School of Electrical Engineering & Telecommunications

ELEC4632 Lab 1

An introduction to linear least square method and system identification

In this lab, you will use linear least square method to identify a model for an input/output data collected from a physical system.

Pre-lab Exercise: MATLAB Refresher

You should write a MATLAB code (m-file) to create a set of sinusoidal data and plot them exactly as shown in Fig. 1, through the following steps.

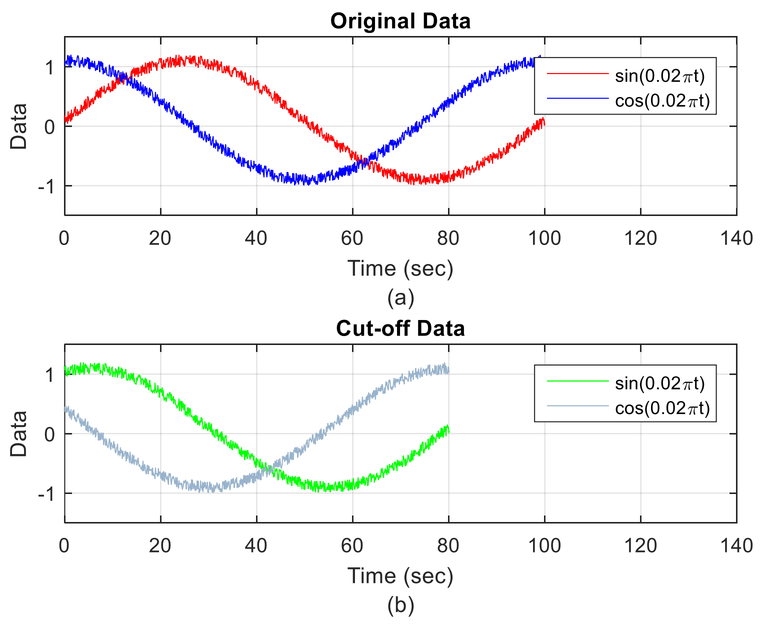


Fig. 1. Comparison of a set of sinusoidal data, (a) Original data, (b) Cut-off data.

1. Create a column vector representing time from 0 to 100 seconds with time spacing of 0.1 second and name it `t`. Use either `linspace` function or use direct vector definition like `t = a:b:c`. Make sure to transpose it as this form of variables are row vectors in MATLAB by default. Type in `help + function_name, such as linspace,` in MATLAB command window to learn how to use the function, and you can also see full help on any function by searching them in the main help page of MATLAB. Do not know which function can achieve your objective? Google is a good searching tool.

2. Use `sin` and `cos` functions to generate two sinusoids, `y1` and `y2`, respectively, with a period of 100 seconds, and add a uniformly distributed noise using `rand` function with the bound of ±0.2.
3. Cut off the first 200 samples from all data and assign them to new variables `t_new`, `y1_new`, and `y2_new`, respectively. Make sure `t_new` starts from zero. You can extract part of a data stored in a variable by using `colon` operator in indexing rows and columns, i.e., `A(a:b,:)` would extract data in matrix `A` from row `a` to row `b` for all columns (recall that `A(c,d)` in MATLAB returns the element stored in row `c` and column `d` of `A`)
4. Store these three cut-off vectors under a new matrix variable named `data_new`. Use `data_new = [t_new y1_new y2_new]` to attach column vectors with the same size and create an n-by-3 matrix, i.e., `data_new`$_{n\times3}$. This method is known as *matrix concatenation*.
5. Plot the original data you created in Step 2 against time, i.e., `y1` versus `t` and `y2` vs `t`, on the top side of the figure with the same colors as shown in Fig. 1(a) using the following functions: `figure`, `subplot`, `plot`, `stairs`, `hold on`, `hold off`, `xlim`, `ylim`, `title`, `grid`, `xlabel`, `ylabel`, `legend`. To display Greek letters, like π, read the *Text Properties* in MATLAB help. Limit x-axis and y-axis between (0, 140) and (−1.5, 1.5).
6. Use the same functions above (except `figure`) to plot the cut-off data at the bottom of the figure as shown in Fig. 1(b). The color code for `cos(0.02*pi*t)` is `[0.6 0.7 0.8]` (see the help for `plot` and RGB color codes)

Introduction to System Identification

In this lab, you will learn how to determine or identify a suitable dynamic model for a process using linear least squares method [1], [2]. In control systems theory, to design a control system for a process using a so-called Model-Based Control method, a mathematical model of the process is needed. In general, there are two methods to model a process (also known as *system identification*). One is by using the physical relationships that describe the dynamics of the process. For instance, using Newton's laws to obtain equations of motion for a mass-spring-damper process, or using Kirchhoff's Voltage and Current laws in addition to Newton's laws to derive a permanent magnet DC motor equation [3]. This method can sometimes result in complex dynamic equations for complicated processes. Another way of finding a dynamic model for a process is by using experimental input and output data, and then trying to match them with a mathematical equation, either linear or nonlinear, depending on the nature of the system and its operating conditions. This method is useful for the cases where there is little information about the physics of the process, or due to the limitation in accessing different parts of the process for parameter measurements, only input and output signals are available to be used. We all know that a process would reveal its dynamic characteristics through its outputs and states variables if a proper input signal is applied to excite all internal modes of the process.

