Progress Report on Time Delay Estimation

(Report #2)

Linjian Xiang

Email: linjian@ualberta.ca

Phone #: (780)8625922

Date 2018-1-21

# 

# **Objectives**

# The objective of this experiment is to find the best method for calculating the time delay between series of datasets. There were six different type of datasets tested, each of them had been added 5, 15 and 50 sample delays during experiments. All of them were tested by five delay-finding method. After collecting all the results, and calculating the Mean Square Error, we can compare and identify the most effective method for delay estimation.

* **Methodologies**

There are commonly five different types of methods for finding the delay:

*Correlation and coherence based approaches*

1. Cross-correlation method (CORR)
2. Coherence method (CSD)

*Discrete-time explicit model based*

1. Auto Regressive model with eXtra input (ARX)
2. Output Error model (OE)
3. Prefiltered ARX model based (denoted by MET1 in this report)

**Method 1:**

The cross-correlation method is based on finding the maximum cross-correlation of the input dataset and the output dataset with respect to *i*, where *i* is the delay shift present in the output dataset:

**Method 2:**

For the coherence method, both of input and output are transformed into frequency domain values. After that, correlation in frequency domain (f\_z) can be calculated and the single-side spectrum of datasets can be found. Then the estimated frequency is determined by 1) The frequency with largest amplitude on single-side spectrum and 2) The value where the magnitude squared coherence has the largest or second largest value. At last the delay is equal to the angle of the frequency where the *f*z has the largest value divided by 2π*f*. (The angle can be found by calculating the cross power spectral density of datasets):

The above two methods simply find the best delay estimation by exploiting the correlations among signals in both time and frequency domains. The following three methods are model-based, in which the delay is modeled as an explicit parameter in the discrete-time model, and several system identification methods are used to find the parameters, including the delay samples.

**Method 3:**

For the ARX model based method, its model can be represented by:

Where A(q), B(q) are polynomials of the delay operator *q*-1in the regression model. Then let

=]

=

Therefore, the notation ,

And the ARX model can be expressed by:

The delay can be found by using the Least-Squares Estimation to calculate the local minimum of the loss function in terms of .

**Method 4:**

The OE model is following:

It is supposed that the relation between input *u* and the undisturbed output *y* can be written as a linear difference equation, and the disturbances consist of white measurement noise.

The method of finding delay is to calculate the variance of the noise component *e*(*t*) for *n*k= 1,2, …*n*, and the delay is equal to the *n*k corresponding to the lowest error variance. Therefore, the computation time is about nk-times of others methods.

**Method 5:**

The OE gives a better result than ARX model based method, but takes much more computation time. The MET method is proposed to imitate OE method but with much lower computational demands. Assume we have a true system the OE structure can be also written as . At the same time the system can also be estimated by ARMAX structure which is , a Box-Jenkins model type, and the coefficients A1 and C1 will be approximations of F(q). If the input u(t) and output y(t) are filtered through 1/C(q), we will get the filtered model . The delay can be find by applying ARX method on the filtered model.

Unfortunately, the standard way to estimate a model of this structure also requires a numerical search as with the OE model which we tried to avoid. Another way is to first estimate a state space model and then convert it to an ARMAX model. This conversion will be possible if the order of the state space model and the orders of the polynomials A(q), B(q) and C(q) are high enough to describe the true system. The pseudo codes the above methods are included in the appendix.

* **Simulation Models and Datasets**

To test the capability of above methods, datasets were designed from easiest random noise to more complexed cases. For this report, six different types of datasets were used to test delay methods:

1. A random data set input passed through a delay and a noise adder.
2. A sinewave input gets decayed overtime and passed through a delay and a noise adder
3. Similar with second case, but the input was changed to be addition of multiple sinewaves.
4. Fourteen sets of TE output data collected from the Simulink, and add delay on them.
5. A step input passed through an open loop system with delay.
6. Similar with the fifth case, but the closed loop system.

