GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 4550 — Control System Design — Summer 2020

Lab #7: Least-Squares Parameter Identification

Contents

1 Background Material

1.1 Linearly Parameterized Models

Given a plant model that is controllable, observable and commandable, we have seen that it is possible to design an integral controller such that the overall system is internally stable and the plant output follows constant or time-varying commands with zero steady-state error even when a constant disturbance is present. However, since this entire design process begins with a plant model, it is worth asking what procedures are available to obtain an appropriate plant model.

Parameter identification methods can be used to estimate unknown plant model parameters. The basic idea is illustrated in Figure 1, where the plant is described by a linearly parameterized model $\mathcal{Y}(t) = \mathcal{U}'(t)\mathcal{P}$; the signals $\mathcal{Y}(t)$ and $\mathcal{U}(t)$ are assumed to be derivable from measurements of $y(t)$ and $u(t)$, whereas the parameters $\mathcal{P}$ are assumed to be unknown. The objective of the parameter identifier is to process $y(t)$ and $u(t)$ so as to obtain the parameter estimate $\hat{\mathcal{P}}$.

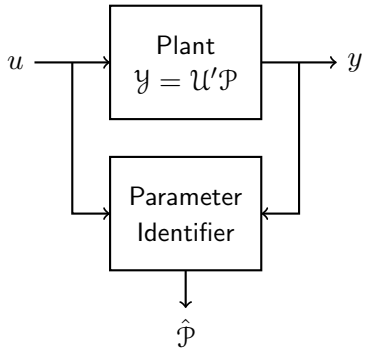

Figure 1: Block diagram illustrating the architecture of parameter identification.

To be specific, suppose our goal is to design an integral controller for a mechanical plant consisting of a mass, spring and damper. We know from physics that such a system is modeled by

$$m\ddot{y}(t) + d\dot{y}(t) + ky(t) = u(t)$$

where $u(t)$ is the force applied by an actuator and $y(t)$ is the position measured by a sensor. This differential equation provides a strong foundation for controller design, but only if we have reasonably accurate estimates of the constant parameters m, d and k.

To see how parameter identification methods work in the context of this example, note that it is possible to rewrite the modeling equation such that a vector of unknown parameters $\mathcal{P}$ appears as a linear factor in a scalar equation $\mathcal{Y}(t) = \mathcal{U}'(t)\mathcal{P}$ by defining

$$\mathcal{Y}(t) = \begin{bmatrix} u(t) \end{bmatrix}, \quad \mathcal{U}(t) = \begin{bmatrix} \ddot{y}(t) \\ \dot{y}(t) \\ y(t) \end{bmatrix}, \quad \mathcal{P} = \begin{bmatrix} m \\ d \\ k \end{bmatrix}.$$

A problem with this specific formulation is that $\mathcal{Y}(t)$ and $\mathcal{U}(t)$ depend on derivatives of measured signals. A differentiator has frequency-response function $j\omega$, so it will amplify high-frequency noise and thus lead to inaccuracies in the parameter identification process.

A remedy to the problem of differentiated measurements involves the introduction of filters. To show the value of filtering, consider the s-domain representation of the plant

$$U(s) = \left(ms^2 + ds + k\right) Y(s).$$

If we multiply both sides of this plant model by $\gamma/(s+\lambda)^3$ then we obtain

$$\begin{bmatrix} \dfrac{\gamma s^0}{(s+\lambda)^3} U(s) \end{bmatrix} = \begin{bmatrix} \dfrac{\gamma s^2}{(s+\lambda)^3} Y(s) & \dfrac{\gamma s^1}{(s+\lambda)^3} Y(s) & \dfrac{\gamma s^0}{(s+\lambda)^3} Y(s) \end{bmatrix} \begin{bmatrix} m \\ d \\ k \end{bmatrix}$$

which corresponds to the model $\mathcal{Y}(t) = \mathcal{U}'(t)\mathcal{P}$ where

