21 SYSTEM IDENTIFICATION

21.1 Introduction

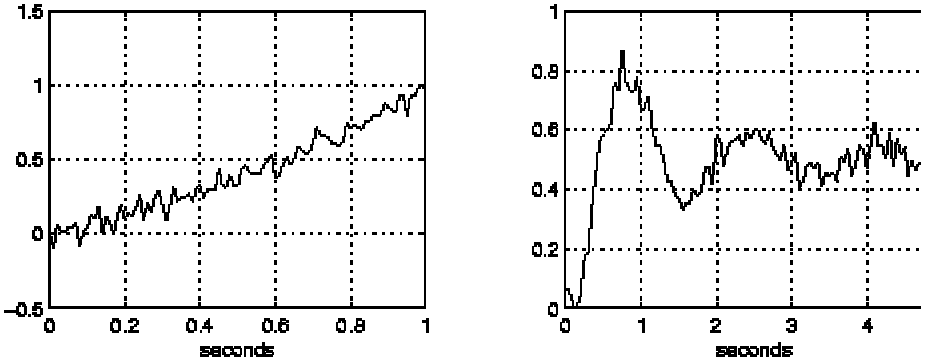
Model-based controller design techniques, such as the LQR and LTR, require a plant model. The process of generating workable models from observed data is the goal of system identification. Good controllers usually have some reasonable robustness guarantees, which motivates identification with simple methods. We discuss in this section four fundamental but useful techniques for approximate system identification of single-input, single-output plants. It should be noted that the area of system identification is a very rich one, and that the methods are only a small subset of what is available.

Except for the last approach, time-domain simulation, the methods are limited to linear models. If different inputs give different linear model coefficients, then it is likely that nonlinear terms are playing a role. The user then has the choice of ignoring the nonlinearity, for example if the operating point is controlled very closely, or developing a controller which takes specific account of the it. In any event, simulations with the nonlinear plant should always be performed to assess the robustness of the control strategy.

21.2 Visual Output from a Simple Input

For low-order plants which can tolerate impulse or step input, a great amount can be learned through step and impulse responses. The basic idea is to express what is observed as a time signal whose Laplace-domain equivalent can be recognized. As an example, consider the plant transfer function $P(s) = k/(\tau s + 1)$, a first-order lag with gain k, and time constant τ. We have $y(s) = k/s(\tau s + 1)$ for the step input $u(s) = 1/s$, and therefore $y(t) = k(1 - e^{-t/\tau})$. The gain k is evident as the maximum value taken by the measured output. The time constant τ is equal to the time required for $y(t)$ to reach the value $k(1 - e^{-1}) = 0.632k$. Similar estimates can be made for second-order systems, especially with the help of step functions parametrized on damping ratio ζ. Systems with order three or higher will usually be more difficult to assess with this visual technique.

Example: Two raw **step responses** in heading were recorded for two different vessels. The first vessel was strongly unstable and had to be powered down abruptly after one second; however, the smoothed data can still be fitted to the curve $y(t) = 0.58(e^t - 1)$. Give estimates of the two plant transfer functions $P(s)$, assuming that the step input was of magnitude one.



The time trace of the first system looks like the Laplace transform pair:

$$y(t) = \frac{1}{b-a}(e^{-at} - e^{-bt}) \leftrightarrow y(s) = \frac{1}{(s+a)(s+b)}$$

for the values $b = 0$ and $a = -1$. Thus for the experimental data,

$$y(s) = \frac{0.58}{s(s-1)}.$$

Since $y(s) = \hat{P}(s)u(s)$, where $u(s) = 1/s$, we have $\hat{P}(s) = 0.58/(s-1)$. The second trace looks like a second-order response of the form

$$\frac{y(s)}{u(s)} = \frac{k\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2},$$

where gain k, undamped frequency ω_n, and ζ are to be determined. First, we note that the steady value of $y(t)$ is about 0.5, so let $k = 0.5$. Next, with respect to the steady value, the first overshoot is about 0.3, and the second overshoot (same side) is about 0.1. The ratio is often written as the logarithmic decrement $\delta = ln(0.3/0.1) \simeq 1.10$, so that the damping ratio is simply $\zeta = \delta/2\pi \simeq 0.175$. Finally, the damped natural period is about 1.7s, leading to the damped natural frequency $\omega_d = 2\pi/1.7 \simeq 3.70 rad/s$. The undamped natural frequency ω_n is related to ω_d as follows:

$$\omega_n = \frac{\omega_d}{\sqrt{1-\zeta^2}} \simeq 3.75 rad/s.$$

Inserting these into the template above, we have

$$\hat{P}(s) = \frac{7.0}{s^2 + 1.28s + 14.1}.$$

21.3 Transfer Function Estimation - Sinusoidal Input

The main idea of transfer function estimation is that $P(s)$ can be estimated by simply dividing the measured output by the measured input, in the *frequency domain*:

$$\hat{P}(s) = \frac{y(s)}{u(s)}. \tag{216}$$

In applications, the input signal $u(s)$ is known quite accurately because it is generated by a computer. Two sources of error can corrupt the output signal $y(s)$, however: real disturbances and sensor noise. In some cases, disturbances may be band-limited (e.g., water waves), and if these occur in a frequency range far away from the dominant dynamics, the estimated transfer function approach will succeed. A similar argument holds for sensor noise, which in many devices is negligible for low frequencies. A sensor which has noise in the frequency range of the system dynamics is problematic for obvious reasons.

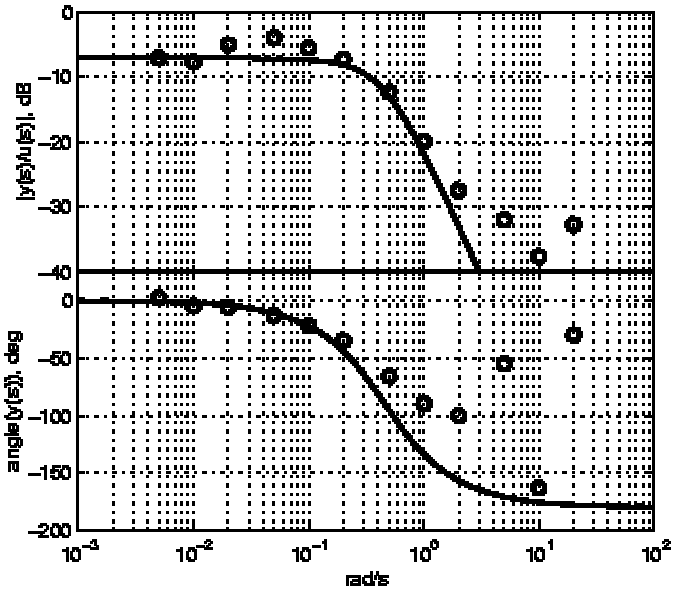
This section discusses the use of periodic inputs to create a Bode plot, while the next section generalizes to broadband input. Bode plots are figures of transfer function magnitude and phase as functions of frequency, which can be either parameterized in terms of poles, zeros, and a gain, or used directly in a loopshaping or Nyquist plot approach.

Certain plants which cannot admit a step or impulsive input will tolerate a sinusoidal input. The idea then is to drive the plant with a narrow-band, i.e., periodic, input signal $u(s)$. For each such test at a specific frequency, compute the magnitude and phase relating the input to the output. Conduct as many tests as are necessary to build a Bode plot of the plant estimate $\hat{P}(s)$.

Example: Sinusoidal rudder angles trajectories, of 10deg amplitude ($|u(t)| = 10 deg$), were implemented on a vessel operating near its cruising speed. The data are plotted, along with the magnitude and phase of the transfer function

$$\hat{P}(s) = \frac{0.093}{s^2 + 0.83s + 0.21},$$

which holds reasonably well at low frequencies. The phase angle of $y(s)$ is taken with respect to the input signal $u(s)$. Note that the experimental magnitude and phase in this example deteriorate at the higher frequencies. This is a property of almost all physical systems, and an indicator that the plant model cannot be trusted above a certain frequency range.



21.4 Transfer Function Estimation - Broadband Input

Many times one needs to deal with experimental data that is broadband, either as the result of a closed-loop run, or of significant disturbances. In this case, frequency-domain analysis can still be used, but elements of spectral analysis are necessary.

21.4.1 Fourier Transform of Sampled Data

For the purpose of analyzing transfer functions, the discrete Fourier transform (DFT) is a standard, optimized tool. It operates on a data vector $x(n)$ of length N:

$$X(m) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi mn/N}, \qquad (217)$$

for $m = [1, N]$. We review several points for dealing with the DFT calculation:

The m'th DFT point is a summation of complex unit-magnitude vectors ($e^{-j2\pi mn/N}$) times the original data ($x(n)$). Dividing the DFT result by N returns the signal to its real magnitude.
The DFT generates a vector of N complex points as outputs. These correspond to the frequency range

$$\omega_m = \frac{2\pi}{dt}\frac{m-1}{N}, \qquad (218)$$

i.e., the first frequency is 0, and the last frequency is slightly less than the sampling rate.