One of the most common methods of empirical system identification in industry using input/output data is the so-called **Step Response** modelling. This method is suitable for processes with slow dynamics which are mostly controlled for regulation purposes or set-point control. An example is provided in Appendix section showing how to find a first order model using a recorded data in continuous-time domain.

Parametric System Identification Using Linear Least Squares Method

Most of the processes in industry are controlled in a way to keep the systems near their operating points, and as a result, they can be represented by linear differential or difference equations (rational transfer function). Therefore, we can determine the finite number of parameters that characterize such a dynamic model, i.e., polynomial coefficients or zeroes and poles. This method is known as *parametric system identification*. However, it is not always possible to model these processes with a first-order transfer function using one simple step response as discussed before, since their dynamics can be more complicated. Therefore, by choosing a suitable input signal and recording the corresponding output using a digital computer, we can determine *causal* discrete-time models[1], as a dynamic model for a process to be controlled. The reason for identifying the models in discrete-time form is that the control system will eventually be implemented on a digital computer, so it would be beneficial to have a discrete-time model of the process and design the control system directly in discrete-time domain, even though the real nature of the process is in continuous-time domain. In this lab, our focus is on *second-order discrete-time models* with strictly proper form[2]. It should be mentioned that, when no information is available about the process, we need to vary the order of the model to get the best approximation of the actual process in terms of closeness of the responses.

Linear discrete-time dynamic models in second-order form are given as follows,

- Transfer function in *Z* domain (z^{-1} is called "delay operator"):

$$G(z) = \frac{Y(z)}{U(z)} = \frac{b_1 z + b_2}{z^2 + a_1 z + a_2} = \frac{b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}. \tag{1}$$

- State Space:

$$\begin{cases} x(k+1) = Gx(k) + Hu(k) \\ y(k) = Cx(k) + Du(k) \end{cases}, \quad G = \begin{bmatrix} 0 & 1 \\ -a_2 & -a_1 \end{bmatrix}, \quad H = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} b_2 & b_1 \end{bmatrix}, \quad D = 0. \tag{2}$$

(*Canonical Controllable Form*)

- Difference Equation:

$$y(k) + a_1 y(k-1) + a_2 y(k-2) = b_1 u(k-1) + b_2 u(k-2). \tag{3}$$

Thus, the unknown parameters of the process model to be determined are {a_1, a_2, b_1, b_2}. In order to estimate these parameters, the so-called *linear least squares method* is used in this lab, which is a well-known and commonly used method for this purpose. More details on least squares method can be found in [1] and [2]. From the three different discrete-time models above, difference equations in Eq. (3) is more suitable since it can be written as a set of linear equations containing discrete-time values of the measured input/output data. Later on, for the purpose of controller design, state space model will be used primarily. Thus, the difference equation can be rearranged as follows,

---

[1] A casual system (also known as a physical or non-anticipative system) is a system whose output depends on previous and current values of input, and perhaps previous values of output [3].

[2] A strictly proper form for a transfer function means that its numerator's order is less than the order of denominator, i.e., less zeros than poles [3]. In discrete-time domain, it means that output depends only on previous values of input and output.

$$y(k) = -a_1 y(k-1) - a_2 y(k-2) + b_1 u(k-1) + b_2 u(k-2),$$

$$y(k) = \underbrace{\begin{bmatrix} y(k-1) & y(k-2) & u(k-1) & u(k-2) \end{bmatrix}}_{\varphi^T(k)} \underbrace{\begin{bmatrix} -a_1 \\ -a_2 \\ b_1 \\ b_2 \end{bmatrix}}_{\theta} \quad \Rightarrow \quad y(k) = \varphi^T(k)\theta. \tag{4}$$

The model $y(k) = \varphi^T(k)\theta$ is called ***regression model*** and the variables inside $\varphi^T(k)$ are called ***regression variables*** or ***regressors***. They contain the previous values of the input and corresponding output values at time steps $k - 1$ and $k - 2$ (k is the discrete time step related to continuous time as $t = kT_s$ for $k = 1, 2, 3 \ldots$ with T_s as the sampling time). Since we use computer to apply input signal with a fixed sampling rate and then measure the corresponding output, N linear equations are constructed, as shown in Fig. 2, with only four unknown parameters for a second-order model, where N is the number of samples.

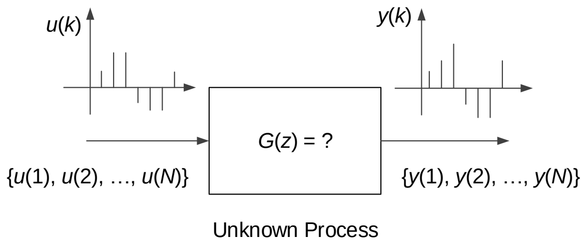


Fig. 2. System identification with input/output data.

