Documentation of the MATLAB and LaTeX framework (Faryadell)

Abolfazl Delavar

Version **1.0.0**

Control Engineering & Computational Neuroscience
http://abolfazldelavar.com
faryadell@gmail.com

November 13, 2022

Contents

Chapter 1

Introduction

a) We have

$$s(i) = \frac{t^2(i)}{2}.g$$

such that

$$\theta = g \quad \& \quad \varphi^T(i) = \frac{t^2(i)}{2}$$

Also the Least Squares problem is

$$\hat{\theta}(i) = \left(\sum_{j=1}^{i} \varphi(j)\varphi^T(j) \right)^{-1} \left(\sum_{j=1}^{i} \varphi(j)y(j) \right). \tag{1.1}$$

for find g, assume that $s(i) = [0, 5, 19.5, 44]$ and $t(i) = [0, 1, 2, 3]$ for $i = [1, 2, 3, 4]$. therefor

$$P(4) = \left(\sum_{i=1}^{4} \left(\frac{t^2(i)}{2} \right)^2 \right)^{-1} = 0.0408$$

$$g_1 = \hat{\theta}_1(i = 4) = P(4) \left(\sum_{j=1}^{4} \left(\frac{t^2(j)}{2} \right) s(j) \right) = 9.7755. \tag{1.2}$$

b) Recurcive Least Squares for calculate one forward step with new point $s(5) = 78.5$ and $t(5) = 4$:

$$P(5) = \left(P^{-1}(4) + \varphi(5)\varphi^T(5) \right)^{-1} = 0.0156$$

$$g_2 = \hat{\theta}_2(5) = \hat{\theta}_1(4) + P(5)\varphi(5) \left(s(5) - \varphi^T(5)\hat{\theta}_1(4) \right) = 9.8125. \tag{1.3}$$

Chapter 2

Object Oriented Programming (OOP)

2.1 Introducing OPP

2.1.1 Class

2.1.2 Object

2.2 Inheritence

2.3 Examples

In Eq (1.1) if

$$y(t) = \phi^T(t)\theta(t) + e(t) \tag{2.1}$$

such that, $e(t)$ is a white noise with zero mean and variance σ^2, then

$$\hat{\theta}(i) = \theta + \left(\sum_{j=1}^{i} \varphi(j)\varphi^T(j)\right)^{-1} \left(\sum_{j=1}^{i} \varphi(j)e(j)\right).$$

therefore,

$$E\{\hat{\theta}\} = \theta. \tag{2.2}$$

On the other hand we know that

$$\text{var}(\hat{\theta}) = E\{\hat{\theta}^2\} - E^2\{\hat{\theta}\}. \tag{2.3}$$

Proof. We expand the given equation on both side:

$$\begin{aligned} E\{(\hat{\theta} - \theta)^2\} &= E\{\hat{\theta}^2 + \theta^2 - 2\theta\hat{\theta}\} \\ &= E\{\hat{\theta}^2\} + E\{\theta^2\} - 2E\{\theta\hat{\theta}\} \\ &= \theta^2 + E\{\hat{\theta}^2\} - 2\theta E\{\hat{\theta}\}. \end{aligned} \tag{2.4}$$

$$\begin{aligned} bias^2 + var &= (\theta - E\{\hat{\theta}\})^2 + E\{\hat{\theta}^2\} - E^2\{\hat{\theta}\} \\ &= \theta^2 + E\{\hat{\theta}^2\} - 2\theta E\{\hat{\theta}\}. \end{aligned} \tag{2.5}$$

The both side of equation are equal, so the proof is complete. □

Chapter 3

Fundamental of Dynamical Systems

3.1 Ordinary Differential Equations (ODEs)

3.1.1 Linear Time Invariant (LTI)

LTI systems without input delay

LTI systems with delayed inputs

3.1.2 Nonlinear ODEs without input delay

3.1.3 Nonlinear ODEs with delayed inputs

3.2 Partial Differential Equations (PDEs)

3.3 Examples

For estimation the parameters, using the 'ARX' command in MATLAB, we achieve the Table 3.1. Model 4 is better than other models, because MSE is small and Fit percent is bigger than others. But why model 5 is not good? It is not a good model because $a_2 \simeq 0$ and there is not much difference between model 4 and model 5. therefore model 4 with fewer poles is a good select for Representation system.

