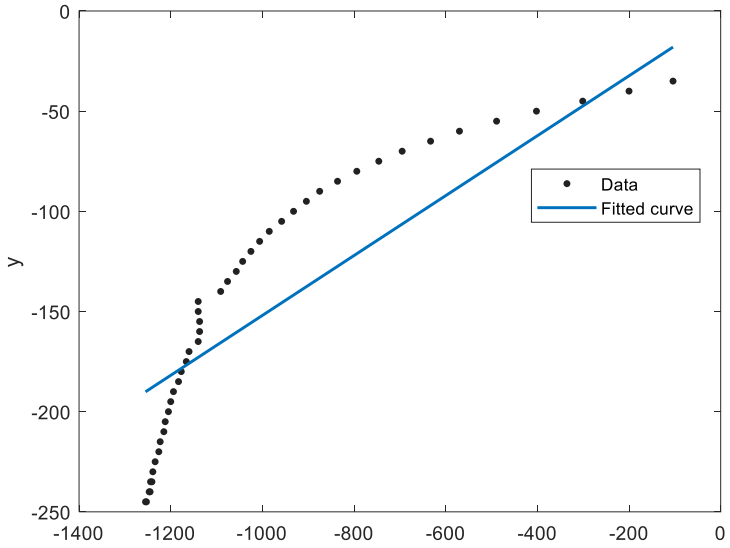


The resulting structure f contains the fields of the various coefficients:

f.p1 f.p2 or f.q1 the function being (p1*x+p2)/ (x + q1).

- And we can also model the inverse function, that is to say the command necessary to obtain a certain speed.

```
fi=fit(vitm,cmdm,'rat11')
plot(fi,vitm,cmdm);
```



And it may work or not at all, depending on the type of motor chosen, as can be seen on the previous curve

2.4.3 Correction/ Code improvement

As we have just seen, the results obtained may not be similar to the data provided especially for the modelling of inverse functions.

This is due to the fact that the method of finding coefficients is not linear. It is iterative and can converge to totally erroneous values.

In practice, it appears that the functions command->speed are modelled correctly while the inverse functions are not.

One way to do this is to provide initial values for the parameters p1 p2 and q1.

```
f=fit(cmdm,vitm,'rat11','StartPoint',[x y z])
```

Or x y and z are “not too silly” values of the parameters.

In particular it can be noted that if y=(p1x+p2)/(x+q1) then x = (-q1y + p2)/(y-p1).

The command becomes:

```
fi=fit(vitm,cmdm,'rat11','StartPoint',[-f.q1,f.p2,-f.p1]);
plot(fi,vitm,cmdm);
p1i=fi.p1; p2i=fi.p2; q1i=fi.q1;
```

On the other hand for an implementation on microcontroller, it is better to have integer coefficients.

We will transform the writing of the inverse function

x = (-q1i y + p2i)/ (y-p1i).

The smallest parameter is p1i. We will write it as p1i=num/den with num and den are integers

From there we rewrite the function in the form

x = (-q1i *den y + p2i*den)/ (y*den-num). the multiplier of y in the denominator is no longer 1 but at q2i.

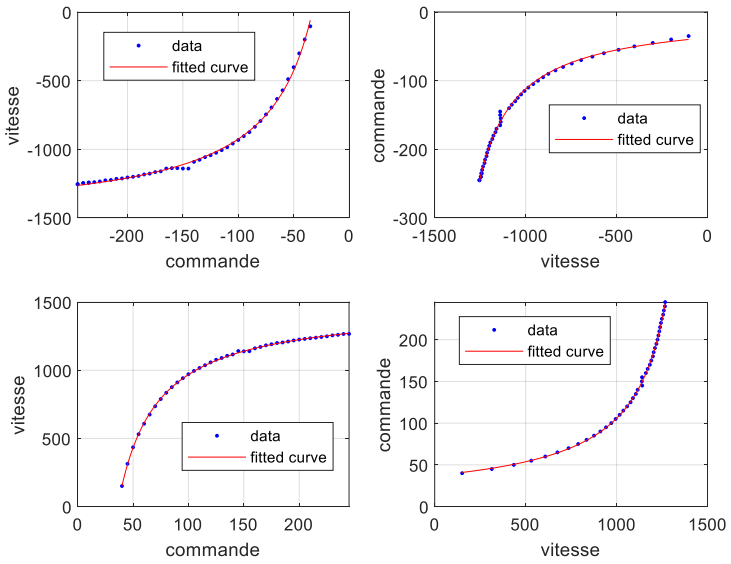
This can be done using the code

```
ratap=rats (p1i,10);
ind=find(ratap=='/');
num=eval(ratap(1:ind-1)); den=eval(ratap(ind+1:end));
disp(['p1i = ' num2str(num) '; p2i = ' num2str(round(p2i*den)) '; q1i = ' num2str(round(q1i*den)) ';
q2i = ' num2str(den)]);
```

