System Identification Techniques

Statistics Directed Reading Program Autumn 2024

Mentor: Yash Talwekar

Mentee: Yash Anand

UNIVERSITY *of* WASHINGTON

Understanding System Identification

Definition: System identification involves building mathematical models of dynamic systems using observed data.

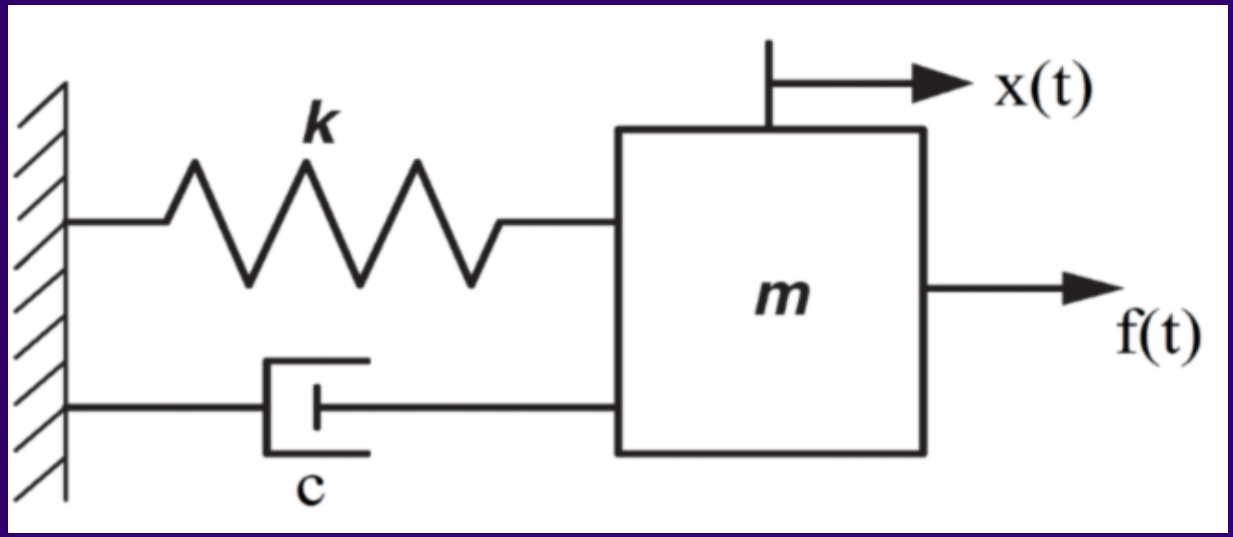
In simpler words : System identification is figuring out how something works by looking at what you put into it and what comes out. For example, how a machine or system behaves by watching how it reacts when you press buttons or give it instructions.

- **Goal of Presentation:** Explain key concepts and apply system identification to a simple example.

Why Do We Need System Identification?

- **Why It's Important:**
  - > Helps understand system behavior.
  - > Identified models can forecast how a system will respond to future inputs.
  - > Helps in control system design and optimization to improve performance.
  - > In systems where properties change over time, identification can help update models to stay accurate.
  - > System identification can assist in simulations and what-if scenarios, guiding better decisions.

Spring-Mass-Damper System as an Example



$$m * x'' + c * x' + k * x = F(t)$$

Spring Force (kx): The restoring force from the spring, proportional to the displacement x. The larger the displacement, the greater the spring force trying to bring the mass back to equilibrium.

Damping Force (c*x'): The force from the damper, which resists the motion and is proportional to the velocity (x'). This force acts to reduce the oscillations over time.

Inertial Force (m*x''): The force required to accelerate the mass, according to Newton's second law.

Relation to System Identification:

- System parameters like mass, damping, and spring constant are similar to the coefficients in mathematical equations.

- Demonstrates the dynamics of input (force) and output (displacement) and how they relate with each other.

Core Concepts in System Identification

- **Linear and Multivariate Systems:**
  - Linear: System of equations where the relationship between the input and output data is linear.
  - Multivariate: Systems with multiple inputs or outputs. Analyses the relationship between multiple variables simultaneously.

- **Stationarity:**
  - Assumes statistical properties to the system(mean, variance) are constant over time.