Table 3.1: Estimated Parameters with LS method

Model	a_1	a_2	b_1	b_2	MSE	Fit(%)
1	-	-	0.9078	-0.9212	0.3727	58.87
2	-0.7390	-	0.8832	-	0.1705	72.18
3	-0.9856	-0.3485	0.9315	-	0.04384	85.89
4	-0.5229	-	0.9033	-0.4809	0.003321	96.12
5	-0.525	-0.001746	0.9035	-0.479	0.00332	96.12

Chapter 4

MATLAB framework

4.1 Introduction of the models structure

The provided framework is shared to help scientists working on system engineering, control, computational neuroscience, robotic, etc. This package contains several simple files that connect to each other like diagram illustrated in Fig. 4.1. As it is clear from Fig. 4.1, '`main.m`' is the file which must be called to run the simulation. Note that running other files is not recommended, although it can be usefull in some situations. There are 5 called functions in the '`main.m`' file which will be elaborated in the following.

4.2 Framework files and directories

4.2.1 Basic files

The group of all files exist in the main root is named basic files, such as '`main.m`', '`simulation.m`', and so forth.

'main.m'

If you are a person who want to run the simulation once, it just need to write '`main`' in the command window. However, if the project is developed to simulate several times, you should make a structure variable of necessary parameters, e.g. '`extOpt`', and insert it into the '`valuation(extOpt)`' function existed in the main file. After that, by calling '`main.m`' using the expression '`run main.m`', you can have a control

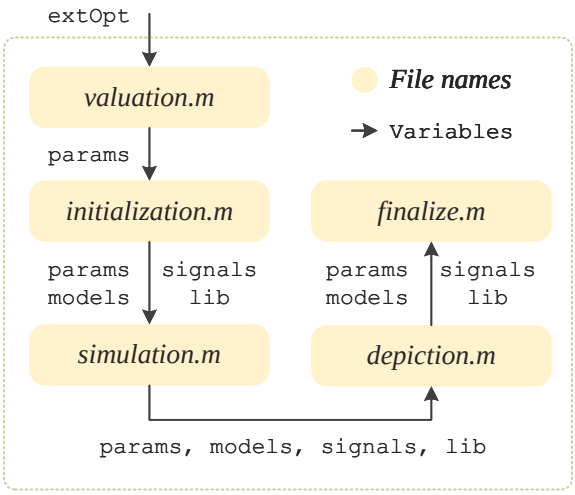

Figure 4.1: The internal blocks of '`main.m`' file.

```
function params = valuation(extOpt)
    % [Parameters] <- (External variables as a structure variable)
    % extOpt is used for external control on whole simulation by a
    % second m file. If you want to import data here, make a structure data and
    % put it into the 'validation()' function in 'main.m'.

    %% Simulation parameters
    % Tho below variables are time-step and simulation time, respectively.
    % You might need to modify them arbitrarily.
    params.step = 0.001; %(double)
    params.tOut = 9;     %(double)

    % The given line below is a dependent variable. You normally
    % should NOT change it.
    params.n = floor(params.tOut/params.step); %(DEPENDENT)

    % The two next variables carry the folders from which you can use to
    % call your fales and saving the results organizely.
    params.loadPath = 'data/inputs';  %(string)
    params.savePath = 'data/outputs'; %(string)

    % Do you want to save a diary after each simulation? So set the below logical
    % variable "true". The below directory is the place your logs are saved.
    params.makeDiary = true;   %(logical)
    params.diaryDir  = 'logs'; %(string)

    % The amount of time (in second) between each commands printed on CW.
    params.commandIntervalSpan = 5; %(double)
```

Figure 4.2: The content inside the '`valuation.m`'.

to your main file for multi-run simulations. This command can be used in a '`for`' loop in your mother file to run the simulation, get and save the results, repeatedly. Note that, it might be helpful if '`close all; clear; clc;`' expression in '`main.m`' is removed, but it can dependent on your project purposes.

'valuation.m'

The first function named '`valuation.m`' is considered to collect all static variables such as simulation time and step-time. You can define your constant variables in this function in a structure variable named '`params`'. For example, to set a constant value, which is represents π, you have to write '`params.pi = 3.14;`' into the function '`valuable.m`'. Figure 4.2 is an image of this function.

