Transfer-Function Identification

4.1: Introduction to transfer functions

- Issues with nonparametric system ID:
  - Needed to identify an infinite number of values to determine either a unit-pulse- or frequency response to uniquely identify system.
  - When there is disturbance, we need more data than values to identify, to somehow average out the effects of the disturbance.
  - So, prefer to find a low-order approximate model of system (*i.e.,* low number of parameters to identify).
- We now begin to look at parametric system ID—identifying a small set of parameters (here, in transfer-function form) to define a model.
  - Most control-system analysis and design methods work directly with either transfer-function or state-space models;
  - Can easily get unit-pulse and frequency responses from parametric models if needed (converse isn't generally true).
- Start by reviewing discrete-time transfer function models.
- Then, look at some common *model structures* used in system ID, then at *optimization methods*, then at *validation* of models.

Parametric models of linear systems

- In this section of notes, we will assume transfer-function (TF) models $G(q)$ and $H(q)$ for system and noise dynamics respectively,

$$y[k] = G(q)u[k] + H(q)e[k].$$

- The $u[k]$ signal is known (we applied it).
- The $e[k]$ signal is assumed to be a sequence of independent RVs,
  - Usually assume zero-mean, white, Gaussian.
  - May know $\mathbb{E}[e^2[k]]$, but may need to estimate that too.
- $G(q)$ is the transfer function between (known) input and output.
- $H(q)$ is the transfer function between disturbance and output.

- The transfer functions can be defined as[1]

$$G(q) = \sum_{k=1}^{\infty} g[k]q^{-k} \qquad \text{and} \qquad H(q) = \sum_{k=0}^{\infty} h[k]q^{-k}.$$

- If the unit-pulse responses are known, we can compute $G(q)$ and $H(q)$ directly. But, we're trying to avoid this approach...
- Fortunately, many systems can be very well described by (short) discrete-time difference equations. For example,

$$y[k] = \alpha y[k-1] + \beta u[k-1].$$

- If we write this equation in operator form, we get[2]

$$(1 - \alpha q^{-1})y[k] = \beta q^{-1}u[k]$$

$$y[k] = \frac{\beta q^{-1}}{(1 - \alpha q^{-1})}u[k] = \underbrace{\frac{\beta}{(q - \alpha)}}_{G(q)}u[k].$$

[1] Note the different summation starting indices, due to assumption that $G(q)$ has no strictly feedthrough path (*i.e.*, $g[0] = 0$). If $g[0] \neq 0$, then both indices start at $k = 0$.

[2] The second line of this equation does not really follow from the first, since q^{-1} is an operator and not a variable. However, we can use a similar approach with z transforms to rigorously come up with the same result, if we permit rational-polynomial functions of q. This is really an abuse of notation, but when everybody understands what you're talking about, I suppose it's okay.

- From this example, we see that a system having infinite length unit-pulse response can be perfectly described by a transfer function having only two unknown parameters, α and β.
- All systems described by linear constant coefficient difference equations can be represented with transfer functions that are "rational polynomial in q". For example,

$$G(q) = \frac{B(q)}{A(q)} = q^{-n_k} \frac{b_1 q^{-1} + \cdots + b_{n_b} q^{-n_b}}{1 + a_1 q^{-1} + \cdots + a_{n_a} q^{-n_a}}.$$

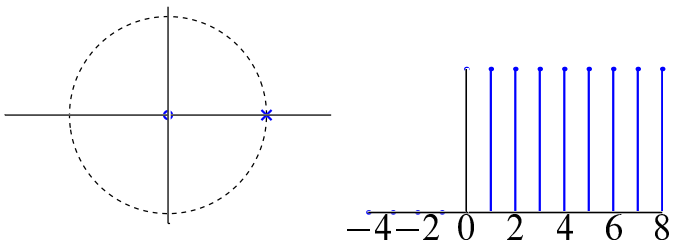
  - Here, the system has a transport delay of n_k samples, n_b feedforward coefficients, and n_a feedback coefficients.
  - So, there are a total of $n_a + n_b + 3$ values to determine, in order to define the transfer function ($3 = \dim\{n_{a,}, n_b, n_k\}$).
  - This is much more compact than trying to find the entire unit-pulse or frequency responses.
- By analogy with the z-domain transfer function, we can treat q as a variable and factor the numerator and denominator polynomials.
  - Roots of the numerator polynomial are called zeros of the transfer function;
  - Roots of the denominator polynomial are called poles of the transfer function.
- System response to initial conditions or input stimuli are *qualitatively* defined by pole locations. The zero locations are required to *quantify* the result.
  - Poles tell if the system is stable or unstable, smooth or oscillatory, fast or slow.