(please see Appendix for simulation models constructed in Simulink)

For open and close loop system (dataset 5 and 6), a step input passed through slow, medium and fast PI controller then delay and a system to generate dataset.

* **Test results and Discussion**

Each of the testing cases has three different time-delay added: 5, 15 and 50 samples. Because the noise profile added are different (with different seed for noise generator) every time the program is run, for consistency, each simulation case is run by multiple times (e.g. 10 times). The results are following.

Table1: Delay calculating result with 10-times iterations

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Simulated delay(samples) | **corr** | **csd** | **oes** | **arx** | **met1** |
| **Case1.** | **delay rand noise** | |  |  |  |
| 5 | 5 | 0.90 | 5 | 2.1 | 5 |
| 15 | 20 | 1.97 | 20 | 13.7 | 20 |
| 50 | 70 | 4.99 | 70 | 60.5 | 36.4 |
| MSE | 6.87 | 15.67 | 6.87 | 3.65 | 4.83 |
| **Case 2.** | **delay multi sin** | |  |  |  |
| 5 | 5 | 16.73 | 3.5 | 10.6 | 1 |
| 15 | 20 | 39.72 | 17.3 | 25.3 | 2 |
| 50 | 70 | 98.54 | 66.3 | 77.3 | 21.5 |
| MSE | 6.87 | 18.57 | 5.50 | 9.90 | 10.52 |
| **Case 3** | **delay sin damp** | |  |  |  |
| 5 | 5 | 4.83 | 34.8 | 10.9 | 0 |
| 15 | 20 | 20.56 | 77.3 | 28.4 | 0 |
| 50 | 70 | 37.27 | 102.2 | 79.8 | 0 |
| MSE | 6.87 | 4.63 | 28.85 | 11.06 | 17.48 |
| **Case 5** | **Open-loop with step input, slow PI** | | | |  |
| 5 | 19.5 | - | 10.5 | 53.2 | 16.9 |
| 15 | 52.9 | - | 28.8 | 97 | 34.8 |
| 50 | 90.8 | - | 47 | 153.1 | 53.2 |
| MSE | 19.18 | - | 5.05 | 46.75 | 7.77 |
|  | **Open-loop with step input, medium PI** | | | |  |
| 5 | 29.5 | - | 2.5 | 51.5 | 13 |
| 15 | 73 | - | 4 | 95 | 29.2 |
| 50 | 112.4 | - | 19.3 | 134.7 | 47.9 |
| MSE | 29.54 | - | 10.90 | 41.81 | 5.48 |
|  | **Open-loop with step input, fast PI** | | | |  |
| 5 | 17.1 | - | 20.7 | 47.9 | 16.8 |
| 15 | 45.1 | - | 27.5 | 91.4 | 34.8 |
| 50 | 83.1 | - | 32.9 | 144 | 54.2 |
| MSE | 15.44 | - | 8.79 | 42.83 | 7.81 |
| **Case 6** | **Closed-loop with step input, slow PI** | | | |  |
| 5 | 16.1 | - | 7 | 46.4 | 16.2 |
| 15 | 41.1 | - | 21.2 | 99.3 | 32.7 |
| 50 | 81.3 | - | 33.2 | 142.7 | 51.7 |
| MSE | 14.07 | - | 6.01 | 43.98 | 7.00 |
|  | **Closed-loop with step input, medium PI** | | | |  |
| 5 | 25.1 | - | 3.1 | 31.8 | 12.8 |
| 15 | 66.5 | - | 6.9 | 77.6 | 27.4 |
| 50 | 106.8 | - | 17.6 | 116.5 | 45.1 |
| MSE | 26.42 | - | 11.15 | 31.72 | 5.14 |
|  | **Closed-loop with step input, fast PI** | | |  |  |
| 5 | 11.3 | - | 7.5 | 33.2 | 15.6 |
| 15 | 36.1 | - | 16.1 | 82.7 | 32.7 |
| 50 | 75.4 | - | 21.1 | 126.1 | 50.3 |
| MSE | 11.20 | - | 9.67 | 35.23 | 6.87 |
|  |  |  |  |  |  |
| average MSE | 15.17 | - | 10.31 | 29.66 | 8.10 |

From the table, we see that the CSD method did not work on open-loo or closed-loop system we designed. One reason is that for those systems, input has a deterministic trend, e.g. steps (step from 0 to end).