- **Autoregression:**
  - Models current output as a function of the past output(Lags).

Equation Error Model Structure

- Definition:
  - A mathematical framework for System identification where errors in equations represent model inaccuracies.
  - Example Equation: $$y_n = a_1 y_{n-1} + a_2 y_{n-2} + b_1 u_{n-1} + e_n$$
  - Name: Auto regressive model with exogenous input (ARX)
  - Purpose : Estimate parameters a_1, a_2, b_1 to define System behavior

Parameter Estimation Process

- **Steps:**
  - > Collect input-output data.
  - > Assume an equation structure(e.g., AR model)
  - > Solve for coefficients: a_1, a_2, b_1 using the optimization techniques.
    - a_1 represents how much the most recent past output affects the current output.
    - a_2 indicates the influence of the second most recent past output on the current output
    - b_1 Represents how the most recent input affects the current output

AR model Equation: $$y_n = a_1 y_{n-1} + a_2 y_{n-2} + b_1 u_{n-1}$$

Matrix form : $\mathbf{y} = \mathbf{A}\theta$

design matrix

$$\begin{bmatrix} y_3 \\ y_4 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} y_2 & y_1 & u_2 \\ y_3 & y_2 & u_3 \\ \vdots & \vdots & \vdots \\ y_{N-1} & y_{N-2} & u_{N-1} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ b_1 \end{bmatrix}$$

output vector

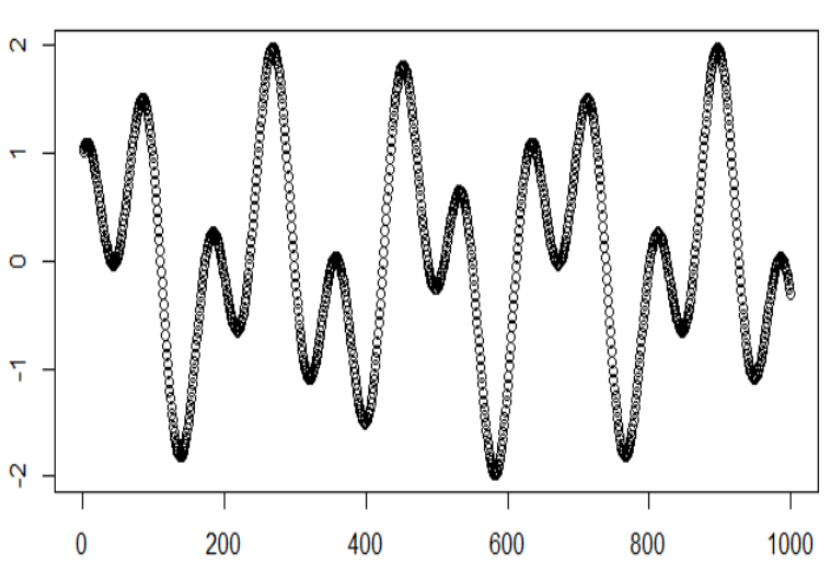
parameter vector

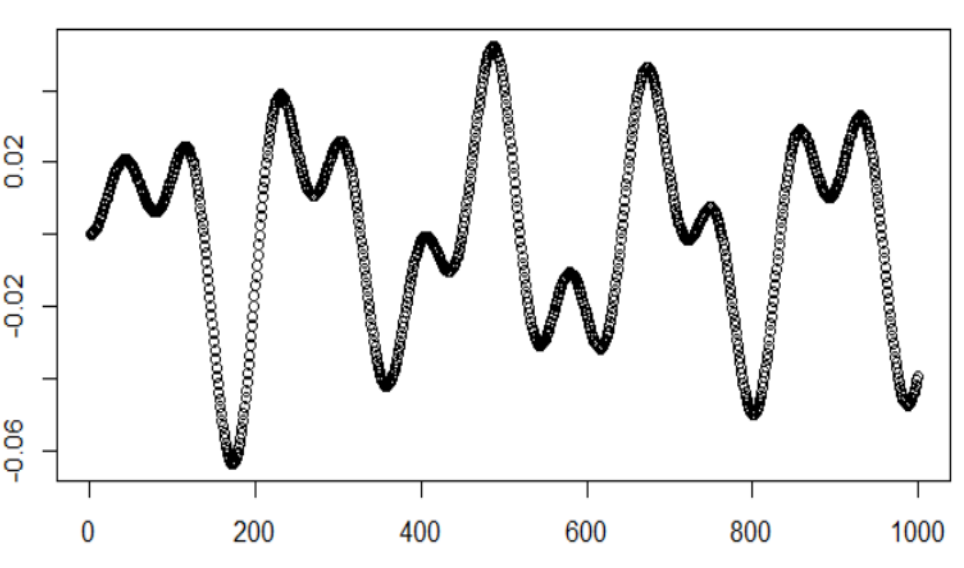
Line of Best fit : $\theta = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y}$ **(Also known as Moore Penrose psuedo Inverse)**