4.2: Some examples of time responses versus pole locations

- The following shows some correspondence between the q-plane and some discrete-time unit-pulse-response signals.

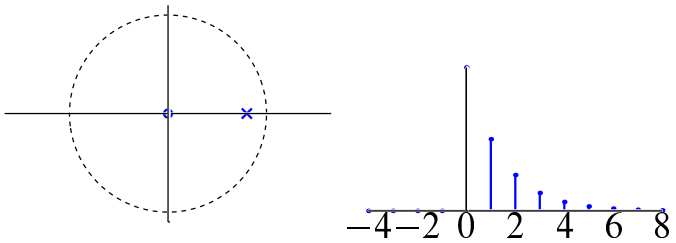
Unit step:

- $g[k] = 1[k]$.
- $G(q) = \dfrac{q}{q-1}, \quad |q| > 1.$



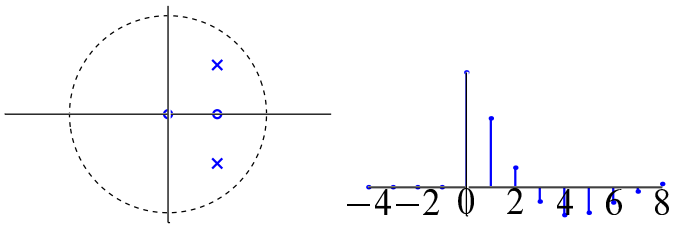
Exponential (geometric):

- $g[k] = a^k 1[k], \quad |a| < 1.$
- $G(q) = \dfrac{q}{q-a}, \quad |q| > |a|.$



General cosinusoid:

- $g[k] = a^k \cos[\omega k] 1[k], \ |a| < 1.$
- $G(q) = \dfrac{q(q - a\cos\omega)}{q^2 - 2a(\cos\omega)q + a^2}$ for $|q| > |a|$.



- The radius to the two poles is a; the angle to the poles is ω.
- The zero (not at the origin) has the same real part as the two poles.
  - If $\omega = 0$, $G(q) = \dfrac{q}{q-a}$ . . . geometric!
  - If $\omega = 0$, $a = 1$, $G(q) = \dfrac{q}{q-1}$ . . . step!
- Pole *radius* a is the geometric factor, determines settling time.

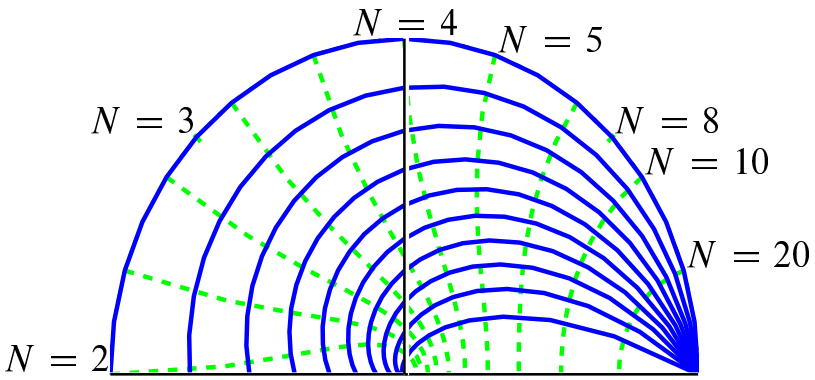
1. $|a| = 0$, finite-duration response. *e.g.*, $\delta[k-N] \Longleftrightarrow q^{-N}$.
2. $|a| > 1$, growing signal which will not decay.
3. $|a| = 1$, signal with constant amplitude; either step or cosine.