Therefore, the problem of solving a set of linear equations is shown as below,

$$\begin{cases} y(1) = \begin{bmatrix} y(0) & y(-1) & u(0) & u(-1) \end{bmatrix}\theta \\ y(2) = \begin{bmatrix} y(1) & y(0) & u(1) & u(0) \end{bmatrix}\theta \\ y(3) = \begin{bmatrix} y(2) & y(1) & u(2) & u(1) \end{bmatrix}\theta \\ \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ y(N) = \begin{bmatrix} y(N-1) & y(N-2) & u(N-1) & u(N-2) \end{bmatrix}\theta \end{cases} \tag{5}$$

$$\Rightarrow \underbrace{\begin{bmatrix} y(1) \\ y(2) \\ y(3) \\ \vdots \\ y(N) \end{bmatrix}}_{Y} = \underbrace{\begin{bmatrix} y(0) & y(-1) & u(0) & u(-1) \\ y(1) & y(0) & u(1) & u(0) \\ y(2) & y(1) & u(2) & u(1) \\ \vdots & \vdots & \vdots & \vdots \\ y(N-1) & y(N-2) & u(N-1) & u(N-2) \end{bmatrix}}_{\Phi} \underbrace{\begin{bmatrix} -a_1 \\ -a_2 \\ b_1 \\ b_2 \end{bmatrix}}_{\theta} \tag{6}$$

$$\Rightarrow \; Y = \Phi\theta. \tag{7}$$

As can be seen in Eq. (7), vector Y and matrix Φ contain all recorded input and output information, and θ is the vector of unknown model parameters to be found. The result is similar to the case where we are dealing with solving a set of linear equations having more unknown variable than the number of equations, which indicates that there is no unique solution to this problem. Least squares method can offer the best solution for model parameters $\{a_1, a_2, b_1, b_2\}$ in the sense of minimum error between the left hand side and the right hand side of those linear equations. Moreover, we know that the real process is not perfectly linear, and the actual measurements $y(k)$ in each time step does not have a linear relationship with the previous measurements and the input values as the model implies, i.e., $Y \approx \Phi\theta$. Thus, the error between the actual output values and the model output values, i.e., $E = Y - \Phi\theta$, includes both measurement noise and model uncertainty as well. Finally, **the least squares solution that gives the best estimate for model parameters $\hat{\theta}$ is obtained as follows**,

$$\hat{\theta} = (\Phi^T \Phi)^{-1} \Phi^T Y \tag{8}$$

The obtained solution $\hat{\theta}$ in Eq. (8) is the best estimate of the model parameters in the sense of least squares (note that $\theta = [-a_1 \ -a_2 \ b_1 \ b_2]^T$ whereas we seek $[a_1 \ a_2 \ b_1 \ b_2]^T$). It is clear, however, that the main condition for solving the above matrix equation (also known as *normal equations*) is that the matrix $\Phi^T\Phi$ has full rank and it is positive-definite (*well-conditioned*), which is known as *Persistent Excitation* condition. More details on least square solution in Eq. (8) are provided in Appendix.

Selection of a Suitable Input

So far, we have learned how to find the best estimate of coefficients for a system model using linear least squares. In the next step, we want to know how to perform data collection for system identification. The most recommended types of input signals for system identification are *Pseudo Random Binary Signal (PRBS), Square-Wave Signals*, or the sum of many sinusoidal waves. The amplitude of the input signal should be bounded to keep the system in its linear operation region. It should be mentioned that any process has constraints on its input in terms of the maximum range of the input amplitude allowed to be applied before the process is damaged. In addition, the resulting output $y(k)$ should be much larger than measurement noise. Otherwise, the output would not be usable for system identification and it might create a so-called *ill-conditioned* $\Phi^T\Phi$; and therefore, the persistent excitation condition is no longer met.

DC Offset Compensation

The output of a linear time-invariant (LTI) system is always zero when zero input is applied (assuming zero initial conditions). Thus, if a system responds to zero input with the nonzero value, the system is called to have *offset* value in its output (DC offset), which probably exists in industrial processes. The other form of offset in systems is when the input range is limited to only positive or negative values. In this case, the value in the middle of the input range is called *input offset*, and the corresponding output to this input is called *output offset*. In both above-mentioned cases, the offsets should be removed from both input and output data before filling the matrix Φ to be able to find an LTI model accepting both negative and positive input values. However, in the actual control operation, the control input $u(k)$ should be calculated from offset-free output (output offset y_{offset} should be subtracted from the measured output $y_m(k)$ to be used in the computation of $u(k)$). Then, input offset u_{offset} should be separately added to the calculated control input to find the actual process input $u_{in}(k)$ as shown in Fig. 3.

