# System Identification Homework - Final Report

**Francesco Biondi** [153497] **Answer to question #1**

In order to estimate the steady-state gain of *G(Z)*, i need to generate an input to feed the system. The most suitable input to do this is a step input of amplitude $M$=350, since the input *u(t)* is saturated between *[-M, M]*. Thus, i will have a constant input of amplitude $M$, $\forall t > 0$.

I can obtain this through this command:

```
u = M*ones(N, 1);
```

Then i need to store the response of the system, represented by $y$ and $w$, generated by the following command:

```
[y_u, w_u] = generate_exam(yourID, u)
```

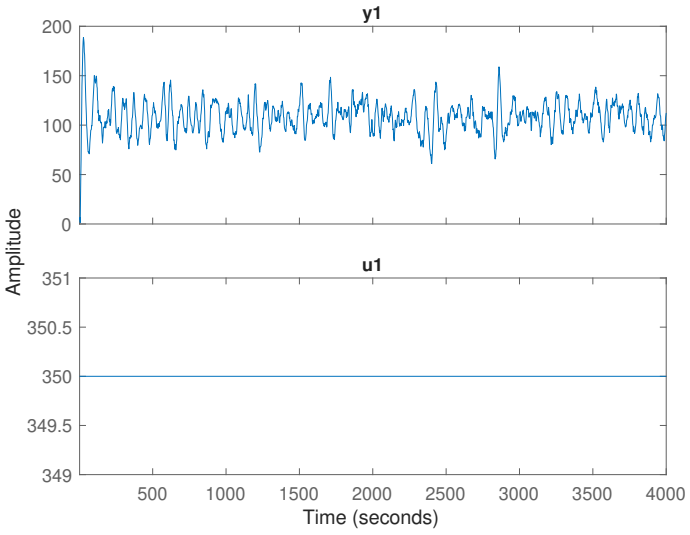
Once $u$, $y\_u$ and $w\_u$ are known, the iddata object encapsulating input-output data is created using the following command:

```
Z1 = iddata(w_u, u, Ts);
```

Where $T_s$ is the sampling time which in my case is egual to 2 seconds.
The three datasets generated [Z1, Z2, Z4] have been saved inside the *SYSID_DATA.mat* file, which is called at the beginning of the main script.

Since the Steady State gain is defined as: $G(Z)_{\text{ssg}} = \lim_{t\to\infty} \frac{w(t)}{M}$ , where *w(t)* is the steady-state output (therefore after an initial possible transient part). The steady-state gain can be approximated by computing the mean of the steady-state output, normalized by the input amplitude $M$, after excluding the transient . In order to see if the system shows a transient behavior, i can look at the plot of $Z$:

**Figure 1: Input-Output Data**




From the response depicted in the graph, i can notice that the system does not exhibit significant transient behavior, apart from the first few samples. Thus, i can exclude the first 100 samples for the computation of the steady-state gain, which in Matlab i can obtain as:

```
SSG = mean(Z1.y(100:end))/M;
```

Doing so, i have that $\boxed{\text{SSG} = 0.3095}$

# System Identification Homework - Final Report

**Francesco Biondi** [153497] **Answer to question #2**

In order to indentify the best Arx model, i chose to use a PRBS (Pseudo Random Binary Sequence) as input for the system. This is obtained with:

```
U=M*idinput(N, 'PRBS', [0 1]); % Generate a PRBs input
```

The dataset Z2, stored in *SYSID_DATA.mat*, is an iddata object containing the output vector y and the input vector w, generated using a PRBS signal. This dataset is used for both estimation and validation Having saved the Z2, is not only convenient for writing/reading the code, but it also ensure consistent results by mitigating the randomness of the input.

The first step is to remove the mean value to eliminate potential biases, then splitting it into two parts, one used for estimation, and the other for validation. In Matlab i can proceed as follows:

```
Z2 = detrend(Z2); % Remove mean from data
Z2e = Z2(1: 1000); % Data set for estimation
Z2v = Z2(1001:2000); % Data set for validation
```

I can set the maximum order of each polynomials of the Arx model that i want to identify, using the function *arxstruc*, as follows:

```
V = arxstruc(Z2e, Z2v, struc(1:5, 1:5, 1:5)); % Set the max order for polynomials to 5
```

Having done this, the function *selstruc* returns the optimal number of parameter for the model, based on the orders selected, and the criteria chosen.

```
nnopt_mdl = selstruc(V, 'mdl');
```

The best ARX model according to the MDL (Minimum Description Lenght) Criterion is $n_a$= 5, $n_b$= 5, $n_k$= 3.