**Case 4:**

For the TE model, there are 14 datasets are chosen from TE MATLAB simulation (they are: xmeas 1 2 3 4 7 8 9 10 11 12 14 15 17 23 40). Delays are directly added to the datasets, the TE data are generated in continues time therefore the results include continues delay added and discrete delay added.

The reason to test TE model outputs is because all the data are dynamically dependent. They may contain colored-noise, and compared with some others cases they are more general.

Table 2: fourteen TE datasets with 5 samples delay

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| data set | TE discrete delay simulated: 5 (samples) | | | | | | | | | | | | | | **average error** | **MSE** |
| **corr** | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 0.0 | 0.0 |
| **csd** | 11.7 | 107.1 | 59.7 | 15.6 | 1.1 | 0.1 | 45.8 | 3.6 | 81.0 | 45.7 | 43.3 | 408.7 | 0.3 | 150.3 | 456.4 | 32.6 |
| **oes** | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 0.0 | 0.0 |
| **arx** | 5.0 | 1.0 | 1.0 | 1.0 | 1.0 | 3.0 | 4.0 | 2.0 | 3.0 | 3.0 | 1.0 | 1.0 | 2.0 | 3.0 | 11.4 | 0.8 |
| **met** | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 0.0 | 0.0 |
| data set | TE continues delay simulated: 5(samples) | | | | | | | | | | | | | | | |
| **corr** | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 0.0 |  |
| **csd** | 450.1 | 1800.3 | 1800.3 | 450.1 | 81.8 | 1.2 | 46.2 | 56.4 | 900.1 | 720.1 | 720.1 | 430.1 | 8.7 | 200.0 | 2982.4 | 213.0 |
| **oes** | 6.0 | 5.0 | 5.0 | 5.0 | 6.0 | 5.0 | 6.0 | 5.0 | 6.0 | 5.0 | 5.0 | 5.0 | 5.0 | 6.0 | 2.2 | 0.2 |
| **arx** | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 4.0 | 3.0 | 1.0 | 2.0 | 3.0 | 4.0 | 5.0 | 12.1 | 0.9 |
| **met** | 5.0 | 3.0 | 3.0 | 5.0 | 5.0 | 5.0 | 5.0 | 4.0 | 5.0 | 4.0 | 1.0 | 2.0 | 3.0 | 1.0 | 7.4 | 0.5 |

Table 3: fourteen TE datasets with 15 samples delay

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| data set | TE discrete delay simulated: 15(samples) | | | | | | | | | | | | | | average error | MSE |
| **corr** | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 5.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 10.0 | 0.7 |
| **csd** | 437.6 | 1746.0 | 1752.5 | 436.8 | 42.5 | 0.1 | 1.3 | 8.6 | 827.4 | 601.9 | 696.5 | 263.4 | 8.4 | 42.8 | 2811.4 | 200.8 |
| **oes** | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 0.0 | 0.0 |
| **arx** | 11.0 | 11.0 | 11.0 | 11.0 | 11.0 | 12.0 | 13.0 | 11.0 | 15.0 | 14.0 | 11.0 | 11.0 | 12.0 | 12.0 | 12.6 | 0.9 |
| **met** | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 0.0 | 0.0 |
| data set | TE continues delay simulated: 15(samples) | | | | | | | | | | | | | | | |
| **corr** | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 6.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 9.0 | 0.6 |
| **csd** | 169.3 | 1800.3 | 457.2 | 450.1 | 10.1 | 0.7 | 17.4 | 15.3 | 787.1 | 275.8 | 720.1 | 390.0 | 6.5 | 92.8 | 2214.5 | 158.2 |
| **oes** | 16.0 | 15.0 | 15.0 | 15.0 | 16.0 | 15.0 | 16.0 | 15.0 | 16.0 | 15.0 | 15.0 | 15.0 | 15.0 | 16.0 | 2.2 | 0.2 |
| **arx** | 14.0 | 14.0 | 14.0 | 14.0 | 16.0 | 14.0 | 16.0 | 14.0 | 16.0 | 13.0 | 14.0 | 13.0 | 14.0 | 14.0 | 4.5 | 0.3 |
| **met** | 16.0 | 16.0 | 16.0 | 16.0 | 16.0 | 15.0 | 16.0 | 15.0 | 16.0 | 15.0 | 14.0 | 15.0 | 15.0 | 14.0 | 3.0 | 0.2 |

