Effective Teaching using MATLAB and SIMULINK

MATLAB Expo, Hyderebad

Arun K. Tangirala

Department of Chemical Engineering
Indian Institute of Technology Madras

April 27, 2017

Discretization

The zero-order hold (ZOH) discretization of a continuous-time system results in the discretized system

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}u(t) \\ y(t) &= \mathbf{C}\mathbf{x}(t) + \mathbf{D}u(t) \end{aligned} \quad \xrightarrow[T_s]{\text{ZOH}} \quad \begin{aligned} \mathbf{x}[k+1] &= \mathbf{A}_d\mathbf{x}[k] + \mathbf{B}_d u[k] \\ y[k] &= \mathbf{C}\mathbf{x}[k] + \mathbf{D}u[k] \end{aligned} \tag{1}$$

under the mapping

$$\boxed{\mathbf{A}_d = e^{\mathbf{A}T_s}; \quad \mathbf{B}_d = (e^{\mathbf{A}T_s} - \mathbf{I})\mathbf{A}^{-1}\mathbf{B}} \tag{2}$$

To be able to use the results of discretization, one needs to compute the **exponential of a matrix.**

Exponential of a matrix in MATLAB

```
% Sample matrix
A = [1 2 ; 3 4];
% Exponential of a matrix the correct way
eA = expm(A);
% The incorrect way
eA2 = exp(A);
isequal(eA,eA2)
```


Computation of $e^{\mathbf{A}t}$

The matrix exponential is defined as an infinite-order Taylor's series expansion

$$e^{\mathbf{A}t} = \mathbf{I} + \mathbf{A}t + \mathbf{A}^2\frac{t^2}{2!} + \cdots \tag{3}$$

1. **Eigenvalue decomposition method:** The exponential is computed as

$$\boxed{e^{\mathbf{A}t} = \mathbf{V}e^{\mathbf{\Lambda}t}\mathbf{V}^{-1}} \tag{4}$$

where $\mathbf{V}$ and $\mathbf{\Lambda}$ are the eigenvector and eigenvalue (diagional) matrices of $\mathbf{A}$. The identity in (4) uses the Taylor's series expansion in (3) and the fact that

$$\mathbf{V}^{-1}\mathbf{A}\mathbf{V} = \mathbf{\Lambda}$$

Computation of $e^{\mathbf{A}t}$ . . . contd.

2. **Laplace Transform method:** The method uses a Laplace-transform route

$$\boxed{e^{\mathbf{A}t} = \mathcal{L}^{-1}\left\{(s\mathbf{I} - \mathbf{A})^{-1}\right\}} \tag{5}$$

The identity in (5) is based on a simple fact that

$$\mathcal{L}\left\{e^{\mathbf{A}t}\right\} = s\mathbf{I} - \mathbf{A}$$

In fact, the second method is preferable and more elegant than the eigenvalue decomposition method.

Solving a set of linear equations

Solve the following set of linear equations:

$$\begin{aligned} 400x_1 - 201x_2 &= 200 \\ -800x_1 + 401x_2 &= -200 \end{aligned}$$

Call this solution $\mathbf{x}_0$.

Further, suppose the $A(1,1)$ element has a slight uncertainty in it. For a possible value of $A(1,1) = 401$, determine the solution (call it $\mathbf{x}_1$).

- How much does $\mathbf{x}_1$ differ from the previous solution $\mathbf{x}_0$?
- Can you explain the reason for the difference?

Solving overdetermined / underdetermined problems

Overdetermined:

$$\mathbf{Ax} \approx \mathbf{b}, \qquad \mathbf{A} \in \mathbb{R}^{N \times M}, \mathbf{x} \in \mathbb{R}^{M \times 1}, \quad \mathbf{b} \in \mathbb{R}^{N \times 1}, \qquad N > M$$

- Least squares methods can be used to solve this problem.
- Solution involves **pseudo-inverse** of $\mathbf{A}$.

However,

i. The matrix $\mathbf{A}$ could be rank deficient!

ii. When $N \times M$ or when rank$(\mathbf{A}) < M$, we run into **underdetermined** problems.

Roots of polynomials

```
% Find roots of a polynomial: x^3 + 6x^2 + 11x + 6
xr = roots([1 6 11 6]) % Takes in the coefficients
pxval = poly(xr);       % Check if you get back the same answ
% Alternatively find roots in a region
px = @(x) x.^3 + 6*x.^2 + 11*x + 6;
xvec = linspace(-4,4,100)
plot(xvec,px(xvec)); hold on
plot(xvec,zeros(length(xvec),1),'r--')
fzero(px,-1.4)
```

In presence of uncertainties?

On several occasions, the polynomial may be only known with error

i. Uncertainties in the coefficients of the polynomial
ii. Polynomial may have been obtained through data fitting

```
% Coefficients of polynomial
Pc = [1 6 11 6];
px = @(x,Pc) Pc(1)*x.^3 + Pc(2)*x.^2 + Pc(3)*x + Pc(4);
% Evaluate over possiblities
delpc2 = 0.2*randn(100,1);
xrvec = [];
for k = 1:length(delpc2)
Pcp = Pc + [0 delpc2(k) 0 0];
xrtemp = fzero(@(x) px(x,Pcp),-1.7);
xrvec = [xrvec; xrtemp];
end
% Plot the solutions
plot(Pc(2) + delpc2,real(xrvec),'x')
```


Solving non-linear equations: Example

Let's solve two non-linear equations:

$$2x_1 - x_2 - e^{cx_1} = 0$$
$$-x_1 + 2x_2 - e^{cx_2} = 0$$

First, set up the function.

```
function Fx = myfun(x,c)
% To be passed on to fsolve
%
% Parameter value of 'c' has to be set by the user

% Equations
Fx = [];
Fx(1) = 2*x(1) - x(2) - exp(c*x(1));
Fx(2) = -x(1) + 2*x(2) - exp(c*x(2))
```

Now pass on this function "anonymously" to `fsolve` as shown below.