The residual correlation analysis (Figure 2) and the comparison between the true output and the 10-step-ahead prediction (Figure 3), are reported below:

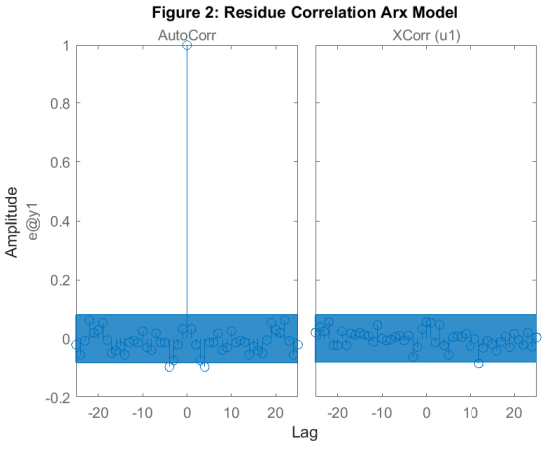


**Figure 2: Residue Correlation Arx Model**

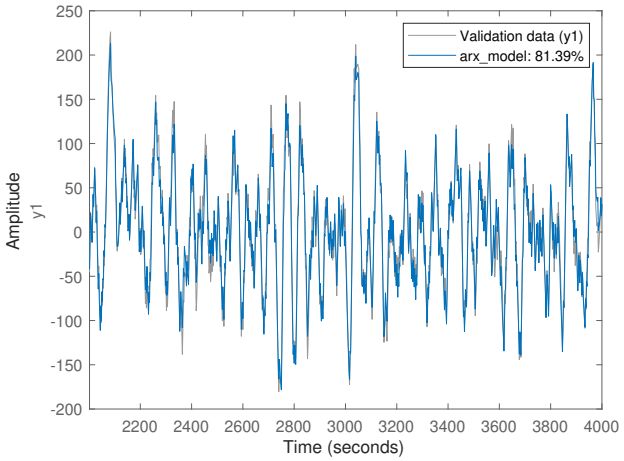


**Figure 3: 10 Step-Ahed prediction comparison**

The identified Arx shows a good FIT of 81.39%, in addition to having all the zeros and poles inside the unitary circle, assuring the asymptotically stability. It also show a good residual correlation, with almost all samples inside the bands, both for the correlation and for the auto-correlation. This means that the residuals are uncorrelated and behave as white noise.

# System Identification Homework - Final Report

**Francesco Biondi** [153497] **Answer to question #3**

Using the same input and output data contained in Z2, i have tested several different model in order to identify the most suitable for G(Z) and H(Z). The most relevant are the following:

- **ARMAX**

  $n_a$= 2, $n_b$= 4, $n_c$= 2 $n_k$= 1

  FIT= 86.31%

- **OE**

  $n_b$= 5, $n_f$= 5, $n_k$= 1

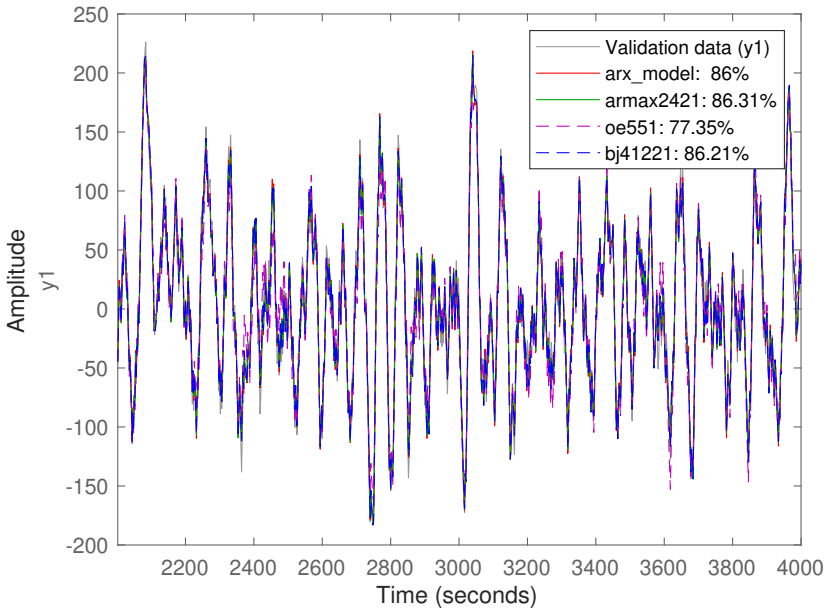
  FIT= 77.35%

- **BJ**

  $n_b$= 4, $n_c$= 1, $n_d$= 2 $n_f$= 2 $n_k$= 1

  FIT= 86.21%

**Figure 4: 5 Step-Ahed prediction comparison**




Looking at the comparison between the model tested, i can say that **oe** is the model which perform the worse, while the **armax** and the **bj** are quite similar. Also, from a residual analysis point of view, the **oe** present a very poor residue correlation, while the other two are comparable.
Neither the models has the poles and zeros inside the unitary circle, so both are not asymptotically stable. For these reasons the best model i can identify is the **ARMAX** model. In the figures below (Figure 5 and 6) are shown the residual analysis and the 5-step-ahead prediction comparison with the true output.

**Figure 5: Residue Correlation (ARMAX)**

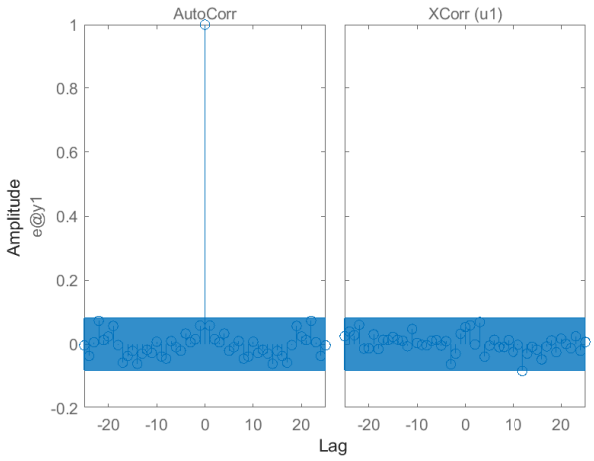
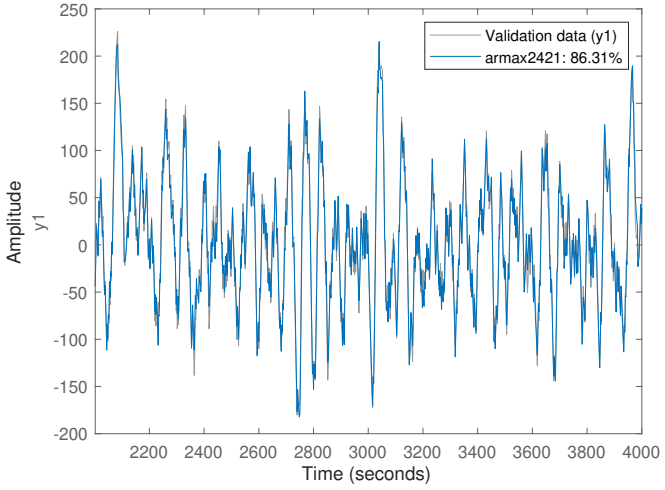



**Figure 6: 5 Step-Ahed prediction comparison (ARMAX)**




To build the two transfer function *G(Z)* and *H(Z)*, i can proceed on Matlab using the *'tf'* function, as follows:

```
G = tf(armax2421.B, armax2421.A, Ts)
H = tf(armax2421.C, armax2421.A, Ts)
```

Which returns:

$$\boxed{G(Z) = \frac{-0.00009451z^3 + 0.00001109z^2 + 0.05833z - 0.04634}{z^2 - 1.871z + 0.9087}} \qquad \boxed{H(Z) = \frac{z^2 - 0.7447z + 0.04351}{z^2 - 1.871z + 0.9087}}$$

Lastly, i can compute the peak grain og G(Z), using the Matlab function *'PeakGain'*, and then converting it into dB:

```
peak_gain_dB=20*log10(getPeakGain(armax2421))
```

Giving as results a value of $\boxed{PeakGaindB = -1.1340}$

# System Identification Homework - Final Report

**Francesco Biondi** [153497] **Answer to question #4**

To identify the most suitable model for $P(z)$, a new dataset $Z_4$ was created. This dataset encapsulates the input $w$ and the output $y$, which represent the closed-loop behavior of the system. The dataset was constructed using the following command:

```
Z4 = iddata(y, w, Ts);
```

The disturbance *n(t)* is as a zero-mean, independent Gaussian white noise that directly overlaps the output signal *y(t)*.