The sampling theorem limits our usable frequency range to only one-half of the sampling rate, called the Nyquist rate. The points higher than π/dt correspond to negative frequencies, and a complex-conjugacy holds: $X(2) = X^*(m), X(3) = X^*(m-1)$, and so on. What happens near the Nyquist rate depends on whether N is odd or even.

An additional multiplication of the DFT result by 2 will give peaks which are of about the right magnitude to be compared with the time-domain signal. With this scaling, the value $X(1)$ is *twice* the true DC value.

The Nyquist rate depends only on the sampling time step dt, but the frequency vector accompanying the DFT can be made arbitrarily long by increasing the number of data points. To improve the frequency resolution of the DFT, a common approach is to zero-pad the end of the real data,

The data $x(k)$ should be multiplied by a *smoothing window* $w(k)$, for two reasons. First, if the window goes to a small value near its tails, Gibb's effect (the spectral signature of a discontinuity between $x(1)$ and $x(N)$) can be minimized. The second rationale for windowing is based on the discrete nature of the transform. Because the DFT provides frequency data at only $N/2$ unique frequencies, there is the possibility that a component in the real data lies in between one of these DFT frequencies. The DFT magnitude at a specific $\omega(m)$ is in fact an *average* of continuous frequencies from the neighborhood of the point. Smoothing windows are generally chosen to achieve a tradeoff of two conflicting properties relating to the average: the width of the primary lobe (wide primary lobes bring in frequencies that belong to the adjacent bins $\omega(m-1)$ and $\omega(m+1)$), and the magnitude of the side lobes (tall side lobes bring in frequencies that are far away from $\omega(m)$).

There are many smoothing windows to choose from; no windowing at all is usually referred to as applying a rectangular window. One of the simplest is the Hann, or cosine, window:

$$w(k) = \frac{1}{2}\left[1 - \cos\left(\frac{2\pi k}{N}\right)\right]$$

The DFT of a given signal is subject to bias and variance. These can be reduced by taking separate DFT's of sub-sections of data (perhaps zero-padded to maintain frequency resolution), and then averaging them. It is common to use segments which do not overlap, but it is not a necessity. This averaging idea is especially important in transfer function verification; see below.

If possible, the DFT should be run on a number of samples N which is a multiple of a large power of two. If N is prime, the DFT will take a long time to execute.

The above steps, all of which are available through the Matlab function `spectrum()` for example, will ensure a fair spectral analysis of a time series.

21.4.2 Estimating the Transfer Function

In continuous time, when the plant input $y(t)$ and output $u(t)$ are transformed into frequency space, the estimated transfer function follows from

$$\hat{P}(s) = \frac{y(s)}{u(s)}. \tag{219}$$

As noted above, spectral analysis of a signal benefits by using multiple segments and averaging; we now want to include the same approach in our estimation of $P(s)$. The procedure is:

For the pth segment of data, compute the transfer function estimate

$$\hat{P}_p(m) = \frac{Y_p(m)Y_p^*(m)}{U_p(m)U_p^*(m)}.$$

For the pth segment of data, compute two covariances and a cross-covariance:

$$\begin{aligned}
\Gamma_{uu}^p(m) &= U_p(m)U_p^*(m) \\
\Gamma_{yy}^p(m) &= Y_p(m)Y_p^*(m) \\
\Gamma_{yu}^p(m) &= Y_p(m)U_p^*(m).
\end{aligned}$$

Construct average values of the transfer function estimate and the other quantities:

$$\begin{aligned}\hat{P}(m) &= avg_p(\hat{P}_p(m)) \\ \Gamma_{uu}(m) &= avg_p(\Gamma^p_{uu}(m)) \\ \Gamma_{yy}(m) &= avg_p(\Gamma^p_{yy}(m)) \\ \Gamma_{yu}(m) &= avg_p(\Gamma^p_{yu}(m))\end{aligned}$$

Compute the *coherence* function, which assesses the quality of the final estimate $\hat{P}(m)$:

$$Coh(m) = \frac{\Gamma_{yu}(m)\Gamma^*_{yu}(m)}{\Gamma_{yy}(m)\Gamma_{uu}(m)}.$$

If the coherence is near zero, then the segmental cross-covariances $\Gamma^p_{yu}(m)$ are sporadic and have cancelled out; there is no clear relation between the input and the output. This result could be caused by either disturbances or sensor noise, both of which are reasonably assumed to be random processes, and uncoupled to the input signal. Alternatively, if the coherence is near one, then the cross-covariances are in agreement and a real input-output relationship exists. With real data, the coherence will deteriorate at high frequencies and also at any frequency where disturbances or noise occur.