Solving non-linear equations: Example (cont.)

```
% Script to demonstrate use of fsolve

% Set value of c
c = -2;
% Call fsolve by passing the handle of myfun
xsol = fsolve(@(x) myfun(x,c),[-3 ; -4])
```

- Make it a point to check if the solution satisfies the given equations.

```
% Pass the solution to the function
myfun(xsol,c)
```

Exercise: Solve the non-linear equations:

$$x_1^2 + x_2^2 - 50 = 0$$
$$x_1 x_2 - 25 = 0$$

Simulation of dynamical systems

Numerical Integration

Solving a set of ODEs

MATLAB offers powerful tools for numerical integration (of ordinary differential equations). Specific routines for handling **stiff** systems are also available. Some of the routines include `ode45`, `ode23`, `ode45s`, `ode23tb` and so on. Also see `odeexamples` for a demo of some interesting dynamical systems.

Let's take up a simple first-order dynamical system concerning the level dynamics in a flow buffer process:

$$\frac{dh(t)}{dt} = \frac{F_i(t)}{A_c} - \frac{C_v}{A_c}\sqrt{h(t)} \tag{6}$$

It is known that $A_c = 2$ units and $C_v = 1.5$ units. We shall learn how to

1. Trace out the level profile to changes in inlet flow
2. Determine the steady-state conditions
3. Develop a **linearized** model of the system (around an operating point)

using MATLAB.

Numerically solving an ODE

ODEs can be numerically solved using the previously mentioned routines. However, there exists an alternative route, which is through SIMULINK, a powerful graphical user interface (GUI) for simulating dynamical systems.

- SIMULINK is a powerful simulator integrated with MATLAB. It allows the user to conveniently build and simulate complex process flowsheets by means of block diagrams.
- SIMULINK consists of an extensive set of libraries that contain different types of basic blocks. Type `simulink` at the MATLAB command to bring up the libraries and a new model window.

A Simulink Model

- A simulink model comprises a set of connected blocks with arrows indicating the direction of flow. Each block in the model has a set of parameters that can be accessed by double clicking on that block.
- There are two types of parameters (properties): (i) Block parameters (properties) and (ii) Simulation parameters. The block properties refer to the properties of the systems and signals that constitute the model (block diagram).
- Simulation parameters refer to the configuration of the simulation such as start/stop time, type of numerical technique, etc.
- Numerical values/workspace variables in MATLAB can be passed as parameters.

Introduction to DEE

The **Differential Equation Editor** as the name suggests, allows us to directly enter the ODEs. This is one of the most useful features of this interface.

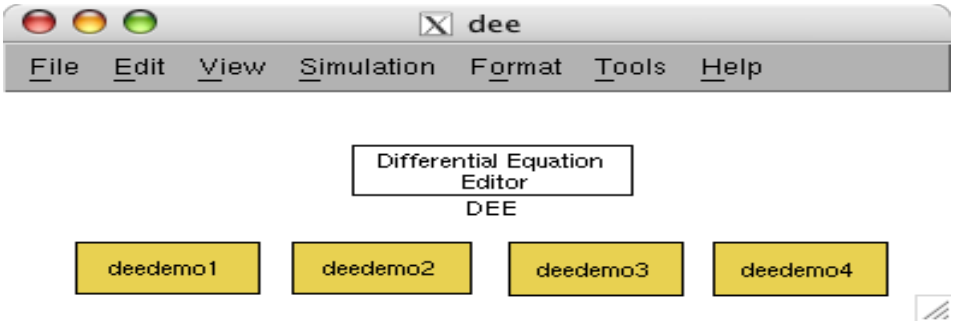
Before we can use the DEE, the ODEs should be written in the standard form:

$$\frac{dx}{dt} = f(x, u, p)$$
$$y = g(x, u, p)$$

Essentially, the LHS of the **state equation** should only contain derivatives and all other terms should be taken over to the right hand side of the equation.

Introduction to DEE (cont.)

The DEE is invoked by typing: `dee`. This opens up a new window



A snapshot of the DEE window when invoked

Setting up the equations

The notation used in the DEE is standard and fixed. States are always denoted by x, inputs by u and outputs by y. The DEE allows us to (i) provide initial conditions and / or (ii) provide external inputs to the system.

1. In order to set up a DEE system, first choose **File** $\rightarrow$ **New** $\rightarrow$ Model. Drag and drop the **DEE** block from the DEE window into the New Model window.
2. Double click on the DEE block to start entering the differential equations. Only the RHS of the differential equations need to be supplied
3. Parameters can be specified as numerical values OR variables in the workspace. However, in the latter case, the parameters have to be declared in the workspace.
4. Note: It is useful to double click on the main block in each of the demos to see how the differential equations have been written.

Finding the steady-state using `trim`

- MATLAB's linearization toolkit contains a function called `trim`. The function `trim` works only with Simulink Models.
- Given a Simulink block diagram, it calculates the steady-state conditions.
- In order to use `trim`, the Simulink block diagram should be redrawn with the In and Out Ports specified at the appropriate nodes.

The syntax is as follows:
`[x,u,y,dx]=trim(sys)` OR `[x,u,y,dx]=trim('sys',x0,u0)` OR
`[x,u,y,dx] = trim(sys,x0,u0,y0,ix,iu,iy)`
where the `sys` is the Simulink Model, supplied in single quotes

Syntax for `trim`

- In the second form of usage, x0 and u0 are initial guesses for x and u respectively. The function returns only one steady-state solution. Since multiple steady-states can exist, it may be necessary to specify either the value of the output, and/or the input, and/or the states (check for degrees of freedom!)
- The third form of the usage of `trim` in fact allows us to specify the states and/or the inputs and/or the inputs at which we intend to find a steady-state. While the arguments `x0,u0,y0` allow us to specify the values (or could be initial guesses), the additional arguments `iu, ix, iy` allow us to specify which of `(x0,y0,u0)` have been specified and which of these have been passed on as initial guesses only.

Using `trim` to obtain steady-state

- For the non-linear tank level model, suppose we wish to find a steady-state, then simply type `[x,u,y,dx] = trim(sys)` at the command prompt, where `sys` is the model name supplied in quotes.
- Suppose we wish to find a steady-state value such that the output (level) is at 9 units, then we type: `[x,u,y,dx] = trim(sys,1,1,9,[],[],1)` at the MATLAB command prompt.
- The above syntax means that we have given initial guesses of $x = 1$, $u = 1$ and asking it to find the steady-state such that the output is $y = 9$ units.
- The values passed to the function override the values of initial conditions specified in the DEE block. This function can be used with any Simulink block diagram.