2.4.4 Work to be done

The modeliser4.m program uses all the previous codes for the negative part of the curves. It can also process the coefficients of several motors.
By choosing k=1 at the beginning of the program, one works with only 1 motor.
The positive parts remain to be addressed

- Open the file modelliser4.m and complete it in order to treat the positive part of the curve identically
  We should get in final 4 curves as shown in the following figure



2.5 Exercise 4: direct control of the motors.

2.5.1 Presentation of tasks

The purpose of this part is to implement the previously calculated control function in order to give a speed signal and test the response of the motor (measure its speed). A function will be implemented to change the speed setting periodically in the form of stair steps

We will be working with a new set of programs from now on. The main change is the creation of periodic tasks that will wake up on a regular basis to carry out orders and measurements.

The Arduino being a mono-processor, it is not possible to implement real tasks as with a multi-core processor.

In order to understand how the stains work, we have provided you with a small project: "Example taches" which shows how the stains work. To activate a task, just run `TaskxOn();` where x is the task number. To disable a task, type `taskxon=false;`

In this project we have 2 tasks: the first task1 which displays the current time value every 500 ms and the second task3 which sends a message every 5 seconds. Task 2 is inactive.

2.5.2 Overview of the control program

The direct motor control program is based on two tasks:

- A first task: task1 which measures the speed of the motor every 200 ms
- A second spot that changes the speed setting every 5s in the form of a triangular signal. The pitch is contained in the variable `deltaconsv`et is worth 200 while the maximum speed setting is contained in the variable `consvmax`

We need a `speed2com` function that returns the command needed to reach a certain speed.

```
int speed2com(int spd,bool pred)
{
 int com=0 ;
 long int spdl=spd;
 if (pred==true)
 {  if (spd<0) {
      com = (spdl*p1i+p2i)/(spdl*q2i+q1i);
   }
   else if (spd>0) {
      com = (spdl*p3i+p4i)/(spdl*q4i+q3i);
   }
 } else {
   com = (spdl * 255) / vitmax ;
 }
 return(com);
}
```

This function is based on coefficients created previously. However, in order to speed up the calculations, it is better to use integer coefficients rather than floating point coefficients. We use a command with 4 coefficients (instead of 3) but which are all integers

```
    com = (spdl*p1i+p2i)/(spdl*q2i+q1i);
```

The display routine shows at the end the expected speed, the calculated setting, and the actual speed. An exportMatlab flag allows you to put the output in matlab format, so that you can import the results directly. When this flag is set to 0, we stay in the Arduino format and we can display the curves in real time using the Arduino visualization tool.

```
void writeText() {
  if (exportMatlab == true)
  {
   Serial.print(consv);
   Serial.print(" ");
   Serial.print(com1);
   Serial.print(" ");
   Serial.println(rpm);
  }
  else
  {
   Serial.print("Cons:");
   Serial.print(consv);
   Serial.print(", Cmd:");
   Serial.print(com1);
   Serial.print(", rpm1:");
   Serial.println(rpm);
  }
}
```

2.5.3 Work to be done

- Open the sample project. Understand how it works and test it on the Arduino board.
- Modify the program content to flash the internal led of the card (connected to pin 13 of the card) by means of a spot with a frequency of 1 hz
- Start the model4.m program by indicating the correct measurement file,
- Open the Test_Sans_Regulation program and modify the beginning of the program to give the correct values to the coefficients p1i, p2i, q1i, q2i, ... from previously found values
- Run the program with ExportMatlab set to False. Open the graphic window (Serial Ploter) of the Arduino and check the correct operation of the assembly. Note any speed deviations and conclude on the need for regulation.