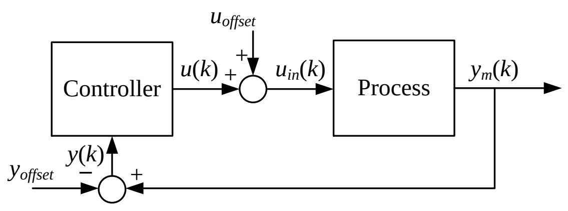


Fig. 3. Compensation of the input and output offsets.

Model Verification

After performing data collection and then identifying the unknown parameters of a process model using linear least squares, it is time to verify the obtained dynamic model to see whether it demonstrates similar behavior to the actual process or not. Hence, the same input signal should be applied to both the model and the process, and then the output responses should be compared as illustrated in Fig. 4. It is preferred to use a different set of input/output data for validation than those used for system identification. The smaller the error between the output responses, the better the quality of the identified model will be. One good criterion for verification of the identified model is the application of *Mean Squared Error* method (MSE) on the difference between output responses or the error signal.

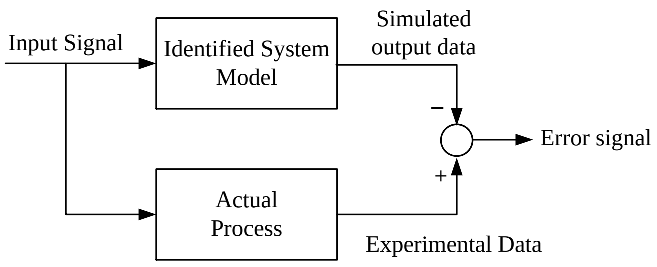


Fig. 4. Model verification after performing system identification.

Lab Exercise (2 marks)

In this lab, you are going to identify a second-order discrete-time linear model as in Eq. (1)-(3) using a pre-collected data from one of the W-T setups through the following steps,

1. **<u>Data extraction and analysis (0.6 marks, checked at 45 minutes)</u>**

   a. Download the pre-collected data from Moodle and save it in the current directory of MATLAB. The data file is named `SysIdenData_StudentVersion.mat`. Load the data into MATLAB Workspace using `load` function. The loaded data should appear in Workspace with the name `LogData` in Structure format. Use the following code to extract individual data in vector array form,

   ```
   t = LogData.time;

   y_act = LogData.signals(1).values(:,2);

   y_actm = LogData.signals(1).values(:,1);

   u_act = LogData.signals(2).values;
   ```

   As you can see, the data contain recorded time in `t`, actual noise-reduced output in `y_act`, original measured output in `y_actm` and actual input data in `u_act`. Then, find the sampling time for which the data were recorded and name it `Ts` or `h`.

   b. Plot these data using the functions you practiced in Pre-lab exercise as shown in Fig. 5.

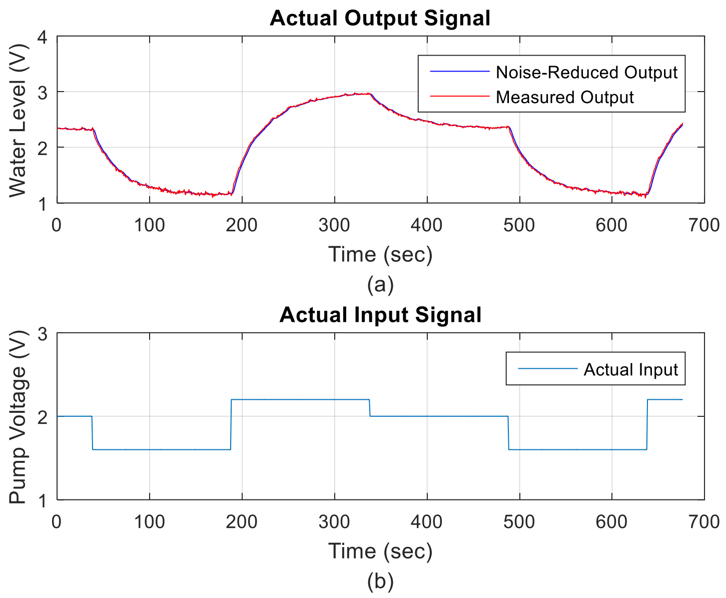


Fig. 5. Original data, (a) Comparison of noise-reduced and measured output signals, (b) Actual input signal.

c. Find the best estimate of the offset value from the noise-reduced output `y_act` and name it `y_offset`. Then, remove the output offset by subtracting `y_offset` from `y_act` and name it `y`. Find input offset as well and name it `u_offset` and remove it from input data `u_act` and name it `u`. Then, plot them in one figure as shown in Fig. 6. As you can see, both offset-free input and output signals in Fig. 6(a) and (b), respectively, begin from zero.