Linearization using `linmod`

The MATLAB function that performs the linearization using a SIMULINK model is `linmod`. The syntax for this function is:

`[A,B,C,D]=linmod('sys')` OR `[A,B,C,D]=linmod('sys',x,u)`

- The first usage simply returns the state-space matrices of the linearized model for the block diagram specified in the Simulink Model 'sys'. With such a call, the function `linmod` assumes the operating conditions to be those specified in the DEE block.

Linearization using `linmod` (cont.)

- The second usage on the other hand allows the user to specify a set of operating conditions around which the user intends to obtain a linearized model. This is achieved by passing the values of the states and the inputs through the arguments `(x,u)`
- For the non-linear tank level setup, the following commands may be used:
  `[A,B,C,D] = linmod('nltank_lm')` wherein we have assumed that the initial state has been set to 16 in the DEE block.

Exercises

1. Simulate, determine the steady-state and develop a linearized model for a two tank system connected in series.
   - The cross-sectional areas are $A_1 = 5$ ft^2 and $A_2 = 10$ ft^2 and the valve constants are $C_{v1} = 2.5$ and $C_{v2} = 5/\sqrt{6}$ units.
2. A **batch reactor** carries out the reactions

$$A \underset{k_{1r}}{\overset{k_{1f}}{\rightleftharpoons}} B \xrightarrow{k_2} C$$

   Assume that each of the reactions is first-order, the reactor operates at constant volume, and there are no feed or product streams

Exercises (cont.)

The modelling equations are:

$$\frac{dC_A}{dt} = -k_{1f}C_A + k_{1r}C_B$$
$$\frac{dC_B}{dt} = k_{1f}C_A - k_{1r}C_B - k_2C_B$$
$$\frac{dC_C}{dt} = k_2C_B$$

where C_A, C_B and C_C represent the concentrations (mol/volume) of components A, B and C respectively.

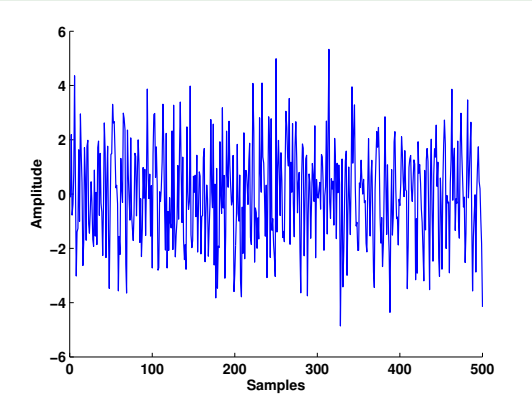
1. For $k_{1f} = 2$, $k_{1r} = 1$, and $k_2 = 1.25\ hr^{-1}$, use ode45 to solve for the concentrations as a function of time. Assume an initial concentration of A of $C_{A0} = 1$ mol/liter. Then plot the concentrations as a functions of time. For what time is the concentration of B maximized?

Motivating Spectral (frequency-domain) Analysis

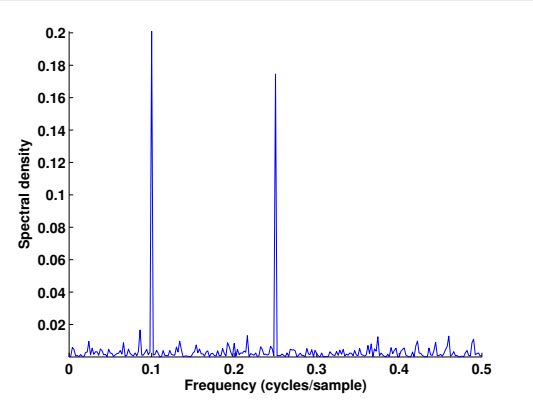
Power Spectral density: Example

Mixture of sines embedded in noise

Series contains $N = 500$ samples of a signal $v[k] = \sin(0.2\pi k) + \sin(0.5\pi k) + e[k]$, where $e[k]$ is the GWN process with variance $\sigma_e^2 = 2.25$.



Spectral density $\longrightarrow$



The periodogram shows distinct peaks at $f_1 = 0.1$ and $f_2 = 0.25$ cycles/sample respectively $\Longrightarrow$ the series predominantly contains two periodic components.

Try detecting these components from the series by a visual inspection!

Fourier Transforms

Variant	Representation	Parseval's relation & Signal requirements
Fourier Series	$x(t) = \sum_{n=-\infty}^{\infty} c_n e^{j2\pi n F_0 t}$ $c_n \triangleq \frac{1}{T_p}\int_{T_p} x(t) e^{-j2\pi n F_0 t}\, dt$	$P_{xx} = \frac{1}{T_p}\int_0^{T_p} \lvert x(t)\rvert^2\, dt = \sum_{n=-\infty}^{\infty} \lvert c_n\rvert^2$ $x(t)$ is periodic with fundamental period $T_0 = 1/F_0$
Fourier Transform	$x(t) = \int_{-\infty}^{\infty} X(F) e^{j2\pi F t}\, dF$ $X(F) \triangleq \int_{-\infty}^{\infty} x(t) e^{-j2\pi F t}\, dt$	$E_{xx} = \int_{-\infty}^{\infty} \lvert x(t)\rvert^2\, dt = \int_{-\infty}^{\infty} \lvert X(F)\rvert^2\, dF$ $x(t)$ is aperiodic s.t. $\int_{-\infty}^{\infty} \lvert x(t)\rvert\, dt < \infty$ or $\int_{-\infty}^{\infty} \lvert x(t)\rvert^2\, dt < \infty$ (weaker requirement)
Discrete-Time Fourier Series	$x[k] = \sum_{n=0}^{N-1} c_n e^{j2\pi k n/N}$ $c_n \triangleq \frac{1}{N}\sum_{k=0}^{N-1} x[k] e^{-j2\pi k n/N}$	$P_{xx} = \frac{1}{N}\sum_{k=0}^{N-1} \lvert x[k]\rvert^2 = \sum_{n=0}^{N-1} \lvert c_n\rvert^2$ $x[k]$ is periodic with fundamental period N
Discrete-Time Fourier Transform	$x[k] = \int_{-1/2}^{1/2} X(f) e^{jfk}\, df$ $X(f) \triangleq \sum_{k=-\infty}^{\infty} x[k] e^{-j2\pi f k}$	$E_{xx} = \sum_{k=-\infty}^{\infty} \lvert x[k]\rvert^2 = \int_{-1/2}^{1/2} \lvert X(f)\rvert^2\, df$ $x[k]$ is aperiodic s.t. either $\sum_{k=-\infty}^{\infty} \lvert x[k]\rvert < \infty$ or $\sum_{k=-\infty}^{\infty} \lvert x[k]\rvert^2 < \infty$ (weaker requirement)