21.5 Time-Domain Simulation

The time-domain simulation approach tweaks the parameters of a simulation so that its output matches the observed output. The method has its main strength in the fact that it applies to *any* model that can be simulated, including those of high order and with significant nonlinearities. On the other hand, the method is computationally expensive and gives no guarantee of a useful solution, or even of convergence.

At the outset, we need to come up with some structure of the plant model. This can be based on physics in many cases. Consider, for example, the case of a mass mounted on a spring and a dashpot, driven by the input force $u(t)$. A fair guess for the *actual* dynamics has the form $my'' + by' + ky = u(t)$, and we plan to look through the three-dimensional parameter space $\vec{\theta} = [m, b, k]$.

The simulation operation can be written this way: $\hat{y}(t) = G(\vec{\theta}, u(t))$, and the system identification problem is to minimize $\|\hat{y}(t) - y_{obs}(t)\|$, say, where $\|\cdot\|$ here indicates the Euclidean norm. For a given parameter vector $\vec{\theta}$, running the simulation generates a new $\hat{y}$, and computing the norm gives a scalar measure of goodness.

Since the normed error is a complicated function of both $u(t)$ and $\vec{\theta}$, the minimization must proceed iteratively. The Nelder-Meade simplex method is easy to use, and can be invoked with the Matlab function `fmins`. As an example, the three programs listed comprise a working Matlab set for identification of a first-order, nonlinear system. Some notes on use:

In this example, the same simulation generates the ``observed'' data and the simulated response. Since the program `simulate` always uses the global variable `theta` as the parameters, we must be careful about setting `theta` in the calling programs.

After a simulation run is complete, the data is interpolated to the same time scale as the observed data, in order to compute the error.

The initial guess for `theta` is a random vector; the Simplex method will take over from this point. In many instances, however, `theta` is roughly known, and a better starting value can be given.

The Simplex method may head into invalid parameter space, e.g., negative mass. The error calculation, however, can be easily augmented by a term which penalizes invalid

parameters, e.g.,
`err = err + 1000*(1-sign(mass))`.

There is no guarantee that a global minimum will be found or even exists. Starting from different initial guesses for θ may help find better results, but we are still at the mercy of the minimization algorithm, and a very complicated function.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all ; clear global ;
global u_obs t_obs y_obs dt theta ;

dt = .3 ; % time step
theta = [1 2] ; % true parameter vector
t_obs = 0:dt:20*dt ; % observed time vector
u_obs = ones(length(t_obs),1) ; % observed input
[t_raw, y_raw] = ode45('simulate', [0 max(t_obs)], 0) ;
y_obs = spline(t_raw, y_raw, t_obs) ; % observed output

[theta_final] = fmins('get_err', randn(2,1)) ;

disp(sprintf('final theta(1):  %g.', theta_final(1))) ;
disp(sprintf('final theta(2):  %g.', theta_final(2))) ;
theta = theta_final ;
[t_raw, y_raw] = ode45('simulate', [0 max(t_obs)], 0) ;
y_sim = spline(t_raw, y_raw, t_obs) ;
figure(1) ; clf ; hold off ;
plot(t_obs, y_obs, t_obs, y_sim, t_obs, u_obs) ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [err] = get_err(theta_arg) ;
global y_obs t_obs theta ;

theta = theta_arg ;
[t_raw, y_raw] = ode45('simulate', [0 max(t_obs)], 0) ;
y_sim = spline(t_raw, y_raw, t_obs) ;
err = norm(y_sim - y_obs) ;
disp(sprintf('error:  %f.', err)) ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ydot] = simulate(t,y) ;

global u_obs dt theta ;
```

```
k = theta(1) ;
tau = theta(2) ;

ind = floor(t/dt) + 1 ; % we have to choose which u_obs to use:
                                % a zero-order hold as implemented.

ydot = ( k*u_obs(ind) - y^3 ) / tau ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```