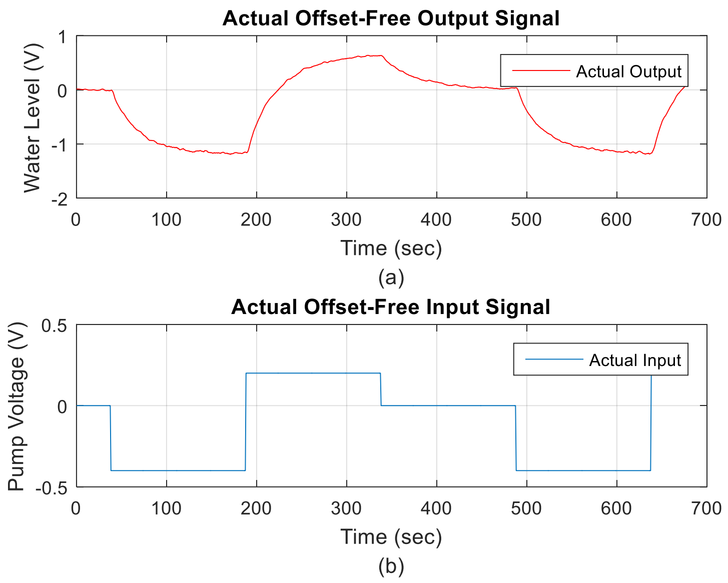


Fig. 6. Offset-free data, (a) Offset-free output signal (noise-reduced), (b) Offset-free input signal.

Hint: As explained in *DC Offset Compensation* section, for a set of input-output data which only contain positive values, like `u_act` and `y_act` here, offset values have to be detected and removed from the data before filling the matrix Φ. If no information is given about the range of input signal, we would assume the first value of the input signal is input offset as you can clearly see in Fig. 5(b). However, we cannot simply consider the same fact for output offset as there is always some noise in the output signal, even in the noise-reduced one `y_act` here. The best approach to estimate the closest value to actual output offset is by taking average from the first period of output data which corresponds to the first period of input data before the first change. Thus, you have to write a loop in MATLAB to automatically detect the number of samples in the first period of output to be used in average. Use `while` function for loop and `mean` function for average.

2. **<u>Identifying a second-order discrete-time linear model (0.6 marks, checked at 1 hour 30 minutes)</u>**

   a. Create matrix Φ as shown in Eq. (6) using the first half of the offset-free input-output data. If starting from `k = 1`, you need to choose the values for `y(−1)`, `y(0)`, `u(−1)`, and `u(0)` as initial conditions. They should be chosen rationally. You do not necessarily need to start from `k = 1`. You can choose to start from some samples ahead, i.e., starting from `k = 10`, for example. You can use *matrix concatenation* to create matrix Φ as explained in pre-lab exercise.

   b. Find the solution of least squares method $\hat{\theta}$ as given in Eq. (8), to obtain the estimate of the second-order discrete-time model parameters $\{a_1, a_2, b_1, b_2\}$.

   c. Create a second-order discrete-time transfer function and state space representation of the identified model in MATLAB, as given in Eq. (1) and (2), and display them on Command Window using `tf` and `ss` functions.

3. **<u>Model verification and simulation (0.8 marks, checked at 2 hours 30 minutes)</u>**

   a. Using the transfer function or state space model you defined in previous part, simulate the identified model, which means finding the response of the identified model to the offset-free input `u` as explained in *Model Verification* section.

   b. You should simulate the model and plot the results, first by using the second half of the offset-free input `u` and compare the simulated output with the second half of the offset-free output `y`, and then, by using the entire offset-free input `u` and compare the simulated output with the entire offset-free output `y` as shown in Fig. 7(a) and (b), respectively. Use `lsim` or `filter` functions to find the simulated response. You could try Simulink to simulate the model too to practice on Simulink too.

   c. In Fig. 7(a), can you explain why the simulated output does not start from the same point as the offset-free output `y`?

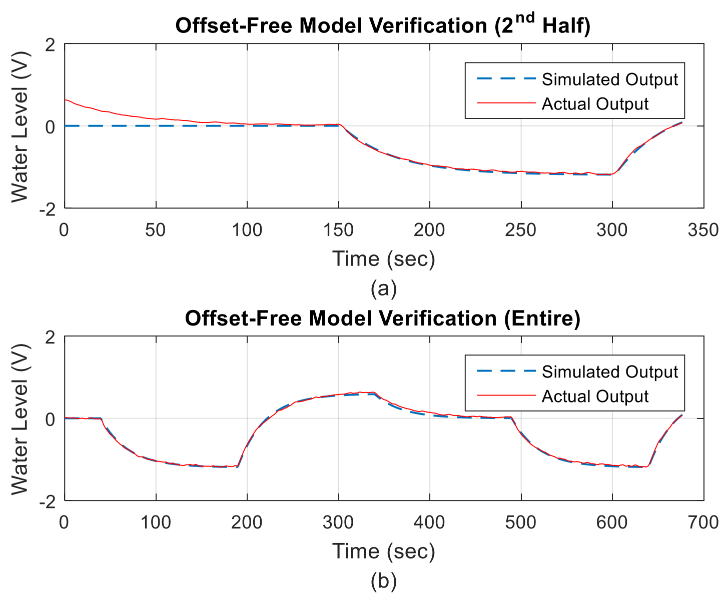


Fig. 7. Model verification, (a) comparison of 2nd half of the simulated output with the 2nd half of actual offset-free output, (b) comparison of the entire simulated output with actual offset-free output.