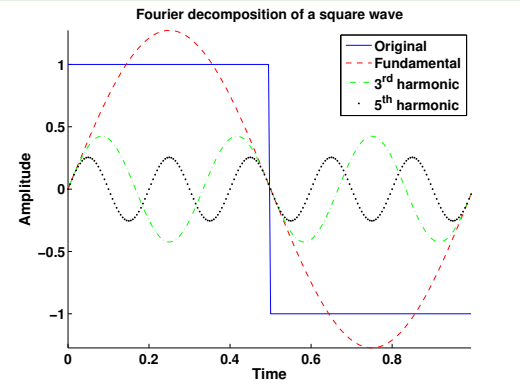
Practical implementation: Discrete Fourier Transform (DFT),

Fourier Transform: Example

The Fourier series representation of the periodic square wave

$x(t) = \begin{cases} 1, & 0 \le t \le 1/2 \\ -1, & 1/2 < t \le 1 \end{cases}$ with period $T_p = 1$ is given by the coefficients

$$c_n = \frac{1}{T_p}\int_0^1 x(t)e^{-j2\pi nt}\,dt = \int_0^{1/2} e^{-j2\pi nt}\,dt - \int_{1/2}^1 e^{-j2\pi nt}\,dt = j\sin\left(\frac{n\pi}{2}\right)\mathrm{sinc}\left(\frac{n\pi}{2}\right)$$



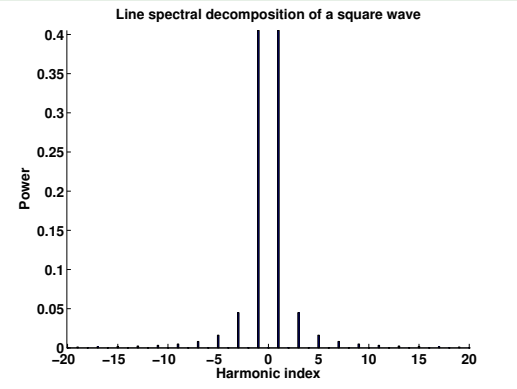


Figure: Power spectral and Fourier decomposition of the square wave

Revisiting example: Cleaning up the measurement

Suppose now that the objective is to extract the periodic components from the measurement in the periodicity detection example.

A standard approach is to first zero out the power spectrum at all frequencies except at those very close to and including f_1 and f_2. Subsequently, reconstruct the signal using the phase information and inverse FT.

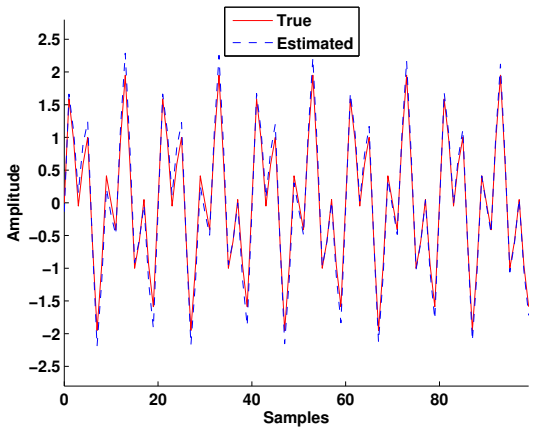


Figure: Comparing the signal estimate with the true signal

MATLAB Script

```
% Generate the measurement
kvec = (0:499)';
vk = sin(2*pi*0.1*kvec) + sin(2*pi*0.25*kvec) + randn(500,2.2

% Compute periodogram
N = length(vk);
[Pxx,wvec] = periodogram(vk-mean(vk),[],N,1);
figure; plot(wvec,Pxx/sum(Pxx));

% Fourier Transform
vkf = fft(vk);
magvkf = abs(vkf(1:end/2)); phasevk = phase(vkf(1:end/2));

% Zero out the contributions (assumed to be) due to noise
magvkf2 = zeros(length(magvkf),1);
magvkf2(50:52) = magvkf(50:52);
magvkf2(125:127) = magvkf(125:127);
vkfmod = magvkf2.*exp(i*phvkf);
vkfmod2 = [vkfmod ; 0 ; flipud(conj(vkfmod(2:end)))];
```


MATLAB Script (cont.)

```
% Estimate the signal
vkhat = ifft(vkfmod2);

% Plot against the true deterministic signal
xk = sin(2*pi*0.1*kvec) + sin(2*pi*0.25*kvec);
figure; plot((0:99),xk(1:100),'r-',(0:99),vkhat(1:100),'b--')
```

Cross-spectral density

This is the frequency-domain counterpart of the cross-covariance function

$$\Phi_{yu}(f) = \sum_{l=-\infty}^{\infty} \sigma_{yu}[l] e^{-j2\pi f l l]} \tag{7}$$

- The CSD measures linear dependence between two (same) frequency components of two different series
- The cross density spectrum is complex
- The magnitude of the CSD gives the strength of common power, while the angle gives the phase lag between the two signals at that frequency

Note: Replacing the cross-covariance function by the auto-covariance function in (7) produces the auto-spectral density discussed previously