Spring mass damper system without exogenous input

X1.0000000000000000e.00 <dbl>
1.027547
1.050180
1.067909
1.080768
1.088811
1.092119
1.090793
1.084958
1.074759
1.060362

X0.0000000000000000e.00 <dbl>
0.0000500000
0.0001507523
0.0003026292
0.0005056263
0.0007593677
0.0010631138
0.0014157706
0.0018159010
0.0022617383
0.0027512008





Making the Design Matrix (A) Using Input Output Values

```r
rows <- nrow(spring_mass_damper)
columns <- ncol(spring_mass_damper)
res <- matrix(0, nrow = rows, ncol = 3)


res[2, 1] <- spring_mass_damper[1, 2]
res[2, 2] <- 0
res[2, 3] <- spring_mass_damper[1, 1]
for (i in 3:rows) {
  res[i, 1] <- spring_mass_damper[i - 1, 2]
  res[i, 2] <- spring_mass_damper[i - 2, 2]
  res[i, 3] <- spring_mass_damper[i - 1, 1]
}

nrow(res)
ncol(res)
head(res, 10)


```

```
[1] 999
[1] 3
              [,1]         [,2]     [,3]
 [1,] 0.0000000000 0.0000000000 0.000000
 [2,] 0.0000500000 0.0000000000 1.027547
 [3,] 0.0001507523 0.0000500000 1.050180
 [4,] 0.0003026292 0.0001507523 1.067909
 [5,] 0.0005056263 0.0003026292 1.080768
 [6,] 0.0007593677 0.0005056263 1.088811
 [7,] 0.0010631138 0.0007593677 1.092119
 [8,] 0.0014157706 0.0010631138 1.090793
 [9,] 0.0018159010 0.0014157706 1.084958
[10,] 0.0022617383 0.0018159010 1.074759
```

Line of Best Fit (Pseudo Inverse)

$$\boldsymbol{\theta} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y}$$

```{r}
pseudo <- solve(t(res) %*% res) %*% t(res)
theta <- pseudo %*% as.matrix(spring_mass_damper[,2])
theta
```

```
          [,1]
[1,]  1.98750
[2,] -0.99000
[3,]  0.00005
```

Verifying the Model

- Use Mean Squared Error (MSE) to evaluate model accuracy.

$$\text{MSE} = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2$$

```r
MSE <- sum(((as.matrix(res) %*% as.matrix(Beta)) - spring_mass_damper[, 2])^2)/rows
MSE
```

```
[1] 2.502503e-12
```

- **Importance:**
  - > Ensures the model represents the system dynamics effectively.

Generalizing the Model

- By generalizing I mean extending the structure of the model. More than 3 coefficients.

$$y_n = a_1 y_{n-1} + a_2 y_{n-2} + a_3 y_{n-3} + b_1 u_{n-1} + b_2 u_{n-2}$$

- Why is it important:
  - Some systems need more past data to make accurate predictions. Adding more terms helps the model understand this complexity.
  - More terms can help the model make predictions that are closer to the real data.
  - A generalized model can be used for a wider variety of systems.
  -