Post-lab Exercise

In this exercise, we want to compare the effect of different model orders in the accuracy of system identification.

1. Follow the same procedure you did in Exercise 2 to identify a first-order model using offset-free data `u` and `y`. The structure of a first-order difference equation is given as below,

$$y(k) = -a_1 y(k-1) + b_1 u(k-1). \tag{9}$$

2. Simulate the first-order model using the entire offset-free input and compare its simulated output with the second order one you obtained before and the offset-free output `y` as shown in Fig. 8.

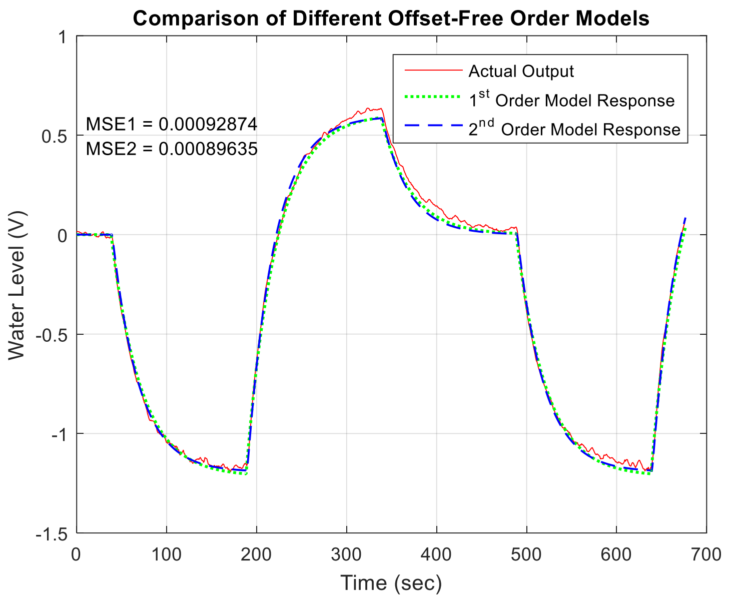


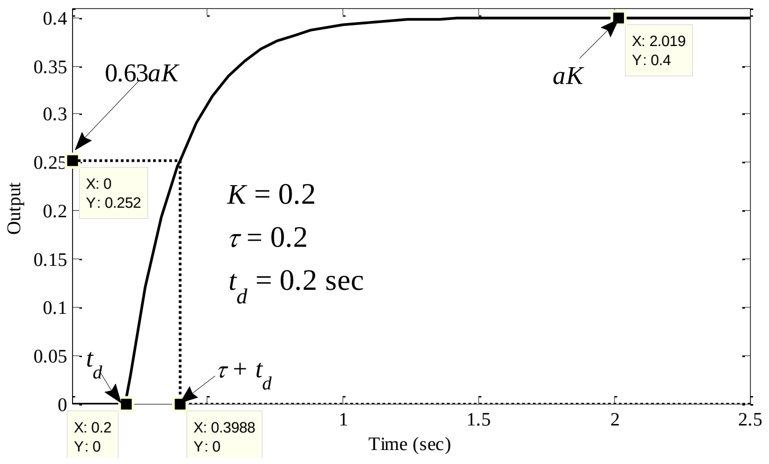
Fig. 8. Comparison of different order models with the actual offset-free output for accuracy purposes.

3. Use *mean squared error* method to numerically compare between these two models as can be seen in the top-left corner of Fig. 8. Which model would you choose for controller design and why?

Appendix

In step response modelling, the process is approximated by a first-order or second-order transfer function mostly with pure delay **Error! Reference source not found.**. As an example, if the response of a process to a step input with the amplitude of *a* = 2 is given as in Fig. A1, we can approximate this process with a first-order transfer function *G*(*s*). The unknown parameters are gain *K*, time constant *τ*, and time delay t_d which can be easily computed as in Fig. A1.