Table 4: fourteen TE datasets with 15 samples delay

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| data set | TE discrete delay 50 | | | | | | | | | | | | | | average error | MSE |
| **corr** | 50.0 | 50.0 | 50.0 | 50.0 | 50.0 | 50.0 | 43.0 | 50.0 | 6.0 | 50.0 | 50.0 | 50.0 | 50.0 | 50.0 | 44.6 | 3.2 |
| **csd** | 61.8 | 1800.3 | 1681.9 | 450.1 | 43.3 | 0.9 | 46.2 | 23.5 | 900.1 | 720.1 | 91.8 | 277.7 | 8.7 | 671.1 | 2739.1 | 195.7 |
| **oes** | 50.0 | 50.0 | 50.0 | 50.0 | 50.0 | 50.0 | 50.0 | 50.0 | 50.0 | 50.0 | 50.0 | 50.0 | 50.0 | 50.0 | 0.0 | 0.0 |
| **arx** | 46.0 | 46.0 | 46.0 | 46.0 | 46.0 | 50.0 | 50.0 | 47.0 | 46.0 | 48.0 | 46.0 | 46.0 | 49.0 | 50.0 | 11.9 | 0.9 |
| **met** | 20.0 | 20.0 | 20.0 | 20.0 | 20.0 | 20.0 | 20.0 | 20.0 | 20.0 | 20.0 | 20.0 | 20.0 | 20.0 | 16.0 | 113.4 | 8.1 |
| data set | TE continues delay 50 | | | | | | | | | | | | | | | |
| **corr** | 50.0 | 50.0 | 50.0 | 50.0 | 51.0 | 50.0 | 43.0 | 50.0 | 7.0 | 50.0 | 50.0 | 50.0 | 50.0 | 50.0 | 43.6 | 3.1 |
| **csd** | 113.4 | 461.9 | 96.9 | 394.6 | 56.9 | 0.1 | 23.0 | 80.3 | 691.6 | 311.7 | 217.8 | 443.0 | 7.1 | 586.5 | 1118.5 | 79.9 |
| **oes** | 51.0 | 50.0 | 50.0 | 50.0 | 51.0 | 50.0 | 51.0 | 50.0 | 51.0 | 50.0 | 50.0 | 50.0 | 50.0 | 51.0 | 2.2 | 0.2 |
| **arx** | 49.0 | 49.0 | 49.0 | 49.0 | 51.0 | 49.0 | 51.0 | 49.0 | 51.0 | 48.0 | 49.0 | 48.0 | 49.0 | 50.0 | 4.4 | 0.3 |
| **met** | 20.0 | 19.0 | 19.0 | 20.0 | 20.0 | 19.0 | 20.0 | 20.0 | 20.0 | 20.0 | 20.0 | 19.0 | 20.0 | 17.0 | 114.2 | 8.2 |

Table 5: averaging the result

|  |  |
| --- | --- |
| **Methods** | **Average MSE(samples)** |
| corr | 3.259839 |
| csd | - |
| oe | 1.296602 |
| arx | 6.351538 |
| met | 3.382785 |