4. $|a| < 1$, decaying signal. Small a = fast decay (see below).

a	0.9	0.8	0.6	0.4
$\approx$ duration N	43	21	9	5

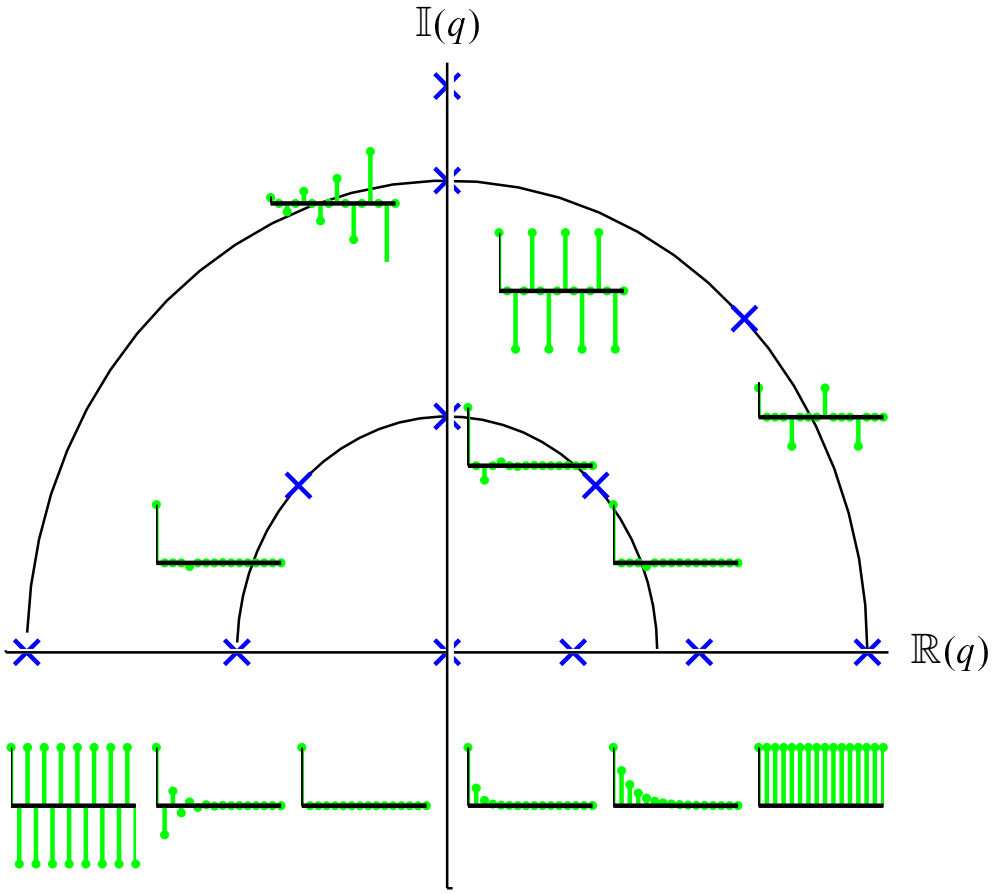
- Pole *angle* ω determines number of samples per oscillation.
  - That is, if we require $\cos[\omega k] = \cos[\omega(k+N)]$, then

$$N = \left.\frac{2\pi}{\omega}\right|_{\text{rad}} = \left.\frac{360}{\omega}\right|_{\text{deg}}.$$



- Solid: cst. damping ratio ζ.
- Dashed: constant natural frequency ω_n.

- Plot to right shows discrete-time unit-pulse responses versus pole locations.



Correspondence with continuous-time signals

- Let $g(t) = e^{-at}\cos(bt)1(t)$.
- Suppose

$$\left.\begin{array}{rcl} a &=& 0.3567/T \\ b &=& \dfrac{\pi/4}{T} \end{array}\right\} \quad T = \text{sampling period.}$$

- Then,

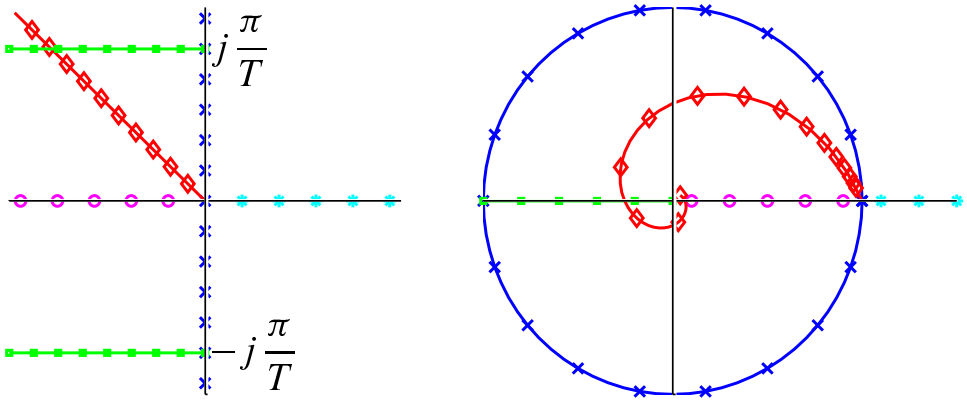
$$g[k] = g(kT) = \left(e^{-0.3567}\right)^k \cos\left(\frac{\pi k}{4}\right) 1[k]$$

$$= 0.7^k \cos\left(\frac{\pi k}{4}\right) 1[k].$$

  (This is the cosinusoid example used in the earlier example).