'initialization.m'

The second file, which is considered as a part of initial tasks before the main simulation, can be undoubtedly mentioned as '`initialization.m`'. The same as '`valuation.m`', it carries kinds of data that is always used in the whole project, while it is not important to have constant values. In other words, here, we usually

```
function [params, models, signals, lib] = initialization(params)
    % [Parameters, Models, Signals, Libraries] <- (Parameters)
    % This function is created to support your signals, systems,
    % extra functions, and all initialization processes before
    % the main simulation.

    %% Loading Libraries
    lib.func = libraries.functionLib();
    lib.draw = libraries.plotToolLib();
    lib.mfun = libraries.yourLib();

    %% Simulation Time Vectors
    signals.tLine = 0:params.step:params.tOut;


    %% Your Personalized Section
    %  Write your codes here ...


    %% Updating
    %  If you need to update or create some new parameters, finalize them here.
    models.updated = 1;
    params.updated = 1;
end
```

Figure 4.3: The content inside the '`initialization.m`'.

define vectors and signals in a structure variable named '`signals`'. Also, all synamic systems, filters, estimators, trackers , etc, must be define in this file in a uniqe variable which is '`models`'. Furthermore, in some projects, several fixed parameters must be set after some initializations. To satisfy this, the parameter structure '`params`' can be change at the end of '`initialization.m`' file. In addition, some useful libraries (Groups of functions) are added in this file which will be explained later. To sum up, This function receives '`params`' as its input variables and after an initial process, four structure variables named '`params`', '`models`', '`signals`', and '`lib`' are sent to the out. Figure 4.3, illustrates a view of this function.

'simulation.m'

It should be mentioned that this file is the most important file in which you have to plan and control your main idea. In other words, the main loop of your simulation has been written in '`simulation.m`'. All variables that are resulted from '`initialization.m` directly came to this file and all of them finally are led to the output. The first part of this file is just before the loop which you can prepare your temporary initial variables that might be used in the loop. To do this, write your codes at the end of '`Initial options`' part, just before the '`for`' loop. The counter variable of this loop, which is '`k`', is a '`double`' variable that starts from zero and is increased up to '`params.n`', which is the number of simulation samples. It is clear that this variable depends on the simulation time and step-time which is set in

```
function [params, models, signals, lib] = simulation(params, models, signals, lib)
    % [Parameters, Models, Signals, Libraries] <- (Parameters, Models, Signals, Libraries)
    % This file is your main core of simulation which there is a time-loop
    % Also, the model blocks and the signals must be updated here.
    % Before the main loop, you can initialize if you need, Also you can
    % finalize after that if you need, as well.

    %% Initial options
    func = lib.func;
    func.sayStart();
    % A trigger used to report steps in command
    trig = [2, 0];

    %% Main loop
    for k = 1:params.n
        % Displaying the iteration number on the command window
        trig = func.disit(k, params.n, trig, params.commandIntervalSpan);


        %% Write your codes here ...


    end

    %% Finalize options
    % To report the simulation time after running has finished
    func.disit(k, params.n, [0, trig(2)], params.commandIntervalSpan);
    func.sayEnd();
end
```

Figure 4.4: The content inside the '`simulation.m`'.

'`valuable.m`'. At the begining of the loop, there is an order that is put to make a report of loop condition. To be exact, it prints a line into the command windows that contains the current sample '`k`', the number of all samples, the progress percentage, and expecting time to the end of the simulation. This report is normally printed each 5 seconds and can be helpful to know how much of the process has done and how much we must be waiting to be completed. The amount of this span is under your control in variable '`commandIntervalSpan`' that is defined in '`valuation.m`' function. After the '`for`' loop, we can see a part named '`Finalize options`', which are utilized to report total simulation time, although you can exploit that for your purposes. A case of this function is demonstrated in Fig. 4.4, and Fig. 4.5 shows the final situation of command window after ending a program.