Thus, i can assume that the **Output Error** is the most suitable model to represents $P(z)$, since it does not attempt to model or filter the noise explicitly; instead, it treats the noise as an independent random residual. In contrast, ARMAX and BJ models try to capture the noise dynamics using an additional transfer function $H(q)$ , which introduces unnecessary complexity. Since $n(t)$ is white and additive, the **OE** model aligns very well with the system's structure and the characteristics of the disturbance.

The simulated response comparison (Figure 7) and the residual correlation (Figure 8) of the most relevant **OE** model identified are reported below:

**Figure 7: Simulated response comparison (OE)**

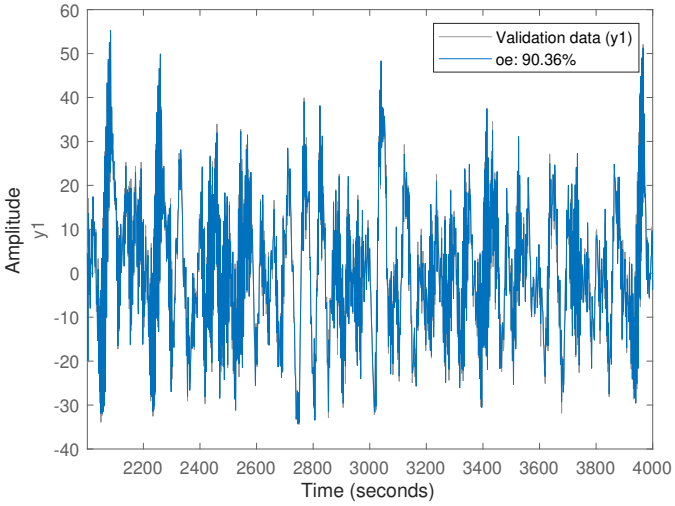
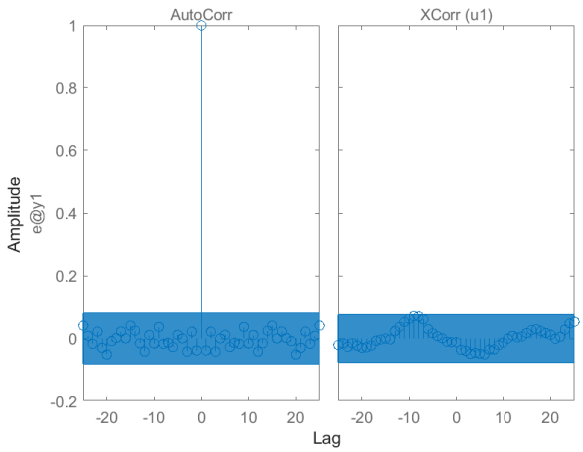



**Figure 8: Residue Correlation (OE)**




The parameters of this **OE** model are: $n_b$= 1, $n_f$= 1, $n_k$= 1, which shows both a very good residue correlation and a FIT of 90.36%.

Its transfer function was computed in Matlab:

```
P = tf(oe.B, oe.F, Ts);
```

The resulting transfer function is:

$$P(z) = \frac{0.3474}{z + 0.9296}$$

Once $P(z)$ was known, i was able to derive $R(z)$ using the closed-loop transfer function relation, for which:

$$P(z) = \frac{R(z)}{1 + k \cdot R(z)} \;\Rightarrow\; R(z) = \frac{P(z)}{1 - k \cdot P(z)} = \frac{\frac{0.3474}{z+0.9296}}{1 - k \cdot \frac{0.3474}{z+0.9296}} = \frac{0.3474}{z + 0.9296 - k \cdot 0.3474}$$

Substituting the value of $k$:

$$\boxed{R(z) = \frac{0.3474}{z + 0.478}}$$

In Matlab i can obtain the same results through the following commands:

```
num_R = oe.B; % Define the numerator of R(z)
den_R = conv(oe.F, [1]) - k * conv(oe.B, [1]); % Compute the denominator of R(z)
R = tf(num_R, den_R, Ts); % Transfer function of R(z)
```

Finally, the pole of $R(z)$ was estimated using the Matlab function *pole*:

```
R_pole = pole(R);
```

Which returns as results $\boxed{-0.4780}$ . This confirms that $R(z)$ is stable, as the pole lies within the unit circle.