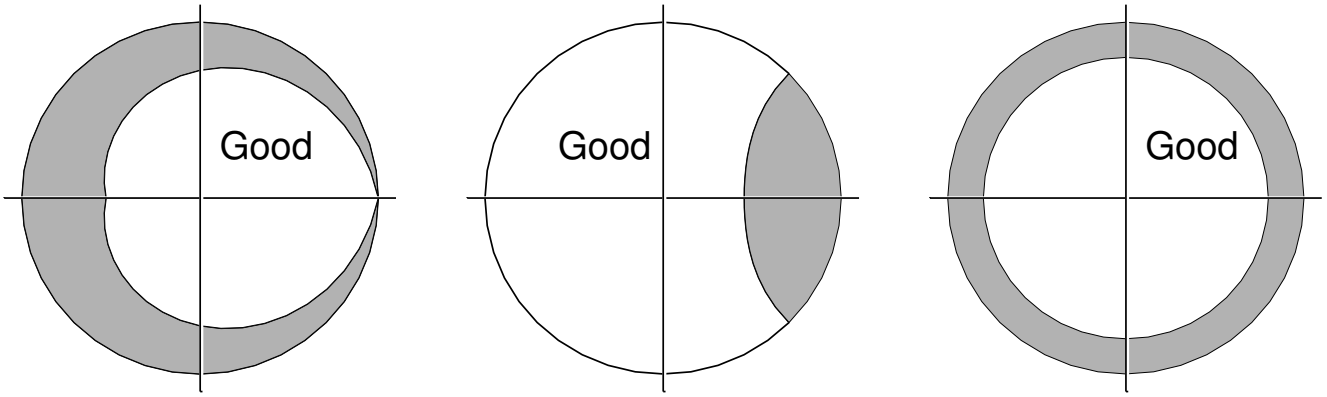
- $G(s)$ has poles at $s_{1,2} = -a + jb$ and $-a - jb$.
- $G(q)$ has poles at radius e^{-aT} angle $\omega = \pm bT$ or at $e^{-aT \pm jbT}$.
  - So, $q_{1,2} = e^{s_1 T}$ and $e^{s_2 T}$.
- In general, *poles* convert between the s-plane and q-plane via $q = e^{sT}$.

EXAMPLE: Some corresponding pole locations:



s-plane q-plane

- $j\omega$-axis maps to unit circle.
- Constant damping ratio ζ maps to strange spiral.
- When considering system response to a step input for controls purposes, the following diagrams may be helpful:

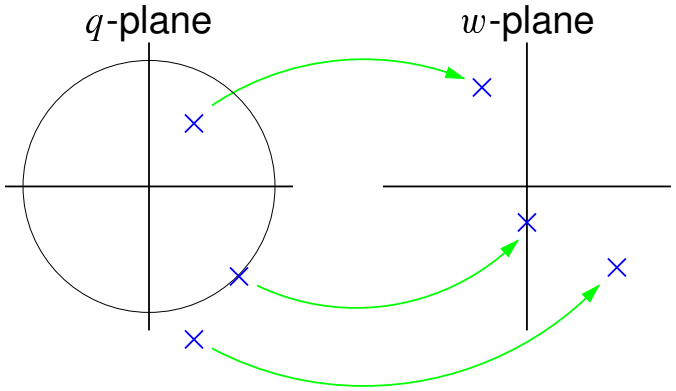


Damping ζ Frequency ω_n Settling time

- Higher-order systems:
  - Pole moving toward $q = 1$, system slows down.
  - Zero moving toward $q = 1$, overshoot.
  - Pole and zero moving close to each other cancel.