$$\mathcal{Y}(t) = \begin{bmatrix} u_0(t) \end{bmatrix}, \quad \mathcal{U}(t) = \begin{bmatrix} y_2(t) \\ y_1(t) \\ y_0(t) \end{bmatrix}, \quad \mathcal{P} = \begin{bmatrix} m \\ d \\ k \end{bmatrix}$$

and the filtered measurements, denoted by $u_i(t)$ and $y_i(t)$, are obtained by passing $u(t)$ and $y(t)$ through filters with transfer functions

$$H_i(s) = \frac{\gamma s^i}{(s+\lambda)^3}, \quad i = 0, 1, 2.$$

The characteristic polynomial

$$(s+\lambda)^3 = s^3 + 3\lambda s^2 + 3\lambda^2 s + \lambda^3$$

may be expressed as

$$(s+\lambda)^3 = s^3 + \lambda_2 s^2 + \lambda_1 s + \lambda_0, \quad \lambda_i = \binom{3}{3-i} \lambda^{3-i}, \quad i = 0, 1, 2$$

and a convenient state-space implementation with $\gamma = \lambda^3$ would be

$$\begin{bmatrix} \dot{\alpha}_0(t) \\ \dot{\alpha}_1(t) \\ \dot{\alpha}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -\lambda_0 & -\lambda_1 & -\lambda_2 \end{bmatrix} \begin{bmatrix} \alpha_0(t) \\ \alpha_1(t) \\ \alpha_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \lambda_0 \end{bmatrix} \alpha(t), \quad \alpha = u, y.$$

The state variables of these filters are precisely the signals needed to populate $\mathcal{Y}(t)$ and $\mathcal{U}(t)$.

More than one parameterization may be considered. For example, suppose that we start with the given model in the standard transfer function representation

$$Y(s) = \frac{(1/m)}{s^2 + (d/m)s + (k/m)} U(s)$$

and then rewrite it in the form

$$s^2 Y(s) = s^0 U(s)(1/m) - s^1 Y(s)(d/m) - s^0 Y(s)(k/m).$$

If we multiply both sides of this plant model by $\gamma/(s+\lambda)^3$ then we obtain

$$\left[\frac{\gamma s^2}{(s+\lambda)^3} Y(s) \right] = \left[\frac{\gamma s^0}{(s+\lambda)^3} U(s) \quad -\frac{\gamma s^1}{(s+\lambda)^3} Y(s) \quad -\frac{\gamma s^0}{(s+\lambda)^3} Y(s) \right] \begin{bmatrix} 1/m \\ d/m \\ k/m \end{bmatrix}$$

which corresponds to the model $\mathcal{Y}(t) = \mathcal{U}'(t)\mathcal{P}$ where

$$\mathcal{Y}(t) = \begin{bmatrix} y_2(t) \end{bmatrix}, \quad \mathcal{U}(t) = \begin{bmatrix} u_0(t) \\ -y_1(t) \\ -y_0(t) \end{bmatrix}, \quad \mathcal{P} = \begin{bmatrix} 1/m \\ d/m \\ k/m \end{bmatrix}.$$

Note that the filtered signals needed for both parameterizations of this system are identical.

The above ideas may be generalized to address nth-order systems modeled by

$$Y(s) = \frac{b_{n-1}s^{n-1} + \cdots + b_1 s + b_0}{s^n + a_{n-1}s^{n-1} + \cdots + a_1 s + a_0} U(s). \tag{1}$$