Table 6: Open-loop and Close-loop result

|  |  |  |  |
| --- | --- | --- | --- |
| Data Sets | OE MET MSE | **Methods** | **Average MSE (samples)** |
| open slow | 5.77 | OE | 8.59 |
| open medium | 5.56 | MET | 6.68 |
| open fast | 7.38 | Average OE MET result | 6.06 |
| close slow | 5.20 | Coor | 19.31 |
| close medium | 6.33 |  |  |
| close fast | 6.10 |  |  |

From average MSE calculated, we see that the Correlation method worked well for some of the simpler simulation cases, it is because for those cases delays are applied directly on to the input data therefore Correlation method has good performance. While looking at the open-loop and closed-loop cases (Table 6), the Correlation method got the MSE 19.31 samples, which is not reliable compared with OE and MET.

Table 7: average computation time for 10 iterations

|  |  |  |
| --- | --- | --- |
|  | **Data set size (samples)** | |
| **Methods** | **2000** | **10000** |
| corr | 0.0016 | 0 |
| csd | 0.075 | 0.0781 |
| arx | 0.0563 | 0.1328 |
| oe | 33.8359 | 39.5422 |
| met | 0.3125 | 0.76 |

By looking at the mathematical derivations, the OE method should have about times computation time compared with the ARX method. From the experiments, for 2000 sized data-set the OE spends 33.8356/0.056 = 604 times of time compared with the ARX and 33.8356/0.3125= 108 times for the MET method. After the data-set size increased 5 times to 10000, the average computation time for each iteration did not increase much. This might because of the MATLAB has optimized the usage of RAM. Furthermore, because of this, it is difficult to predict the computation time of these methods.

For OE and MET methods, both show decent performance. From the experiments, the OE had closer results when the simulation delay is small, and the MET had the better results while the simulation delay is large. Since OE method is very time consuming, MET should be the choice if the computation time is considered. From the Table 5, OE and MET method had MSE 8.59 and 6.68, for the open and closed-loop systems datasets, respectively. From the Table 1 we can find that the OE results is usually smaller than simulated delay and the MET result is larger than the simulated result. Therefore, averaging the delay of OE and MET is worthwhile trying, which got the delay of 6.06, and is better than either MSE or OE.

**Conclusions and future work**

In this experiment, five different method of calculating time-delay was applied on six different types of datasets. Overall, the OE and MET method had very good performance compared with others. Also, from the experiments, combing the OE and MET and averaging the result calculated from both can get an even better result. However, the OE method is very time-consuming compared with other methods. While the time consumption is considered, the MET method should be used. The following tasks are planned:

1. Test methods on CSTR model.
2. Write the python applications for different methods.

# **Appendix**

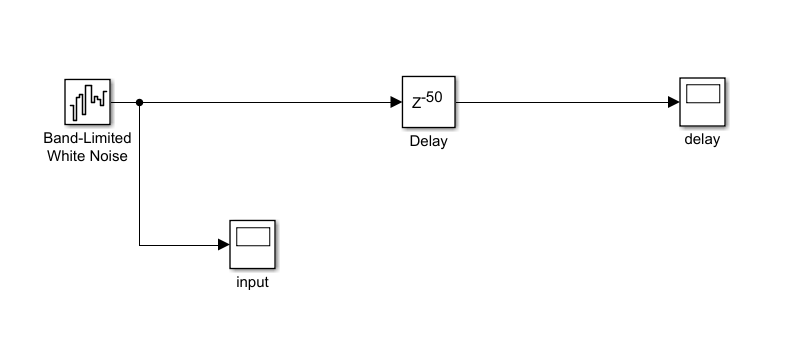


Figure 1: Random noise Data generator

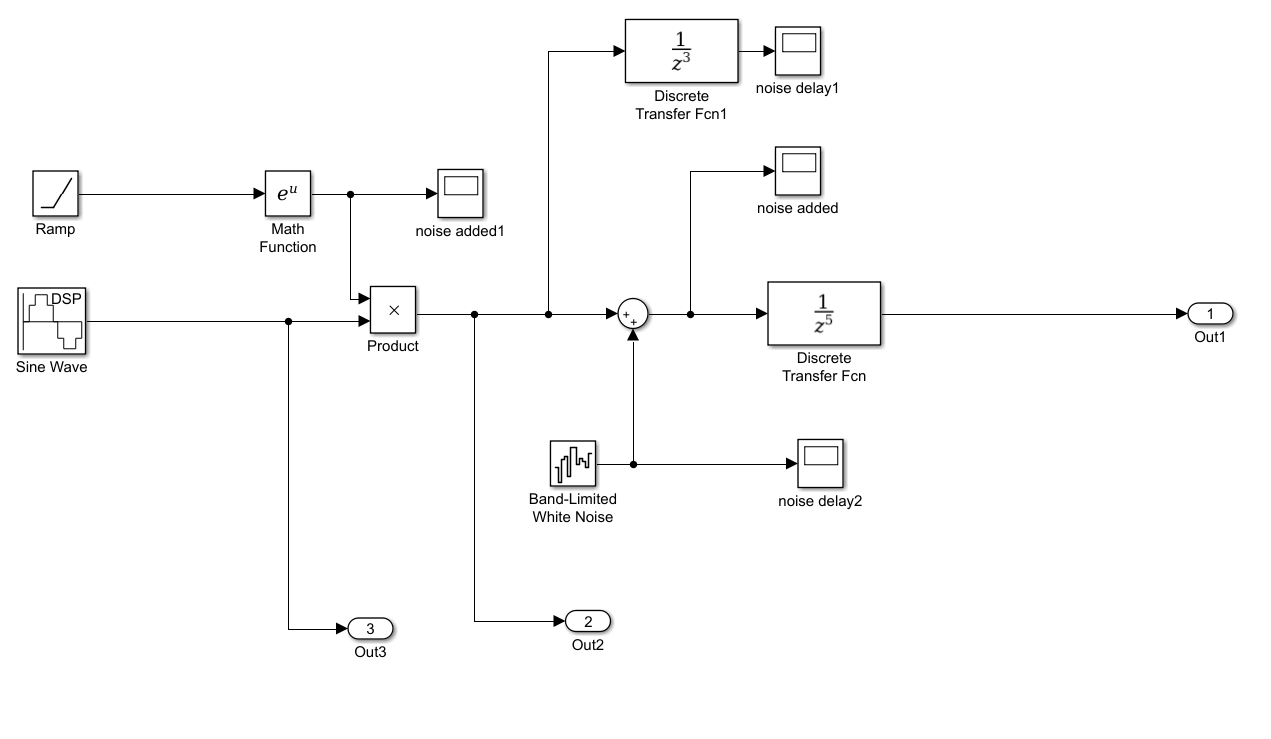


Figure 2: Sin Damping Data generator

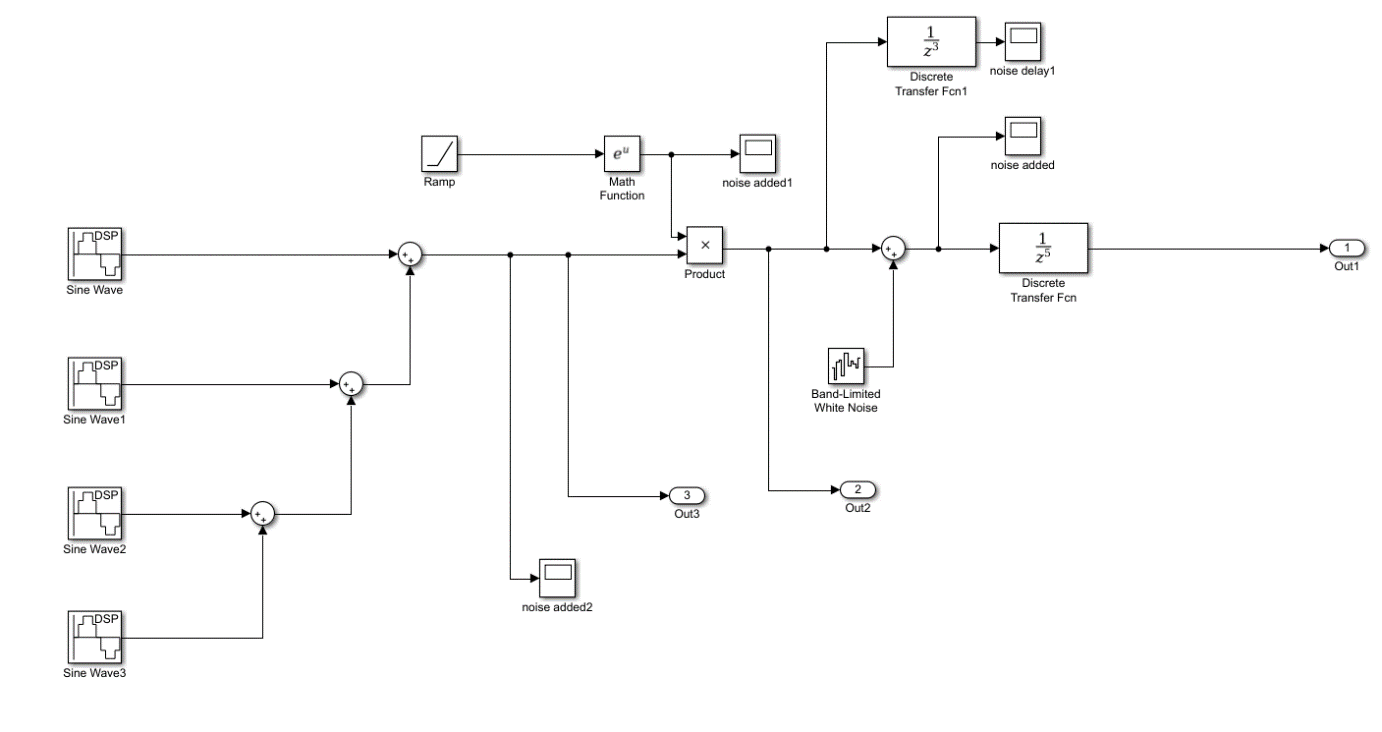


Figure 3: Multiple sin damping Data generator

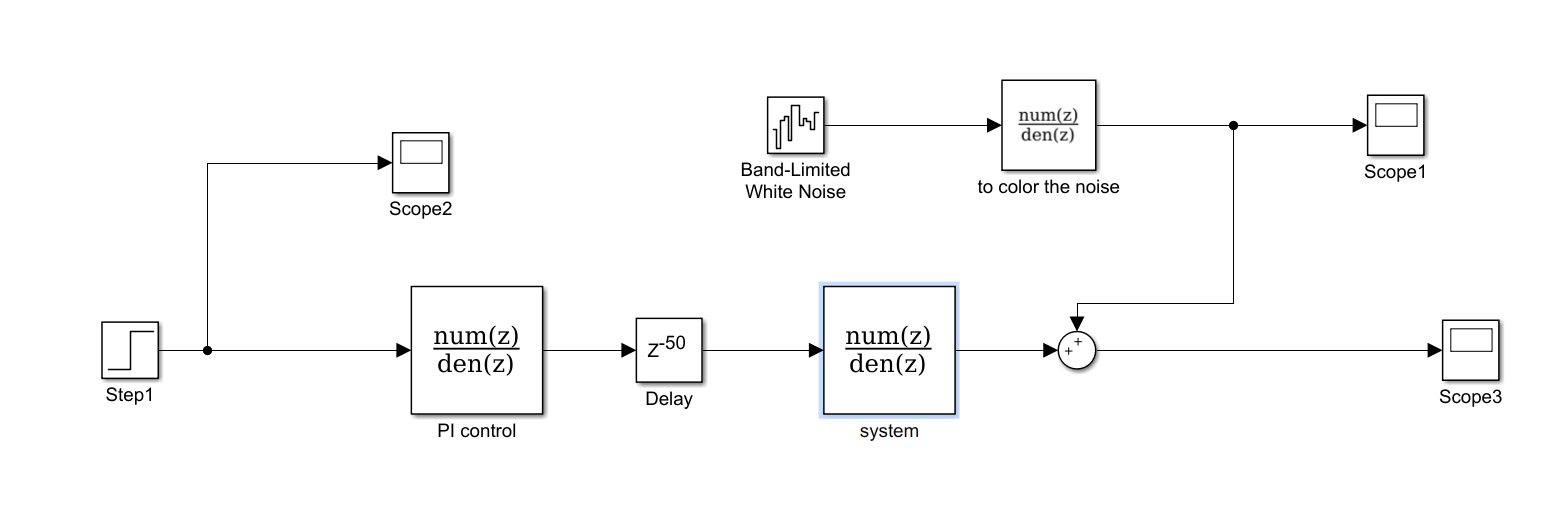


Figure 2: Open loop Data generator

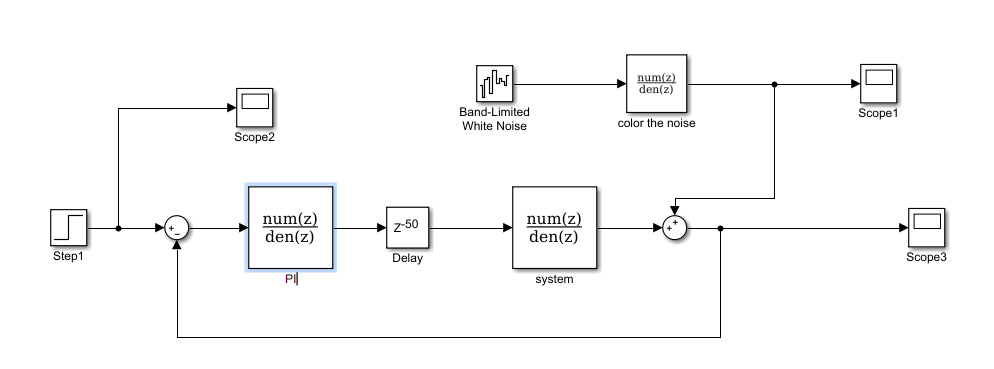


Figure 2: Closed loop Data generator

Pseudo code:

1. Correlation method:

Program corr\_method(input, output)

k = length of input and output

for i =1 to k

c[k] = correlations for (Inuput[k:end] and Output)

k\_max = index of the largest c[k]