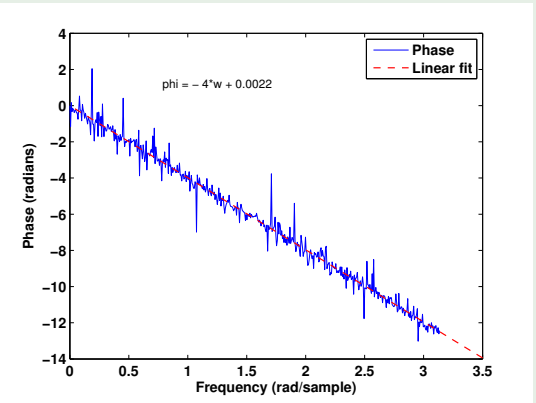
CPSD: Example

Delay estimation using phase

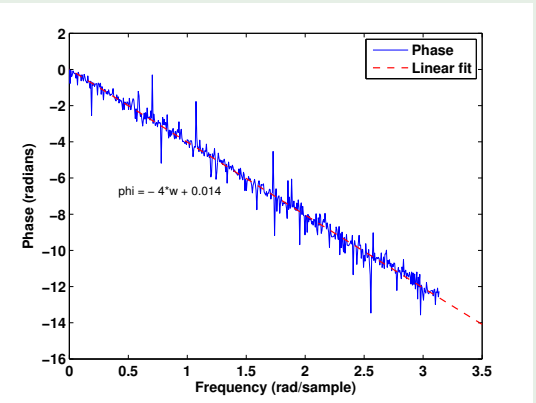
Recall the process $y[k] = Au[k-D] + v[k]$. The cross-spectral density is

$$\Phi_{yu}(\omega) = \sum_{l=-\infty}^{\infty} \sigma_{yu}[l] e^{-j\omega l} = \sum_{l=-\infty}^{\infty} A\sigma_{uu}[l-D] e^{-j\omega l} = Ae^{-j\omega D}\Phi_{uu}(\omega) \tag{8}$$

Thus, $|\gamma_{yu}(\omega)| = A\gamma_{uu}(\omega); \quad \phi_{yu}(\omega) = \angle\gamma_{yu}(\omega) = -D\omega$



(a) White-noise input



(b) Coloured noise input

(Ordinary) Coherence

Coherency is the normalized cross-spectral density

$$\eta_{yu}(f) = \frac{\Phi_{yu}(f)}{\sqrt{\Phi_{yy}(f)\Phi_{uu}(f)}}$$

- Magnitude of coherency is **coherence**
- For **linear time-invariant** systems, **coherence is unity at all frequencies**.

Coherence: Example

LTI system

Consider the process $G(q^{-1}) = \frac{2q^{-1}}{1-0.5q^{-1}}$;. A 1024-long input sequence $u[k]$ with filtered WN characteristics and a cut-off frequency at $\omega_0 = 1.2566$ rad/sec is used. Further, SNR $= \sigma_x^2/\sigma_e^2 \approx 16$. Squared coherence is computed from $N = 1024$ samples.

Theoretically, $|\kappa_{yu}(\omega)|^2 \approx 1$ at low frequencies, where SNR is very high and $|\kappa_{yu}(\omega)|^2 \approx 0$ at high frequencies, regions of very low SNR.

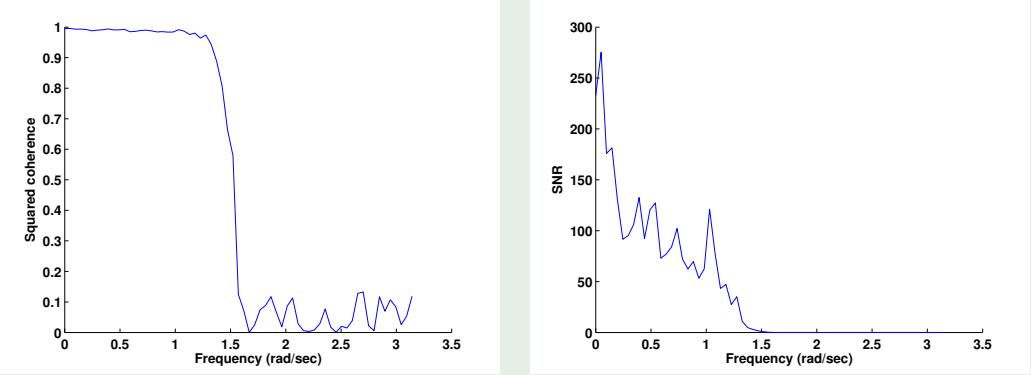


Figure: Squared input-output coherence and SNR for the process

Parameter Estimation

Fitting curve to data: Example

Let's now learn how to fit a mathematical function (curve) to a given dataset.

Data generation

Assume that the data is being generated by a sum of exponentials

$$y(t) = K - (2K/3)e^{-t/\tau_1} - (K/3)e^{-t/\tau_2} \tag{9}$$

where $\tau_1 = 4$, $\tau_2 = 10$ and $K = 6$.

We have observations of the above process at $t = kT_s$, $T_s = 0.5$ sec. Moreover, the sensor noise adds on to these observations, so that:

$$y_m[k] = K - (2K/3)e^{-k/\tau_1} - (K/3)e^{-k/\tau_2} + e[k] \tag{10}$$

where $e[k]$ is a Gaussian distributed random noise.

MATLAB Code for Data Generation

Data generation and visualization

```
% Time instant vector
kvec = linspace(0,99)';
% Generate true response
yk = 6 - 4*exp(-kvec./8) - 2*exp(-kvec./20);
% Add measurement noise
ym = yk + 0.1*randn(length(yk),1);

% Plot the resulting data
figure
plot(kvec,yk,'b-',kvec,ym,'rx','linewidth',2)
set(gcf,'Color',[1 1 1]);
set(gca,'fontsize',12,'fontweight','bold');
box off
xlabel('Sample Time','fontsize',12,'fontweight','bold')
ylabel('Response','fontsize',12,'fontweight','bold')
```


MATLAB Code for Curve Fitting

First, define the function that maps the response to time instant k. Assume for now that K is known.

Function for `lsqcurvefit`

```
function yx = respsys(x,xdata,c)
yx = c + x(1)*exp(-xdata/x(3)) + x(2)*exp(-xdata/x(4));
end
```

Next, write the script that passes this function to `lsqcurvefit`.

Script for fitting curve

```
c = 6;
xhat = lsqcurvefit(@(x,xdata) respsys(x,xdata,c),[1 1 1 2],kv
```

Try changing the options in `lsqcurvefit` to see if you can improve the estimates. Use the `optimoptions` command for this purpose.