'depiction.m'

Do you need to make your data graphical? If you do, here is all you need to plot and depict your novels. As it can be clearly seen that in Fig. 4.6, which is a shot of this function, all quadruple variables are ready to be used here and by thanks of its outputs, it is also possible to modify them, although it is not really common to make any changes after the simulation stage. In addition, one more variable defined at the end of input arguments whose name is '`plotOrNot`'. This is beneficial for you, in case of ignoring the depiction section. To do this, it must be set '`false`' in the '`main.m`' file. Note that the default value of this logical variable is '`true`' which means the illustrator section is enabled. As the same as before, this function

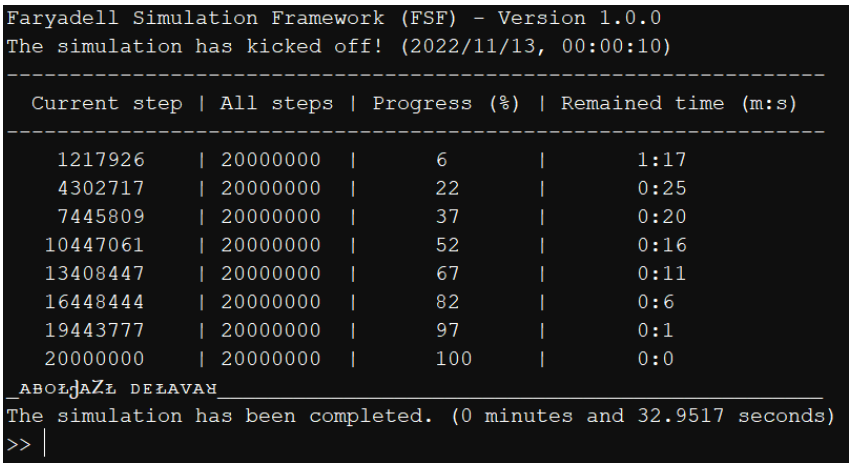

Figure 4.5: Outputs of a project that printed in the command window. As it is distinct, the simulation ran about 50 seconds.

is splitted into three parts which named '`Initialize`', '`Main part`', and '`Finalize`', respectively. The first and last part are used to define initial variables and finalize the plot section. The main part of the file that can be mentioned is the middle part. All MATLAB functions related to graphic, and also a practical library called '`plt`' are ready to be exploited there. Finally, there is nothing more especial here, and the most effective parts of this framework which concentrates on graphics will be elaborated in libraries and observer parts.

'finalize.m'

The last file of the main root is '`finalize.m`' to be utilized for saving obtained data and report some pieces of them on the command window. This function is almost empty at the first, except the order which turn the started diary off, but you can call all variables which is available into that to code your remained ideas. Figure 4.7 is a simple shot of '`finalize.m`' function.

```
function [params, models, signals, lib] = depiction(params, models, signals, lib, plotOrNot)
    % [Parameters, Models, Signals, Libraries] <- (Parameters, Models, Signals, Libraries, Plo
    % All of your plots and depiction objects shoud be coded here
    % To use this, you have initialize, main, and finalize sections.

    if plotOrNot == 1
        %% Initialize
        plt   = lib.draw;           % Import all plotting functions
        n     = params.n;
        nn    = 1:n;                % A vector from 1 to n
        tLine = signals.tLine(nn);  % A time-line vector


        %% Main part
        %  Insert your ideas here ...


        %% Finalize
        %  If you want to save images and graphs use this section to save
        %  them. All variables will be returned to the 'main' automatically,
        %  so you can change any parameters if you need.

    end
    % The end of 'depiction' function
end
```

Figure 4.6: A shot of '`depiction.m`' file.

```
function finalize(~, ~, ~, ~)
    % [] <- (Parameters, Models, Signals, Libraries)
    % If you have ideas about saving data and reporting the final results on
    % the command window, here is the most appropriate place to do this.
    % All data and libraries have imported to this function, just call them and use.


    %% Write your codes here ...


    %% FINAL SETTINGS
    diary off;  % Turning the diary off
end
```

Figure 4.7: A shot of '`finalize.m`' function.

4.2.2 Runner classes

'LTISystem.m'

'nonlinearSystem.m'

'neuronGroup.m'

'estimator.m'

'recursiveLeastSquare.m'

4.2.3 Observers and detectors

'scope.m'

4.2.4 Libraries

'functionLib.m'

'plotLib.m'

'yourLib.m'

4.2.5 System blocks

Chapter 5

A Summary of The LaTeX Framework