In this case, the linearly parameterized model is based on

$$\mathcal{Y}(t) = \begin{bmatrix} y_n(t) \end{bmatrix} \tag{2}$$

$$\mathcal{U}(t) = \begin{bmatrix} u_{n-1}(t) & \cdots & u_0(t) & -y_{n-1}(t) & \cdots & -y_0(t) \end{bmatrix}' \tag{3}$$

$$\mathcal{P} = \begin{bmatrix} b_{n-1} & \cdots & b_0 & a_{n-1} & \cdots & a_0 \end{bmatrix}' \tag{4}$$

and the coefficient matrices of the filters for $u(t)$ and $y(t)$ are

$$\mathcal{A} = \begin{bmatrix} 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 1 \\ -\lambda_0 & -\lambda_1 & \cdots & -\lambda_{n-1} & -\lambda_n \end{bmatrix}, \quad \mathcal{B} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \lambda_0 \end{bmatrix} \tag{5}$$

where λ is adjusted to the bandwidth of the plant and

$$\lambda_i = \binom{n+1}{n+1-i} \lambda^{n+1-i}, \quad i = 0, \ldots, n. \tag{6}$$

Once we obtain a good estimate $\hat{\mathcal{P}}$ of the unknown $\mathcal{P}$—the topic addressed in the next section—a natural choice for the estimated plant coefficient matrices would be

$$A = \begin{bmatrix} -\hat{a}_{n-1} & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ -\hat{a}_1 & 0 & \cdots & 0 & 1 \\ -\hat{a}_0 & 0 & \cdots & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} \hat{b}_{n-1} \\ \vdots \\ \hat{b}_1 \\ \hat{b}_0 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}. \tag{7}$$

With these estimates of A, B and C, design of a controller can proceed.

1.2 Least-Squares Parameter Identification

Assuming zero initial conditions for the plant and filters, the linear equation $\mathcal{Y}(t) = \mathcal{U}'(t)\mathcal{P}$ is true for all $t \geq 0$. If L samples of the filtered signals are recorded at $t = t_1, \ldots, t_L$, then the result is a collection of L numerical constraint equations on the unknown vector $\mathcal{P}$ expressed as

$$\begin{bmatrix} \mathcal{U}'(t_1) \\ \vdots \\ \mathcal{U}'(t_L) \end{bmatrix} \mathcal{P} = \begin{bmatrix} \mathcal{Y}(t_1) \\ \vdots \\ \mathcal{Y}(t_L) \end{bmatrix}. \tag{8}$$

In a typical experiment, we would choose L to be much larger than the number of elements in $\mathcal{P}$. Due to measurement noise, we cannot expect to find $\mathcal{P}$ satisfying all L constraints. Hence, we wish to compute $\hat{\mathcal{P}}$ so that the sum of squared errors E is minimized:

$$E = \sum_{k=1}^{L} \left(\mathcal{U}'(t_k)\hat{\mathcal{P}} - \mathcal{Y}(t_k) \right)^2. \tag{9}$$

This method of approximating unknown vector $\mathcal{P}$ by $\hat{\mathcal{P}}$ is based on the principle of least squares published by Gauss in 1809 (Theory of Motion of the Heavenly Bodies). According to this principle, the value of $\hat{\mathcal{P}}$ that we seek can be found by solving the following system of equations:

$$\begin{bmatrix} \mathcal{U}'(t_1) \\ \vdots \\ \mathcal{U}'(t_L) \end{bmatrix}' \begin{bmatrix} \mathcal{U}'(t_1) \\ \vdots \\ \mathcal{U}'(t_L) \end{bmatrix} \hat{\mathcal{P}} = \begin{bmatrix} \mathcal{U}'(t_1) \\ \vdots \\ \mathcal{U}'(t_L) \end{bmatrix}' \begin{bmatrix} \mathcal{Y}(t_1) \\ \vdots \\ \mathcal{Y}(t_L) \end{bmatrix}. \tag{10}$$

Note that (10) is a linear system of equations, where the number of constraints matches the number of variables. If the filtered data has been generated using an appropriate plant excitation, then (10) will be a consistent system with a unique solution that we can compute numerically.