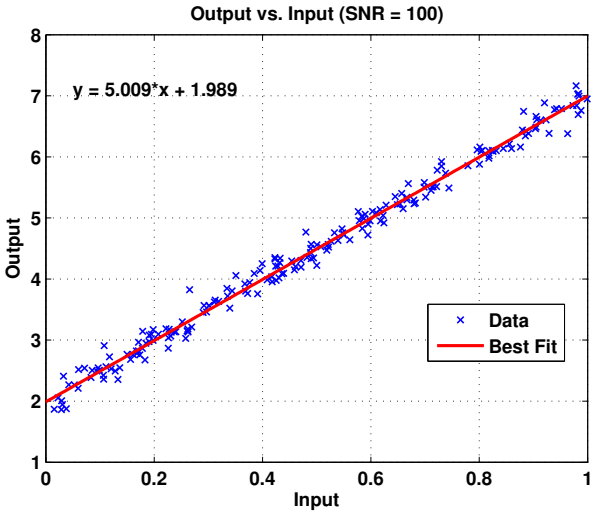
Example: Effect of SNR on parameter estimation

Process : $x[k] = b_1 u[k-1] + b_0; b_1 = 5; b_0 = 2$

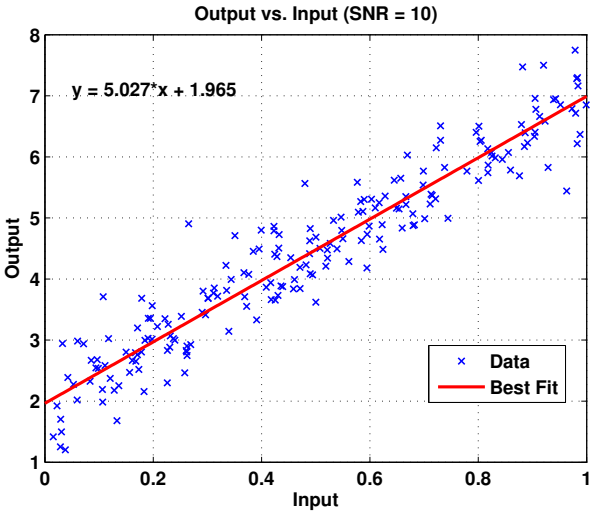
Only $y[k] = x[k] + v[k]$ (measurement) is available.

Goal: Estimate b_1, b_0 from $u[k], y[k]$ data.

Example: Effect of SNR . . . contd.



$\sigma_{\hat{b}_1} = 0.036, \sigma_{\hat{b}_0} = 0.02$



$\sigma_{\hat{b}_1} = 0.114, \sigma_{\hat{b}_0} = 0.064$

Decrease in SNR increases the error in parameter estimates ($\propto \sqrt{1/\text{SNR}}$)

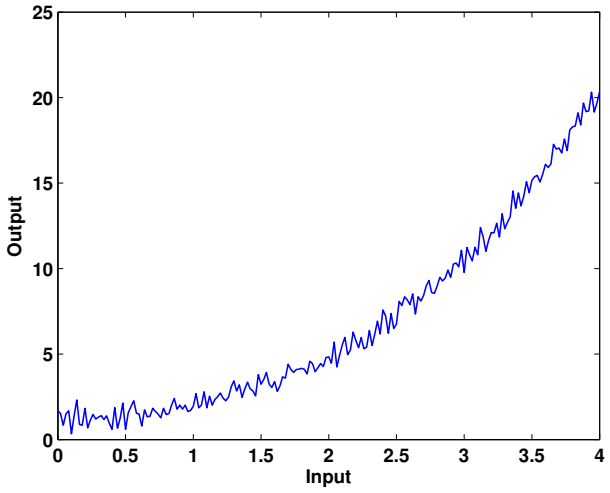
Example: Overfitting

Process : $x[k] = 1.2 + 0.4u[k] + 0.3u^2[k] + 0.2u^3[k]$

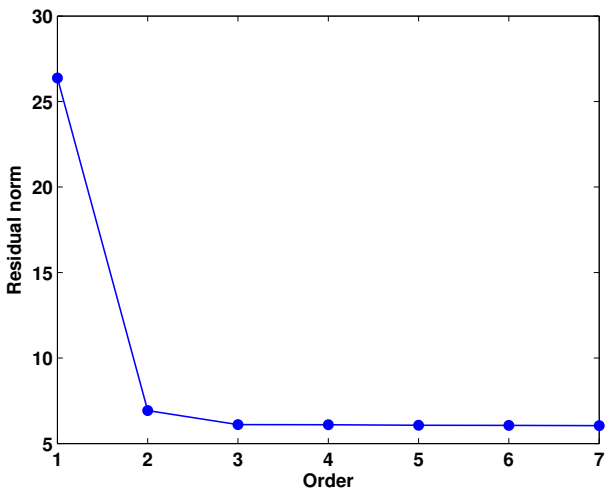
Only $y[k] = x[k] + v[k]$ (measurement) is available.

Goal: Fit a suitable polynomial model.

Example: Overfitting . . . contd.

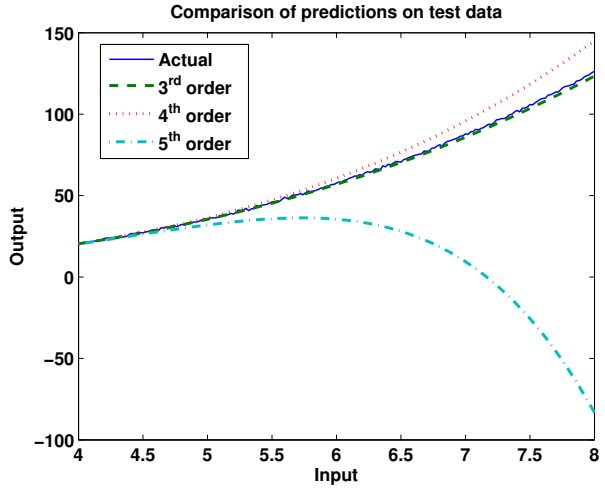


Input-output data



Cross-validation

Example . . . contd.