Generalized model Code

```r
a_max = 3
b_max = 2
for (a in 1:a_max) { ## no of a values
  for(b in 1: b_max) { ## no of b values
    rows <- nrow(spring_mass_damper) ## no of rows in the dataset
    columns <- a + b ## no of columns of the matrix that we are creating in the following iteration
    res <- matrix(0, nrow = rows, ncol = columns) ## creating a  matrix to fill the values in
    for (n in 1:rows) { ## looping through the values in the row of the matrix we are creating
      for(a_col in 1:a) { ## looping through the values in the column of the matrix we are creating
        if(n - a_col > 0) {
          res[n, a_col] <- spring_mass_damper[n - a_col, 2]
        }
      }
      for (b_col in 1:b) {
        if(n - b_col > 0) {
          res[n, a + b_col] <- spring_mass_damper[n - b_col, 1]
        }
      }
    }
    pseudo <- solve(t(res) %*% res) %*% t(res) # same process as before for every possible situation of no of a values and b values
    coefficients_values <- as.matrix(pseudo) %*% as.matrix(spring_mass_damper[,2])
    print(coefficients_values)
    mse <- sum(((as.matrix(res) %*% as.matrix(coefficients_values))- spring_mass_damper[, 2])^2)/rows
    print(mse)
  }
}
```

What it does ?

- It basically does the same process as above mentioned, but it gives us every combination of the coefficients.

Example :

```
[1] " a 1  b 1"
[1] " a 1  b 1 b 2"
[1] " a 1 a 2  b 1"
[1] " a 1 a 2  b 1 b 2"
[1] " a 1 a 2 a 3  b 1"
[1] " a 1 a 2 a 3  b 1 b 2"
```

Results:

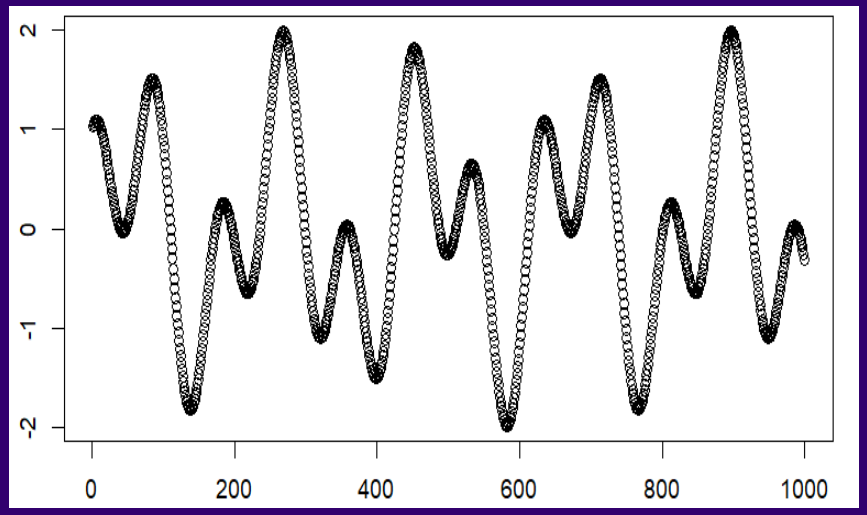
```
                [,1]
[1,] 0.9972073750
[2,] 0.0003912299
[1] 1.329103e-06
                [,1]
[1,]  0.992350932
[2,] -0.008592232
[3,]  0.009027560
[1] 1.032129e-06
          [,1]
[1,]  1.98750
[2,] -0.99000
[3,]  0.00005
[1] 2.502503e-12
                [,1]
[1,]  1.98750e+00
[2,] -9.90000e-01
[3,]  5.00000e-05
[4,]  4.40842e-14
[1] 2.502503e-12
                [,1]
[1,]  1.987498e+00
[2,] -9.899963e-01
[3,] -1.874823e-06
[4,]  5.000009e-05
[1] 2.502505e-12
                [,1]
[1,]  1.987495e+00
[2,] -9.899897e-01
[3,] -5.157607e-06
[4,]  5.000009e-05
[5,]  1.722233e-10
[1] 2.502503e-12
```