1.3 The Two-Mass System

Consider the two-mass system in Figure 2, where $u(t)$ is input force and $y(t)$ is output position. Mass M_1 is characterized by position $x_1(t)$ and velocity $x_3(t)$, whereas mass M_2 is characterized by position $x_2(t)$ and velocity $x_4(t)$. The state-space model of this plant is

$$\dot{x}(t) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -K_c/M_1 & K_c/M_1 & -D_c/M_1 & D_c/M_1 \\ K_c/M_2 & -K_c/M_2 & D_c/M_2 & -D_c/M_2 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 0 \\ 1/M_1 \\ 0 \end{bmatrix} u(t) \tag{11}$$

$$y(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} x(t) \tag{12}$$

and the corresponding transfer function model is

$$Y(s) = \frac{b_1 s + b_0}{s^4 + a_3 s^3 + a_2 s^2} U(s) \tag{13}$$

where

$$b_1 = D_c \frac{1}{M_1 M_2}, \quad b_0 = K_c \frac{1}{M_1 M_2}, \quad a_3 = D_c \frac{M_1 + M_2}{M_1 M_2}, \quad a_2 = K_c \frac{M_1 + M_2}{M_1 M_2}. \tag{14}$$

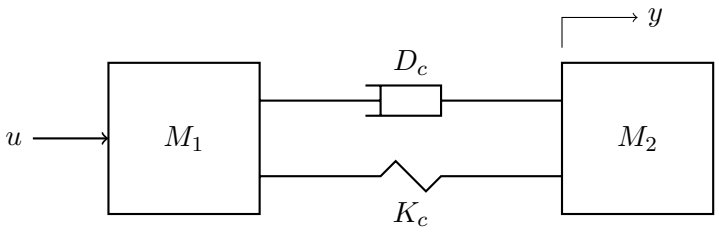

Figure 2: Two-mass system with unknown parameters.

Our objective is to estimate the four unknown transfer function coefficients so that we will be able to design a controller for this plant using the design model

$$\dot{\xi}(t) = \begin{bmatrix} -\hat{a}_3 & 1 & 0 & 0 \\ -\hat{a}_2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \xi(t) + \begin{bmatrix} 0 \\ 0 \\ \hat{b}_1 \\ \hat{b}_0 \end{bmatrix} u(t) \tag{15}$$

$$y(t) = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \xi(t). \tag{16}$$

For this purpose, we need to apply the linearly parameterized model

$$\mathcal{Y}(t) = \begin{bmatrix} y_4(t) \end{bmatrix}, \quad \mathcal{U}(t) = \begin{bmatrix} u_1(t) \\ u_0(t) \\ -y_3(t) \\ -y_2(t) \end{bmatrix}, \quad \mathcal{P} = \begin{bmatrix} b_1 \\ b_0 \\ a_3 \\ a_2 \end{bmatrix} \tag{17}$$

and ultimately solve a system of four equations to obtain the four unknown coefficients:

$$\underbrace{\begin{bmatrix} \mathcal{U}'(t_1) \\ \vdots \\ \mathcal{U}'(t_L) \end{bmatrix}' \begin{bmatrix} \mathcal{U}'(t_1) \\ \vdots \\ \mathcal{U}'(t_L) \end{bmatrix}}_{4\times 4} \begin{bmatrix} \hat{b}_1 \\ \hat{b}_0 \\ \hat{a}_3 \\ \hat{a}_2 \end{bmatrix} = \underbrace{\begin{bmatrix} \mathcal{U}'(t_1) \\ \vdots \\ \mathcal{U}'(t_L) \end{bmatrix}' \begin{bmatrix} \mathcal{Y}(t_1) \\ \vdots \\ \mathcal{Y}(t_L) \end{bmatrix}}_{4\times 1}. \tag{18}$$

Using generic notation, this system of equations may be expressed in the form

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}. \tag{19}$$

Application of Gaussian elimination results in a simpler but equivalent form

$$\begin{bmatrix} a'_{11} & a'_{12} & a'_{13} & a'_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & 0 & a'_{33} & a'_{34} \\ 0 & 0 & 0 & a'_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \end{bmatrix}. \tag{20}$$

Use of back substitution leads to the explicit solution

$$x_4 = \frac{b'_4}{a'_{44}}, \quad x_i = \frac{1}{a'_{ii}} \left(b'_i - \sum_{j=i+1}^{N} a'_{ij} x_j \right), \quad i = 3, 2, 1. \tag{21}$$

We will use the approach of (19)–(21) to solve (18) for the unknown plant parameters.