4.3: Bode plots from discrete-time transfer functions

- Knowing a system's frequency response is key to many system analysis and control synthesis methods.
- Bode plots are plots of frequency response of a system, displayed as separate magnitude- and phase-response plots.
- Frequency response is extracted from transfer functions, knowing geometrically where pure sinusoids exist in s- or q-plane.
  - In s-plane, $H(s)|_{s=j\omega}$ is frequency response for $0 \leq \omega < \infty$.
  - In q-plane, $H(q)|_{q=e^{j\omega T}}$ is frequency response for $0 \leq \omega \leq \omega_s/2$.
- Straight-line tools of s-plane analysis *DON'T WORK!* They are based on geometry and geometry has changed—$j\omega$-axis to q-unit circle.
- To use straight-line tools, must convert $H(q)$ to an equivalent form $H(w)$ where unit circle in q-plane maps to $j\omega$-axis in w-plane.
  - Ideally, interior of q-plane unit circle maps to LHP in w-plane, and exterior of q-plane unit circle maps to RHP in w-plane.
  - It is not accurate to label the destination plane the s-plane. It is often called the w-plane, and the transformation between the q-plane and the w-plane is called *the w-Transform*.
    - That is, for the $H(w)$ we come up with, $H(w) \neq H(s)|_{s=w}$.



- A transform that satisfies these requirements is the bilinear transform:

$$H(w) = H(q)\big|_{q=\frac{1+(T/2)w}{1-(T/2)w}} \qquad \text{and} \qquad H(q) = H(w)\big|_{w=\frac{2}{T}\frac{q-1}{q+1}}.$$

- Three things to check:

  1. Unit circle in q-plane $\mapsto$ $j\omega$-axis in w-plane.
  2. Inside unit circle in q-plane $\mapsto$ LHP in w-plane.
  3. Outside unit circle in q-plane $\mapsto$ RHP in w-plane.

- If true,

  1. Take $H(q) \mapsto H(w)$ via the bilinear transform.
  2. Use straight-line methods to plot Bode plot of $H(w)$.

CHECK: Let $q = re^{j\omega T}$. Then, q is on the unit circle if $r = 1$, q is inside the unit circle if $|r| < 1$ and q is outside the unit circle if $|r| > 1$.

$$q = re^{j\omega T}$$

$$w = \frac{2}{T}\frac{q-1}{q+1}\bigg|_{q=re^{j\omega T}} = \frac{2}{T}\frac{re^{j\omega T}-1}{re^{j\omega T}+1}.$$

- Expand $e^{j\omega T} = \cos(\omega T) + j\,\sin(\omega T)$ and use the shorthand $c \triangleq \cos(\omega T)$ and $s \triangleq \sin(\omega T)$. Also note that $s^2 + c^2 = 1$.

$$w = \frac{2}{T}\left[\frac{rc + jrs - 1}{rc + jrs + 1}\right] = \frac{2}{T}\left[\frac{(rc-1)+jrs}{(rc+1)+jrs}\right]\left[\frac{(rc+1)-jrs}{(rc+1)-jrs}\right]$$

$$= \frac{2}{T}\left[\frac{(r^2c^2-1)+j(rs)(rc+1)-j(rs)(rc-1)+r^2s^2}{(rc+1)^2+(rs)^2}\right]$$

$$= \frac{2}{T}\left[\frac{r^2-1}{r^2+2rc+1}\right] + j\frac{2}{T}\left[\frac{2rs}{r^2+2rc+1}\right].$$

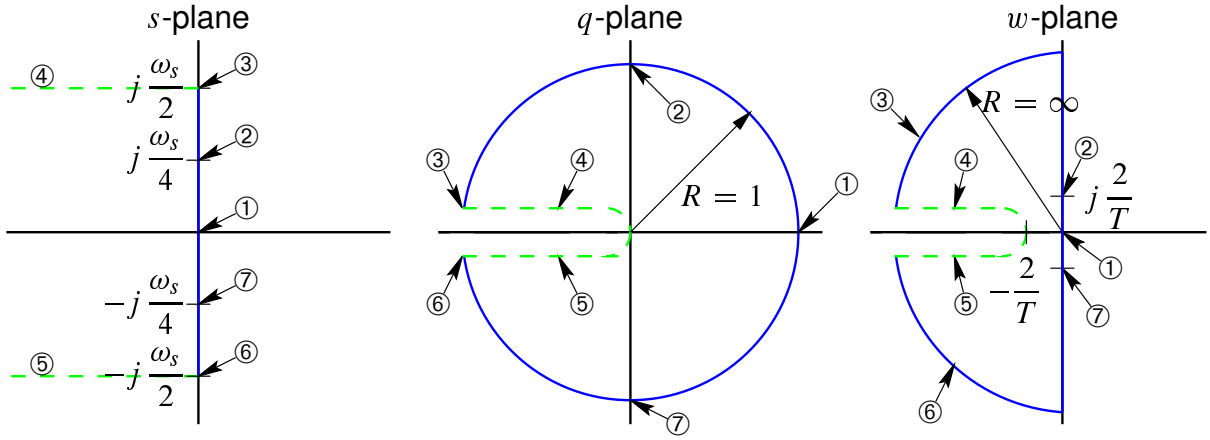
- Notice that the real part of w is 0 when $r = 1$ (w is on the imaginary axis), the real part of w is negative when $|r| < 1$ (w in LHP), and that