delay = k\_max

1. Coherence method:

Program csd\_method(input,output)

zero mean the input and output

I\_f = Fourier transferred input

I\_o = Fourier transferred output

k = length of I\_f and I\_o

for i = 1 to k

c[k] = correlations in frequency domain for (I\_f[k:end] and I\_o)

k\_max = index of the largest |c[k]|

phi = angle of the correlation at i\_max

delay = phi/(2\*pi\*frequency)

1. Auto Regressive model with eXtra input

Program arxstructd(input,output)

define: na – Order of polynomial A(q)

nb – Order of polynomial B(q) + 1

nk – range of fixed leading zeros of the B polynomial

nn = [na,nb,nk]

loss function V = arxstruct([output,input],nn)

order of ARX nn\_sel = selstruct(V)

delay = nn\_sel[3]

1. Output Error model

Program oestructd(input,output)

define: nb – Order of polynomial B(q) +1

nf – Order of polynomial F(q)

nk – range of fixed leading zeros of the B polynomial

for i = 1 to length(nk)

nn[i] = [nb,nf,nk[i]]

estimated system: Model = oe([output,input],nn[i])

loss function V[i] = Model.NoiseVariance

k\_min = index of the minimum V[i]

delay = nk[k\_min]

1. MET

Program met1structd(input,output)

define:

order: order of estimated model

modelSs = estimate state space model: n4sig(output,input ..)

[A,B,C,D,E] = coefficient of polynomial modal of modelSs

uFilt = filter the input by C

yFilt = filter the output by C

do arxstructed(uFilt,yFilt)