2 Lab Assignment

2.1 Tasks

Both tasks use the timing parameters in Table 1 and linker command file `user_RAM.cmd`.

Table 1: Specified Clock and Timer Frequencies

CPU clock frequency	$f_{\text{clk,cpu}}$	200 MHz
Timer interrupt frequency	f_{tmr}	100 Hz

2.1.1 Develop an Embedded Solver

Being able to solve a system of linear equations on a computer is an essential tool of scientific computing. Making this tool available on a microcontroller enables new embedded applications including the ability to apply least-squares methods for plant parameter identification. In this task, you will develop C code to solve a system of linear equations on the F28379D microcontroller, using the supplied Matlab program `reference_code.m` as a guide.

1. Run `reference_code.m` to see how Gaussian elimination followed by back substitution provides a systematic method to solve a system of linear equations expressed in the form

   $$Ax = b \quad \Rightarrow \quad a = \begin{bmatrix} A & b \end{bmatrix} = \text{augmented matrix used as program input.}$$

   Note that `reference_code.m` includes five cases to study, each defined by an $N \times (N+1)$ augmented matrix a. The "true" solution is evaluated by extracting the final column of the reduced row echelon form of a (using Matlab's built-in function `rref`). Custom function `solve_system` provides a method that relies only on elementary scalar operations that can easily be transcribed into C. The two solutions are compared for validation purposes using custom function `compute_error`.

2. Develop a C program for the microcontroller that mimics `reference_code.m`, by transcribing its functions `solve_system` and `compute_error` into C. It is not possible for a function in C to `return` arrays, so `solve_system` will have both input and output arguments. Data for the five test cases are provided in header file `examples.h`. Use `asm(" ESTOP0");` in your C code in place of `error('A is rank deficient')` in the supplied Matlab code; this assembly language instruction provides a convenient way to pause your debug session when appropriate. Use `fabs` in C in place of `abs` in Matlab. Your `main.c` should have the following structure.

```
#include "F2837xD_device.h"
#include "math.h"

#define choice ?
#include "examples.h"

void solve_system (float32 a[N][N+1], float32 soln[N]);
float32 compute_error (float32 x[N], float32 soln[N]);

float32 soln[N], numerr;

void main(void)
{
    EALLOW;
    WdRegs.WDCR.all = ?;

    // configure system clock here

    solve_system(a,soln);
    numerr = compute_error(x,soln);

    WdRegs.WDCR.all = ?;
    EDIS;

    while(1)
    {
        EALLOW;
        WdRegs.WDKEY.all = ?;
        WdRegs.WDKEY.all = ?;
        EDIS;
    }
}
```

3. Demonstrate your microcontroller code for all five test cases by displaying `soln` and `numerr` in the expressions window; note that "true" solutions are provided in header file `examples.h`. The errors will be larger than what Matlab reported, since `reference_code.m` relies on the 64-bit floating point default data type; although it's not requested here, you can replace all occurrences of `float32` by `float64` in your C code to improve accuracy.

2.1.2 Perform Parameter Identification

In research and development, there is a general need to subdivide big problems into several smaller problems. For example, it is often critical to test new code modules in isolation so that they may be verified before introducing them into the larger project. In control applications, a systematic code module testing approach might involve the following progression:

- develop and verify a model of plant dynamic behavior over a range of parameter values and excitation conditions in Matlab;

- using the verified plant model, develop and verify algorithms for plant identification and plant control in Matlab;
- using the verified plant model and Matlab algorithms, develop and verify corresponding C codes running on a microcontroller;
- using the verified C codes, develop and verify additional software and hardware to interface the microcontroller with the real-world physical plant.