the real part of w is positive when $|r| > 1$ (w in RHP). Therefore, the bilinear transformation does exactly what we want.

- When $r = 1$,

$$w = j\frac{2}{T}\frac{2\sin(\omega T)}{2 + 2\cos(\omega T)} = j\frac{2}{T}\tan\left(\frac{\omega T}{2}\right).$$

- That is, in w-plane, $H(w)|_{w=j\omega_w}$ is the frequency response for $0 \le \omega_w < \infty$. Straight-line tools work, but frequency axis is warped!
- The following diagram summarizes the relationship between the s-plane, q-plane, and w-plane:



PROCEDURE:

1. Convert $H(q)$ to $H(w)$ by $H(w) = H(q)|_{q=\frac{1+(T/2)w}{1-(T/2)w}}$.
2. Simplify expression to rational-polynomial in w.
3. Factor into zeros and poles in standard "Bode Form".
4. Plot the response exactly the same way as an s-plane Bode plot. Note: Plots are versus $\log_{10}\omega_w$ ... $\omega_w = \frac{2}{T}\tan\left(\frac{\omega T}{2}\right)$. Can re-scale axis in terms if ω if we want.

EXAMPLE: Plot the straight-line w-plane Bode plot for a system with transfer function:

$$\text{Let } G(q) = \frac{0.368q + 0.264}{q^2 - 1.368q + 0.368}.$$

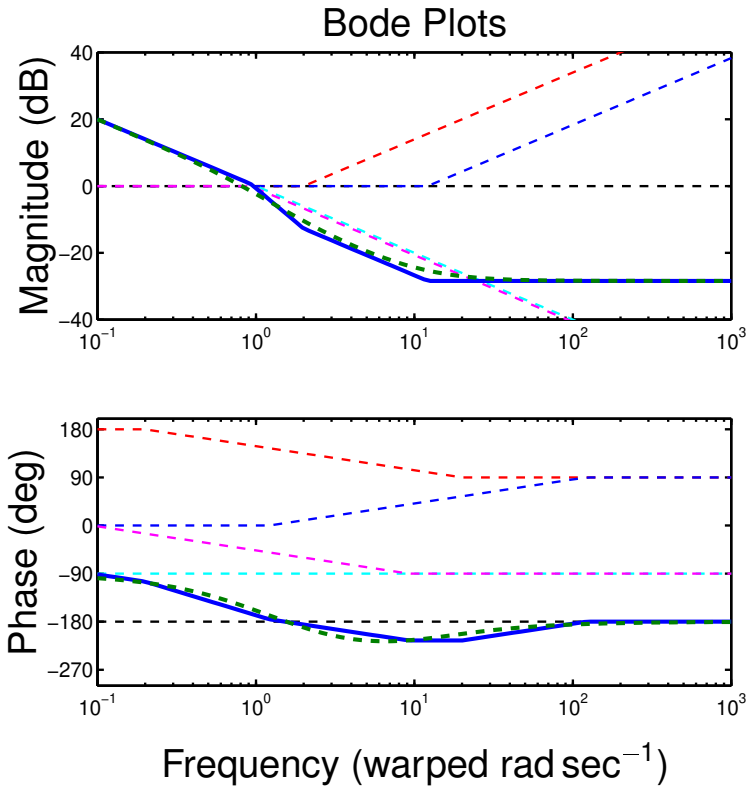
(1,2)

$$\begin{aligned} G(w) &= \frac{0.368\left[\frac{1+0.5w}{1-0.5w}\right] + 0.264}{\left[\frac{1+0.5w}{1-0.5w}\right]^2 - 1.368\left[\frac{1+0.5w}{1-0.5w}\right] + 0.368} \\ &= \frac{0.368(1+0.5w)(1-0.5w) + 0.264(1-0.5w)^2}{(1+0.5w)^2 - 1.368(1+0.5w)(1-0.5w) + 0.368(1-0.5w)^2} \\ &= \frac{-0.0381(w-2)(w+12.14)}{w(w+0.924)}. \end{aligned}$$