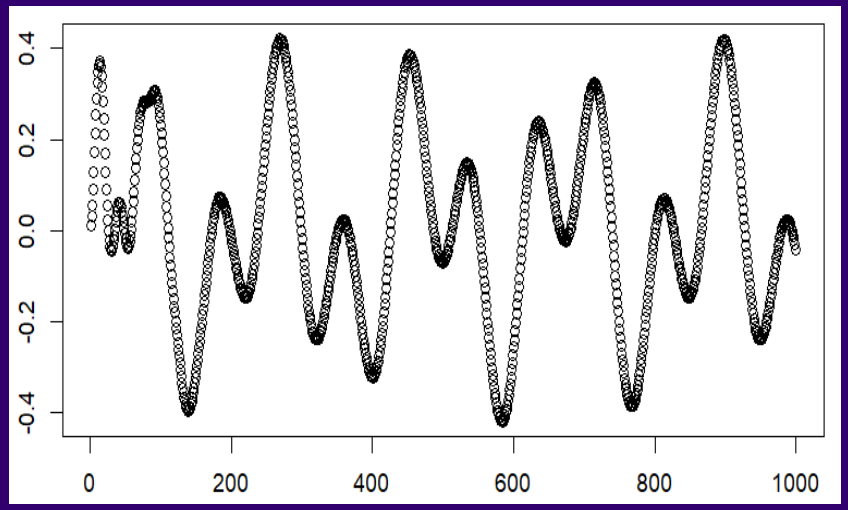
Two other example datasets with exogenous input.

- Dataset one for predicting the parameters.

X1.0000000000000000e.00 <dbl>
1.027547
1.050180
1.067909
1.080768
1.088811
1.092119
1.090793
1.084958
1.074759
1.060362

X1.077118007797454912e.03 <dbl>
0.01037433
0.03011889
0.05538561
0.09133380
0.12894794
0.17185662
0.21360755
0.25426266
0.29059211
0.32390446

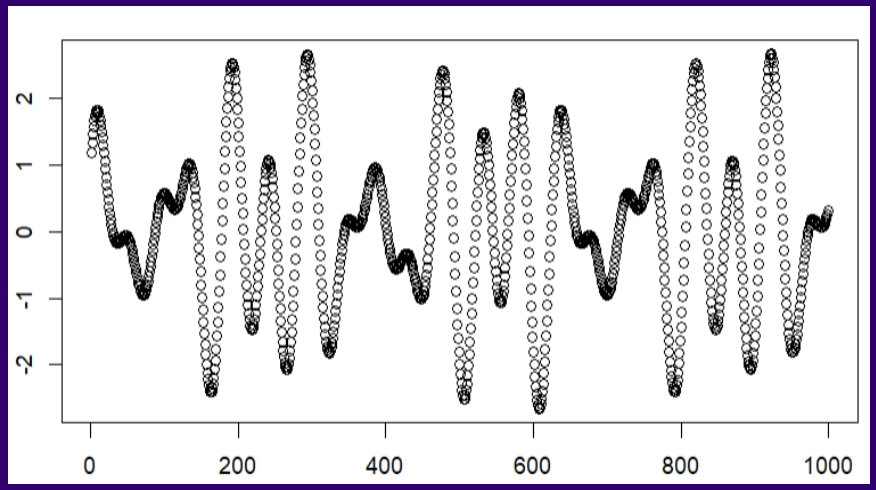


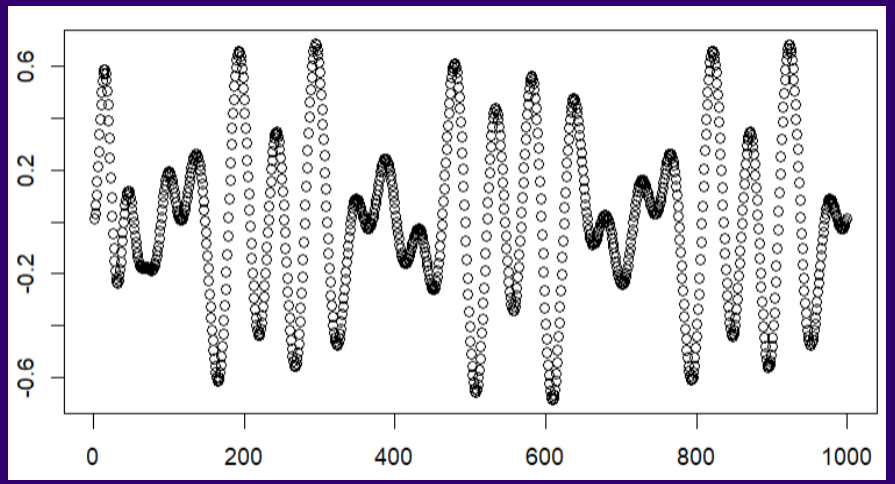


- Dataset two for validating the model.