In the previous task, you focused on the third item in the above progression by transcribing `solve_system` from Matlab to C. In the present task, you will focus again on getting an algorithm to work on a microcontroller (but in this case without starting from Matlab). In particular, you will develop microcontroller code to perform least-squares parameter identification on a two-mass plant as modeled in §1.3. To avoid the need for interfacing with a real-world two-mass plant, its dynamic behavior will instead be emulated within the microcontroller. The plant emulation will be achieved by incorporating a supplied static library into your project; in this way, your microcontroller code will interact with the virtual plant in a real-time sense and—to align with real-world applications—you will not know the true parameters of the virtual plant under consideration. Your `main.c` should have the following structure; note specific assignments to `N`, `L`, `T` and `lambda`.

```
#include "F2837xD_device.h"
#include "lab7_module.h"
#include "math.h"

#define pi 3.14159265
#define N 4
#define L 1000
#define T 0.01

interrupt void timerISR(void);

void filter_read_update (void);
void build_system (void);
void solve_system (float32 a[N][N+1], float32 soln[N]);

float32 lambda = 2, t, u, y, U[L], Y[L], UU[L][N], YY[L], a[N][N+1], soln[N];
Uint16 n = 0;

void main(void)
{
    EALLOW;
    WdRegs.WDCR.all = ?;

    // configure clock
    // configure timer
    // configure interrupts

    WdRegs.WDCR.all = ?;
    EDIS;
```

```
    while(1)
    {
        EALLOW;
        WdRegs.WDKEY.all = ?;
        WdRegs.WDKEY.all = ?;
        EDIS;
    }
}

interrupt void timerISR(void)
{
    if (n < L)
    {
        t = n*T;

        filter_read_update();

        if ((t >= 0) && (t < 2*pi)) u = 3*sin(t)-sin(3*t);
        else u = 0;

        plant_input_output();

        U[n] = u;
        Y[n] = y;

        n++;
    }
    else
    {
        EALLOW;
        WdRegs.WDCR.all = ?;
        EDIS;
        CpuTimer0Regs.TCR.bit.TSS = ?;

        build_system();
        solve_system(a,soln);
    }

    PieCtrlRegs.PIEACK.all = ?;
}
```

1. The first phase of operation is a real-time phase, with one update occurring every 10 ms in the timer ISR. During each update, four actions are taken.

   (a) The first action is to execute function `filter_read_update` which you will prepare. As its name suggests, this function is intended to perform two tasks.

i. It reads the filtered versions of $u(t)$ and $y(t)$, namely signals $u_0(t)$, $u_1(t)$, $y_2(t)$, $y_3(t)$ and $y_4(t)$, and populates the arrays `UU[L][4]` and `YY[L]` according to

$$\underbrace{\begin{bmatrix} u_1(t_1) & u_0(t_1) & -y_3(t_1) & -y_2(t_1) \\ \vdots & \vdots & \vdots & \vdots \\ u_1(t_L) & u_0(t_L) & -y_3(t_L) & -y_2(t_L) \end{bmatrix}}_{\texttt{UU[L][4]}} \begin{bmatrix} \hat{b}_1 \\ \hat{b}_0 \\ \hat{a}_3 \\ \hat{a}_2 \end{bmatrix} = \underbrace{\begin{bmatrix} y_4(t_1) \\ \vdots \\ y_4(t_L) \end{bmatrix}}_{\texttt{YY[L]}} \tag{22}$$

This system of equations is a precursor to (18), i.e. prior to squaring down.

ii. It updates the state variables of the two 5th-order filters, defined according to (5)–(6), one for $u(t)$ and one for $y(t)$, using forward Euler discretization with a 10 ms time increment, beginning from initial conditions set equal to zero.