(3)

$$G(j\omega_w) = \frac{-\left(j\frac{\omega_w}{2} - 1\right)\left(j\frac{\omega_w}{12.14} + 1\right)}{j\omega_w\left(j\frac{\omega_w}{0.924} + 1\right)}.$$

(4)



4.4: System ID with transfer-function models

- Having reviewed discrete-time systems, we return to the problem of system ID using transfer-function models.
- Recall that we are assuming a system model with system and noise dynamics specified by $G(q)$ and $H(q)$,

$$y[k] = G(q)u[k] + H(q)e[k],$$

with $G(q) = B(q)/A(q)$ and $H(q) = C(q)/D(q)$.

ISSUES: Do all of $A(q)$, $B(q)$, $C(q)$, $D(q)$ exist? And, what are the values for n_k, n_a, n_b, n_c, and n_d?

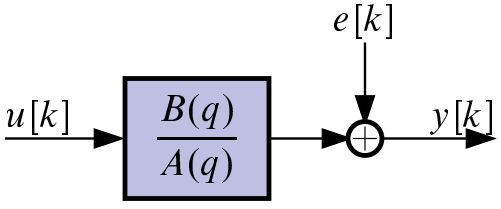
- Typically use time response to estimate delay n_k directly.
- Then, try certain "standard" structures involving $A(q)$, $B(q)$, etc., to see which fits "best".
- Involved in this is the selection of model order via some criteria, and validation of the model.

Standard model forms

- Different approaches to using transfer-function models differ primarily on how disturbance is factored into the system response.
- We look at four "standard" model forms that have different properties: some are easier to identify, but others are more general.

Output error model

- The output error (OE) approach models the system as shown in the diagram.



- That is, $G(q,\theta) = \dfrac{B(q,\theta)}{A(q,\theta)}$ and $H(q,\theta) = 1$, and

$$y[k] = \frac{B(q,\theta)}{A(q,\theta)}u[k] + e[k].$$

- The OE model is parametrized by $\theta = \begin{bmatrix} a_1 & \cdots & a_{n_a} & b_1 & \cdots & b_{n_b} \end{bmatrix}$.
- The filters are defined as

$$A(q,\theta) = 1 + a_1 q^{-1} + \cdots + a_{n_a} q^{-n_a}$$

$$B(q,\theta) = b_1 q^{-1} + \cdots + b_{n_b} q^{-n_b}.$$

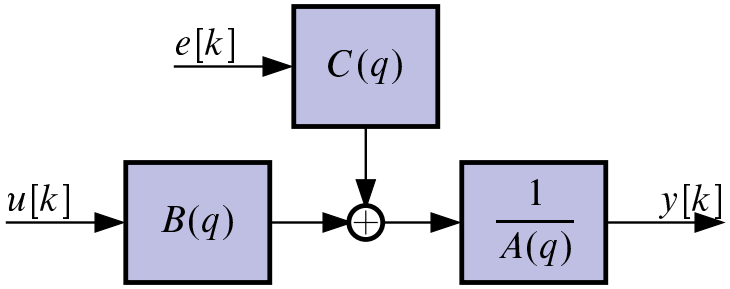
- The noise source is the difference (error) between the actual and noise-free output.
- Good to use when *system dominated by white sensor noise.*
- Expect problems when the noise spectrum is shaped (colored noise, process noise). Why?
- Denote the noise-free output by $w[k]$. Then, the difference equation is

$$w[k] + a_1 w[k-1] + \cdots + a_{n_a} w[k-n_a] = b_1 u[k-1] + \cdots + b_{n_b} u[k-n_b]$$

$$y[k] = w[k] + e[k].$$

ARMAX model

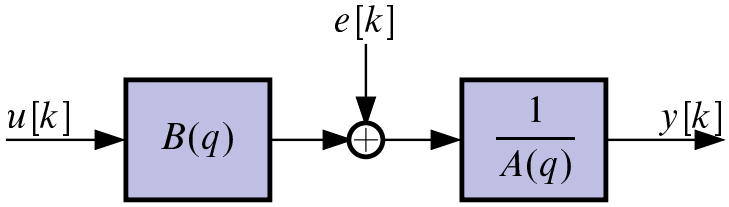
- The ARMAX (auto regressive with moving average and exogenous (or extra) input) approach models the system as shown in the diagram.