In this example, it is interesting to see that the original system was a second-order transfer function as shown in Fig. A2(a). The approximate first-order transfer function outputs a similar response with a high accuracy to the original system response as illustrated in Fig. A2(b). However, if the response has oscillations, second-order or higher-order models should be considered and the unknown parameters should be determined using different approaches.



$$G(s) = \frac{Ke^{-t_d s}}{1+\tau s}, \quad U(s) = \frac{a}{s}$$

$$Y(s) = G(s)U(s) = \frac{Ke^{-t_d s}}{1+\tau s} \times \frac{a}{s}$$

$$\Rightarrow \quad y(t) = aK(1-e^{\frac{-(t-t_d)}{\tau}})$$

If $\quad t \to \infty \quad \Rightarrow \quad y_{ss} = \lim_{t\to\infty} y(t) = aK$

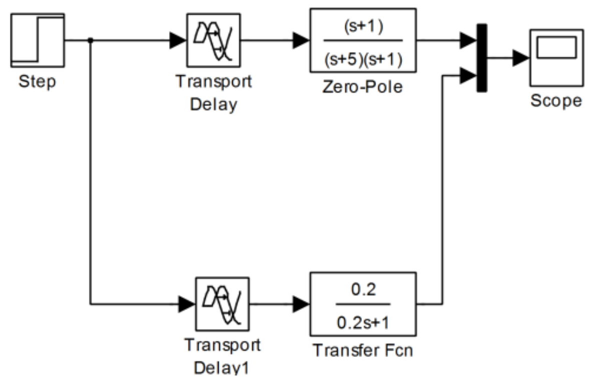
Thus $\quad K = \frac{y_{ss}}{a}$

If $\quad t = \tau + t_d \quad \Rightarrow \quad y(\tau + t_d) = 0.63aK$

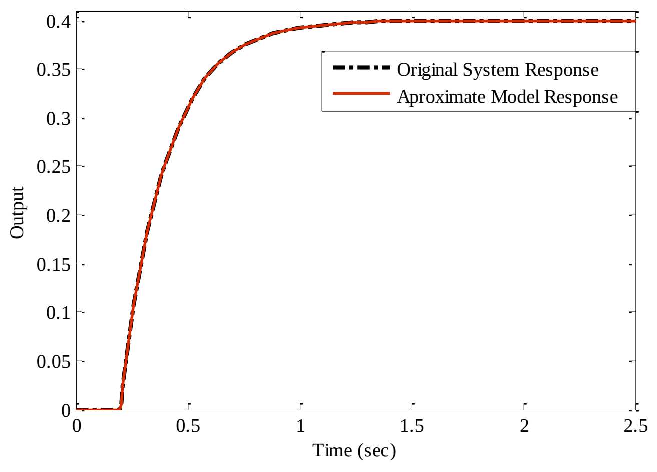
Thus $\quad \tau = y^{-1}(0.63aK) - t_d$

(Geometrically from the graph)

Fig. A1. The response of the original system to a step input and the values of the approximate first-order model.



(a)



(b)

Fig. A2 (a) Simulation block diagram, (b) Comparison of the step responses generated by original and approximate models.

Linear Least Squares Method

One of the most common methods for parametric identification is the so-called *Least Squares* (LS) method. The method of least squares grew out of the fields of astronomy and geodesy as scientists and mathematicians sought to provide solutions to the challenges of navigating the Earth's oceans during the age of exploration. The accurate description of the behavior of celestial bodies was the key to enabling ships to sail in open seas, where sailors could no longer rely on land sightings for navigation (source: Wikipedia). In this lab, our focus is on the linear least squares method in mathematics rather than statistics known as *linear regression*, even though both have the same concept but different interpretations.

Mathematically, linear least squares method is mostly applied in the problem of approximately solving a set of linear equations where there are more equations than the unknown variables or parameters. The best approximation is obtained by minimizing the sum of squared differences between the data values (right-hand side of the equations) and their corresponding modelled values (left-hand side of the equations). The approach is called “linear” least squares since the assumed equations to be estimated are linear in the parameters. Linear least squares problems are convex and have a closed-form solution that is unique, provided that the number of data points used for constructing the equations are equal to or more than the number of unknown parameters. One of the most common application of linear least squares is in curve-fitting problems, where a set of data point from an observation or experiment are fitted with a curve having a linear-in-parameter form. To see how linear least squares method works and understand it better, consider the following simple problem of solving two linear equations with two unknowns,

$$\begin{cases} a_1u_{11} + a_2u_{12} = y_1 \\ a_1u_{21} + a_2u_{22} = y_2 \end{cases}. \tag{A1}$$

What are the possible answers for a_1 and a_2 that satisfy both equations if u_{ij} and y_i are known and belong to real numbers set (for i and j = 1, 2)? From linear Algebra, we know that if the equations are linearly independent, there exists a unique solution as below,

$$\begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix}\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix}^{-1}\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \quad \Rightarrow \quad \begin{cases} a_1 = \dfrac{u_{22}y_1 - u_{12}y_2}{u_{11}u_{22} - u_{12}u_{21}} \\ \\ a_2 = \dfrac{u_{11}y_2 - u_{21}y_1}{u_{11}a_{22} - u_{12}u_{21}} \end{cases}, \tag{A2}$$

$$\det\begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{pmatrix} = u_{11}u_{22} - u_{12}u_{21} \neq 0.$$

If the above equations are linearly dependent (determinant equals zero), no specific solution could be found (one variable can be chosen randomly/arbitrarily in order to find the other one). Now consider the following three equations,

$$\begin{cases} a_1u_{11} + a_2u_{12} = y_1 \\ a_1u_{21} + a_2u_{22} = y_2 \\ a_1u_{31} + a_2u_{32} = y_3 \end{cases}. \tag{A3}$$