(b) The second action is to provide an excitation to the virtual plant. The response of a plant to its excitation depends on the unknown plant parameters, so the objective here is to generate input-output data that is informative for parameter identification purposes. This can be achieved by applying excitation that includes a sufficient number of frequency components within the bandwidth of the plant. In this case, we apply

$$u(t) = \begin{cases} 3\sin(t) - \sin(3t) & , \ 0 \le t \le 2\pi \\ 0 & , \ \text{otherwise} \end{cases} \tag{23}$$

(c) The third action is to execute function `plant_input_output` which is made available in executable form using the supplied library file `lab7_10ms_euler.lib` and its associated header file `lab7_module.h`. As its name suggests, this function updates the solution of plant model (11)–(12) using forward Euler discretization with a 10 ms time increment; it assumes zero initial conditions, it takes the present value of $u(t)$ as its input, and it returns the present value of $y(t)$ as its output. The response of the virtual plant obtained in this fashion depends on the unknown parameters and the applied excitation.

(d) The fourth action is to log the input $u(t)$ and output $y(t)$ of the virtual plant throughout the interval of real-time operation. These data allow us to observe how the virtual plant is responding to the excitation, but more importantly they also provide a basis to check the quality of fit once the parameter identification effort has been completed.

2. The second phase of operation is an off-line post process, organized into three steps.

(a) The first step is to disable the watchdog to avoid unwanted resets during number crunching, and to stop the timer to avoid unwanted interrupts during number crunching.

(b) The second step is to execute function `build_system` which you will prepare. The purpose of this function is to multiply (22) on both sides by the transpose of the left side's coefficient matrix, squaring down to obtain four equations and four variables. In this step, you will need to multiply matrices and vectors. To assist with these computations, the functions `mm_prod` and `mv_prod` shown below could be used for matrix-matrix and matrix-vector products; note that it is not possible for a function in C to return a non-scalar result, so here both inputs and outputs appear as function arguments. Keep in mind that you want your final system of equations to be represented in the form of an augmented matrix, so this step ends with production of array `a[N][N+1]`.

```
#define N ?
#define M ?
#define P ?

void mm_prod (float32 A[N][M], float32 B[M][P], float32 C[N][P])
{
    Uint16 i, j, k;

    for (i = 0; i < N; i++)
    {
        for (j = 0; j < P; j++)
        {
            C[i][j] = 0;
            for (k = 0; k < M; k++)
            {
                C[i][j] += A[i][k]*B[k][j];
            }
        }
    }
}

void mv_prod (float32 A[N][M], float32 x[M], float32 y[N])
{
    Uint16 i, j;

    for (i = 0; i < N; i++)
    {
        y[i] = 0;
        for (j = 0; j < M; j++)
        {
            y[i] += A[i][j]*x[j];
        }
    }
}
```

(c) The third step is to execute function `solve_system`, which yields the least-squares estimates $\hat{b}_0$, $\hat{b}_1$, $\hat{a}_2$ and $\hat{a}_3$ of the unknown plant parameters. This function has already been verified in the previous task, so no further development is required. Demonstrate your parameter identification method by running your code with array `soln` displayed in the expressions window. Since the real-time phase of data collection involves 1000 samples at 100 Hz, you will need to let your program run for more than 10 s before you will see any results. You can determine if your program output is reasonable; just build a simulation model of your identified plant in Matlab using (15)–(16), simulate it with the known excitation, and compare the response of your simulated/modeled system to your experimental/measured system. The quality of fit may be quantified using

$$\text{quality of fit } (\%) = 100 \left(1 - \frac{\| y_{\text{measured}} - y_{\text{modeled}} \|}{\| y_{\text{measured}} - \text{mean}\{ y_{\text{measured}} \} \|} \right). \tag{24}$$

2.2 Deliverables

To be eligible for full credit, do the following:

1. Upload narrated videos showing completion of §2.1.1 and §2.1.2.
2. Upload your `main.c` file for §2.1.2.