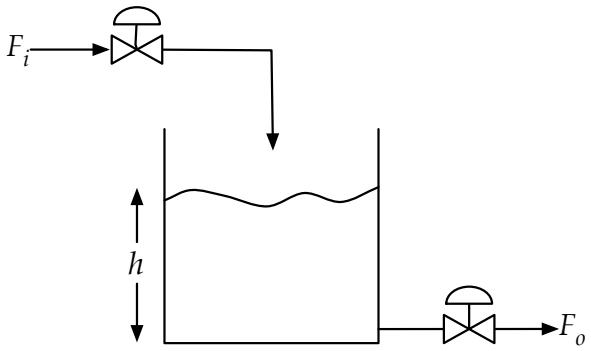


Overfitting occurs whenever the local chance variations are treated to be a part of the global characteristics

3^{rd} order fit: $\hat{y}[k] = \underset{(\pm 0.05)}{1.17} + \underset{(\pm 0.1)}{0.35}\, u[k] + \underset{(\pm 0.06)}{0.36}\, u^2[k] + \underset{(\pm 0.01)}{0.19}\, u^3[k]$

4^{th} order fit: $\hat{y}[k] = \underset{(\pm 0.04)}{1.23} + \underset{(\pm 0.13)}{0.14}\, u[k] + \underset{(\pm 0.6)}{0.14}\, u^2[k] + \underset{(\pm 0.054)}{0.085}\, u^3[k] + \underset{(\pm 0.00}{0.01}$

Empirical linearization of a liquid-level system



Objective: Build a model to explain the response in $h(t)$ for changes in $F_i(t)$

For our purposes, we shall adopt the **empirical approach** - perform an "experiment" wherein the inlet flow is excited and level readings are obtained. Subsequently, a model is built from the input-output data.

Data generating process

Process: $\dfrac{dh(t)}{dt} + \dfrac{1}{A_c} C_v \sqrt{h(t)} = \dfrac{1}{A_c} F_i(t)$

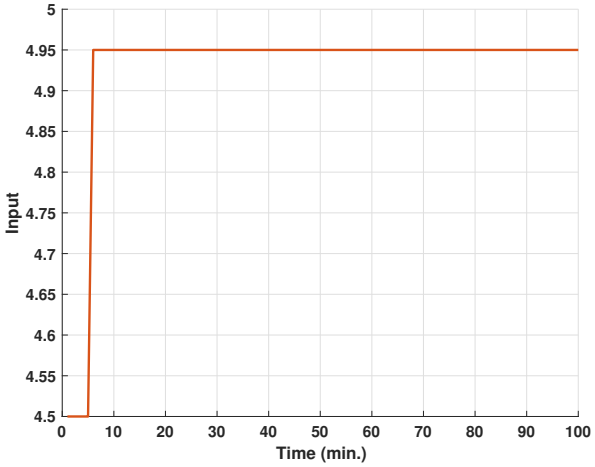
where A_c is the cross-sectional area, C_v is the valve coefficient

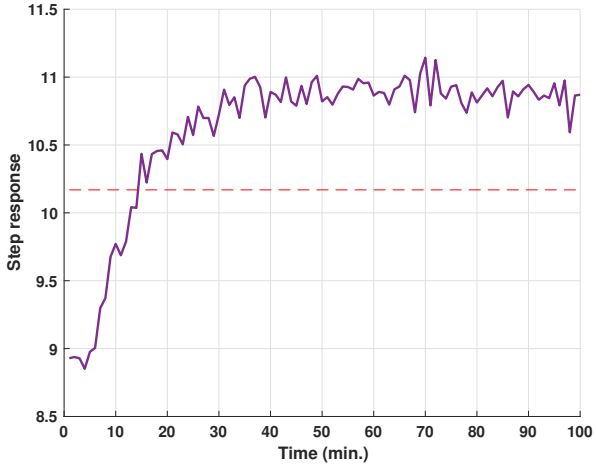
Parameter	A_c	C_v	h_0	T_s
Value	2 units	1.5 units	9 units	1 unit

Input: Flow variations are introduced as a pseudo-random binary sequence (PRBS), consisting of short and long duration pulses switching between two binary levels

Measurement noise: A random error (white-noise) with variance adjusted to obtain SNR 10.

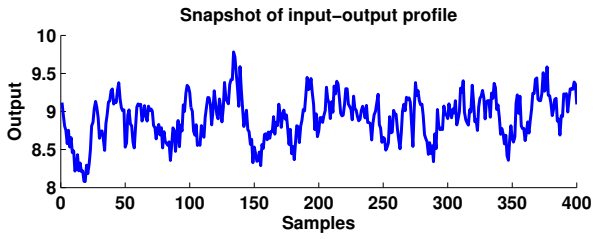
Preliminary experiment (step response)

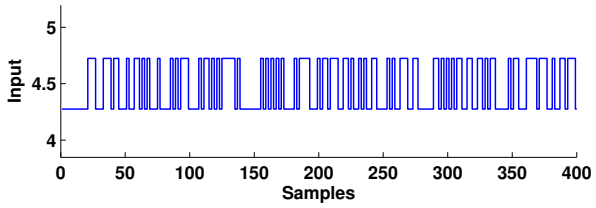




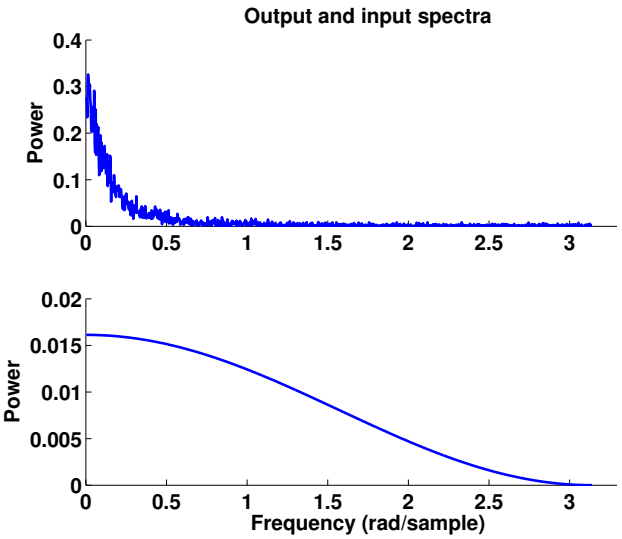
- Time constant of approximately 8 min. (first-order approximation).
- Choose sampling interval and input frequency content accordingly.