It is not usually possible to find a unique solution that satisfies all equations at the same time (there are more equations than the unknown variables/parameters a_1 and a_2). However, it is possible to find a solution that approximately solves the above equations in some best sense, i.e., the right-hand side of each equation has the closest value to its left-hand side. Thus, the *error* equations can be defined as follows,

$$\begin{cases} e_1 = y_1 - \underbrace{(a_1 u_{11} + a_2 u_{12})}_{\hat{y}_1} \cong 0 \\ e_2 = y_2 - \underbrace{(a_1 u_{21} + a_2 u_{22})}_{\hat{y}_2} \cong 0 \\ e_3 = y_3 - \underbrace{(a_1 u_{31} + a_2 u_{32})}_{\hat{y}_3} \cong 0 \end{cases} \Rightarrow \underbrace{\begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix}}_{E} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}}_{Y} - \underbrace{\begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \\ u_{31} & u_{32} \end{bmatrix}}_{\Phi} \underbrace{\begin{bmatrix} a_1 \\ a_2 \end{bmatrix}}_{\theta}, \qquad \text{(A4)}$$

$$\Rightarrow \quad E = Y - \underbrace{\Phi\theta}_{\hat{Y}} = Y - \hat{Y}.$$

Therefore, **least squares problem is defined as the best solution $\hat{\theta}$ which minimizes the sum of the squared errors e_i** (for i = 1, 2, 3). This can be written as the minimization of the following quadratic cost/loss/objective function $V(\theta)$,

$$V(\theta) = \frac{1}{2}\sum_{i=1}^{3} e_i^2 = \frac{1}{2}(e_1^2 + e_2^2 + e_3^2) = \frac{1}{2}E^T E = \frac{1}{2}\|E\|^2 \qquad \Rightarrow \qquad \hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, V(\theta) = \begin{bmatrix}\hat{a}_1 & \hat{a}_2\end{bmatrix}^T. \qquad \text{(A5)}$$

To find the solution that would give the minimum value of $V(\theta)$, you just need to take the derivative with respect to independent variable θ and then find the zeros of the resulting equation. From calculus, however, we know that $V(\theta)$ is a scalar function with two independent variables a_1 and a_2 (in the form of a vector, $\theta = [a_1 \;\; a_2]^T$). Therefore, you need to take partial derivative of the scalar cost function $V(\theta)$ with respect to vector θ, also known as *gradient* of $V(\theta)$, as follows,

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, V(\theta,k) = \underset{\theta}{\operatorname{argmin}}\, \frac{1}{2}E^T E = \underset{\theta}{\operatorname{argmin}}\, \frac{1}{2}\left((Y - \Phi\theta)^T (Y - \Phi\theta)\right)$$

$$\frac{\partial V(\theta)}{\partial \theta} = 0 \quad \Rightarrow \quad \frac{1}{2}\frac{\partial}{\partial\theta}(E^T E) = 0$$

$$\frac{1}{2}\frac{\partial}{\partial\theta}((Y - \Phi\theta)^T (Y - \Phi\theta)) = 0$$

$$\frac{1}{2}\frac{\partial}{\partial\theta}((Y^T - \theta^T\Phi^T)(Y - \Phi\theta)) = 0$$

$$\frac{1}{2}\frac{\partial}{\partial\theta}(Y^T Y - Y^T\Phi\theta - \underbrace{\theta^T\Phi^T Y}_{=(Y^T\Phi\theta)^T} + \theta^T\Phi^T\Phi\theta) = 0$$

$$\frac{1}{2}\frac{\partial}{\partial\theta}(Y^T Y - 2\theta^T\Phi^T Y + \theta^T\Phi^T\Phi\theta) = 0$$

$$\frac{1}{2}(-2\Phi^T Y + 2\Phi^T\Phi\theta) = 0$$

$$\Rightarrow \quad \hat{\theta} = (\Phi^T\Phi)^{-1}\Phi^T Y$$

Reminder from algebra:

$$\frac{\partial}{\partial x} x^T Q x = 2Qx \qquad \text{(A6)}$$

$$\frac{\partial}{\partial x} x^T Q = \frac{\partial}{\partial x} Q^T x = Q$$

for a symetric matrix Q and vector x.

References

[1] K. J. Astrom and B. Wittenmark, *Adaptive Control*. 2nd ed., Upper Saddle River, NJ: Prentice Hall, 1995.

[2] L. Ljung, *System Identification: Theory for the User*. 2nd ed., Upper Saddle River, NJ: Prentice Hall, 1999.

[3] N. S. Nise, Control Systems Engineering. 7th ed., Hoboken, NJ: Jhon Wiley and Sons, 2015.

V1 by Dr. Hailong Huang, July 2018.

V0 by Dr. Arash KHATAMIANFAR, July 2017.