X1.0000000000000000e.00 <dbl>
1.171283
1.324163
1.457412
1.570114
1.661669
1.731791
1.780507
1.808145
1.815322
1.802924

X5.401779389559876977e.04 <dbl>
0.01023215
0.03277909
0.06432833
0.10213662
0.15378339
0.21027305
0.27053399
0.33474424
0.39553997
0.45209346





Applying same process but with two datasets

```
a_max = 3
b_max = 2
for (a in 1:a_max) { ## no of a values
  for(b in 1: b_max) { ## no of b values
    rows <- nrow(spring_mass_damper_2) ## no of rows in the dataset
    columns <- a + b ## no of columns of the matrix that we are creating in the following iteration
    res <- matrix(0, nrow = rows, ncol = columns) ## creating a  matrix to fill the values in
    res_validation <- matrix(0, nrow = rows, ncol = columns)
    for (n in 1:rows) { ## looping through the values in the row of the matrix we are creating
      for(a_col in 1:a) { ## looping through the values in the column of the matrix we are creating
        if(n - a_col > 0) {
          res[n, a_col] <- spring_mass_damper_2[n - a_col, 2]
          res_validation[n, a_col] <- spring_mass_damper_validation[n - a_col, 2]
        }
      }
      for (b_col in 1:b) {
        if(n - b_col > 0) {
          res[n, a + b_col] <- spring_mass_damper_2[n - b_col, 1]
          res_validation[n, a + b_col] <- spring_mass_damper_validation[n - b_col, 1]
        }
      }
    }
    pseudo <- solve(t(res) %*% res) %*% t(res) # same process as before for every possible situation of no of a values and b
values
    coefficients_values <- as.matrix(pseudo) %*% as.matrix(spring_mass_damper_2[,2])
    print(coefficients_values)
    mse <- sum(((as.matrix(res_validation) %*% as.matrix(coefficients_values)) - spring_mass_damper_validation[, 2])^2)/rows
    print(mse)

  }
}
```

Results :

```
             [,1]
[1,] 0.76557431
[2,] 0.05008306
[1] 0.001008722
             [,1]
[1,]  0.9384638
[2,]  0.1650810
[3,] -0.1519288
[1] 0.0002856018
              [,1]
[1,]  1.87630245
[2,] -0.93307690
[3,]  0.01145683
[1] 6.249205e-06
               [,1]
[1,]  1.857445736
[2,] -0.911094932
[3,]  0.017229382
[4,] -0.006424158
[1] 6.201679e-06
              [,1]
[1,]  1.23965543
[2,]  0.27975548
[3,] -0.60339702
[4,]  0.01685009
[1] 3.395649e-06
                [,1]
[1,]  1.2395002845
[2,]  0.2794338469
[3,] -0.6027528263
[4,]  0.0170999014
[5,] -0.0002844123
[1] 3.396089e-06
```

For understanding :

```
[1] " a 1  b 1"
[1] " a 1  b 1 b 2"
[1] " a 1 a 2  b 1"
[1] " a 1 a 2  b 1 b 2"
[1] " a 1 a 2 a 3  b 1"
[1] " a 1 a 2 a 3  b 1 b 2"
```

Conclusion:

- **Overfitting Problem**: When too many parameters are added to the model, the model can start to fit the noise in the data instead of the actual system behavior.
- **Result**: This leads to a **lower Mean Squared Error (MSE)** on the training data, but poor performance on new, unseen data because the model is too complex for the real-world system.
- **Why It Happens**: The model becomes overly sensitive to small fluctuations or outliers in the training data, resulting in a model that doesn't generalize well.
- **Key Insight**: While reducing MSE on the training data may seem good, it doesn't necessarily mean the model is better; it might just be memorizing the data.

The End!