Visualizing the input-output data





Input-output data



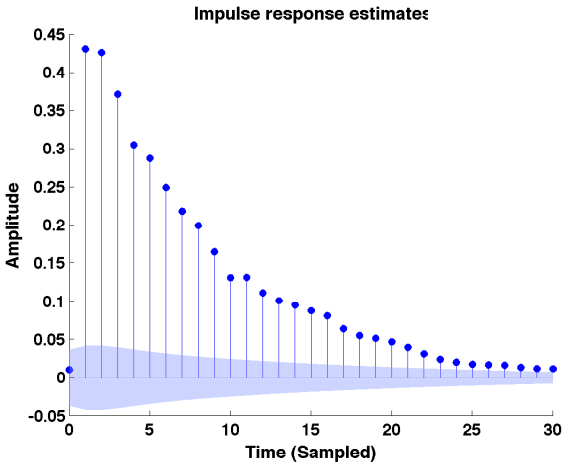
Input-output power spectra

Inferences

- No visible trends or non-stationarities
- Input contains primarily low frequencies
- Spectral plots suggest that the system is a low-pass filter

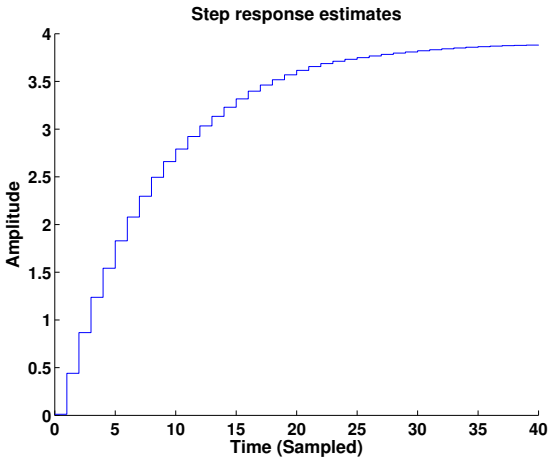
- Partition the data set into $N = 1500$ samples for training and remaining for testing.
- Work with deviation variables $y[k] = \tilde{y}[k] - \bar{y}[k]$, $u[k] = \tilde{u}[k] - \bar{u}[k]$

Non-parametric (response) model estimation



Estimated impulse response

$$y[k] = \sum_{l=0}^{M-1} g[l]u[k-l]$$



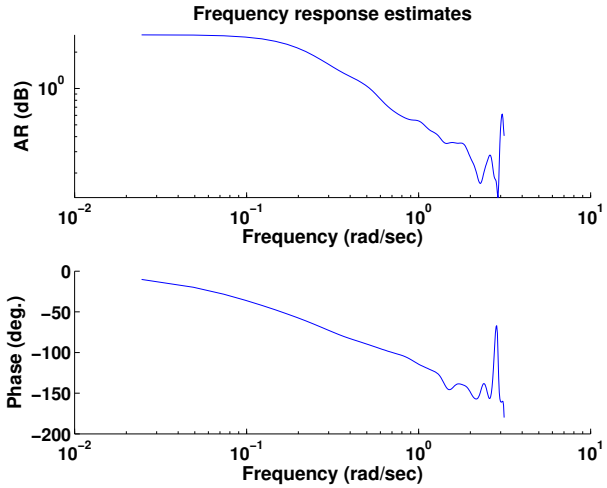
Estimated step response

$$y_s[k] = \sum_{n=0}^{k} g[n]$$

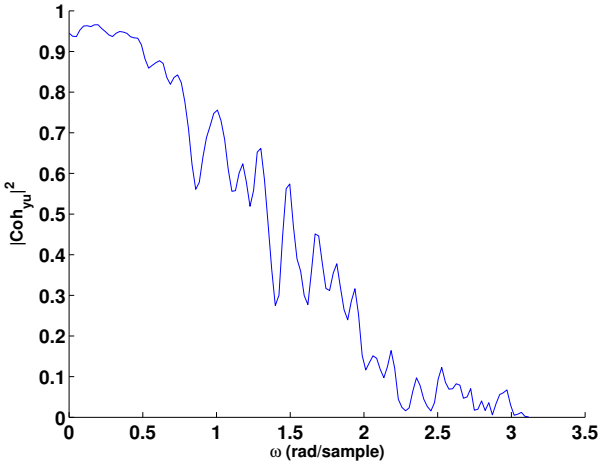
Inferences

- Stable system with a delay of one sample (due to ZOH)
- First-order (or higher-order overdamped) dynamics
- Gain of approximately 3.7 units. Time-constant of approx. 8 minutes.

Frequency domain analysis



Estimated frequency response



Estimated coherence

Inferences

- Low-pass filter characteristics
- Gain and time-constant estimates can also be obtained from FRF
- Coherence plot suggests that good fits can only be expected in the low frequency range

Identified (Empirical) Transfer Function Model

Model for the liquid-level system

Following a systematic procedure for identification, the model for the liquid-level system is

$$y[k] = \frac{\underset{(\pm 0.005)}{0.4621}\, q^{-1}}{1 - \underset{(\pm 0.002)}{0.8826}\, q^{-1}} u[k] + e[k] \tag{11a}$$

Comparison with the approximate discretized model

Compare the estimated model with the approximate, linearized, discretized model:

$$x[k] = -0.8825x[k-1] + 0.47u[k-1] \tag{12}$$

obtained at a sampling interval of $T_s = 1$ min.

Compare this with the estimated model

$$\hat{a}_1 = -0.8826(\pm 0.002),\ \hat{b}_1 = 0.4621(\pm 0.006)$$

With minimal assumptions and knowledge of the process, we are able to discover the underlying model with reasonable accuracy!

Useful links

- `http://www.mathworks.com`
  The official site of Mathworks, the makers of MATLAB.
- `https://www.tutorialspoint.com/matlab/`
  Tutorial on MATLAB - offers easy learning material.
- `https://www.math.utah.edu/~eyre/computing/matlab-intro/`
  MATLAB Tutorial at the Department of Mathematics, University of Utah.
- `http://www.matlabtips.com/learning-matlab/`
  Learning MATLAB, for new and advanced users.
- `https://matlabacademy.mathworks.com/R2016b/portal.html?course=gettingstarted`
  Free interactive tutorials by MathWorks