- That is, $G(q,\theta) = \dfrac{B(q,\theta)}{A(q,\theta)}$ and $H(q,\theta) = \dfrac{C(q,\theta)}{A(q,\theta)}$, and
$$A(q,\theta)y[k] = B(q,\theta)u[k] + C(q,\theta)e[k].$$
- It is parametrized by $\theta = \begin{bmatrix} a_1 & \cdots & a_{n_a} & b_1 & \cdots & b_{n_b} & c_1 & \cdots & c_{n_c} \end{bmatrix}$.
- The filters are defined as
$$A(q,\theta) = 1 + a_1 q^{-1} + \cdots + a_{n_a} q^{-n_a}$$
$$B(q,\theta) = b_1 q^{-1} + \cdots + b_{n_b} q^{-n_b}$$
$$C(q,\theta) = 1 + c_1 q^{-1} + \cdots + c_{n_c} q^{-n_c}.$$
- Disturbance and input subject to the same poles.
- Good model if shaped or process noise dominates.
- The difference equation is
$$\begin{aligned} y[k] + a_1 y[k-1] + \cdots + a_{n_a} y[k-n_a] \\ = b_1 u[k-1] + \cdots + b_{n_b} u[k-n_b] \\ + e[k] + c_1 e[k-1] + \cdots + c_{n_c} e[k-n_c]. \end{aligned}$$

ARX model

- The ARX (auto regressive with exogenous (or extra) input) approach models the system as shown in the diagram. That is,
$A(q,\theta)y[k] = B(q,\theta)u[k] + e[k]$,
or, $G(q,\theta) = \dfrac{B(q,\theta)}{A(q,\theta)}$ and
$H(q,\theta) = \dfrac{1}{A(q,\theta)}$.



- The ARX model is parametrized by $\theta = \begin{bmatrix} a_1 & \cdots & a_{n_a} & b_1 & \cdots & b_{n_b} \end{bmatrix}$.
- The filters are defined as

$$A(q,\theta) = 1 + a_1 q^{-1} + \cdots + a_{n_a} q^{-n_a}$$

$$B(q,\theta) = b_1 q^{-1} + \cdots + b_{n_b} q^{-n_b}.$$

- Simplified disturbance model. Not particularly well motivated by any physical intuition, but solution found by *very simple numerical method.*
- The difference equation is

$$y[k] + a_1 y[k-1] + \cdots + a_{n_a} y[k-n_a] = b_1 u[k-1] + \cdots + b_{n_b} u[k-n_b] + e[k].$$

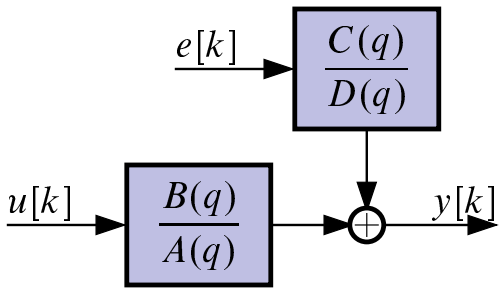
Box–Jenkins model

- The Box–Jenkins (BJ) approach models the system as shown in the diagram. That is,

$$y[k] = \frac{B(q,\theta)}{A(q,\theta)} u[k] + \frac{C(q,\theta)}{D(q,\theta)} e[k],$$

or $G(q,\theta) = \dfrac{B(q,\theta)}{A(q,\theta)}$ and

$$H(q,\theta) = \frac{C(q,\theta)}{D(q,\theta)}.$$



- The Box–Jenkins model is parametrized by

$$\theta = \begin{bmatrix} a_1 & \cdots & a_{n_a} & b_1 & \cdots & b_{n_b} & c_1 & \cdots & c_{n_c} & d_1 & \cdots & d_{n_d} \end{bmatrix}.$$

- The filters are defined as

$$A(q,\theta) = 1 + a_1 q^{-1} + \cdots + a_{n_a} q^{-n_a}$$

$$B(q,\theta) = b_1 q^{-1} + \cdots + b_{n_b} q^{-n_b}$$

$$C(q,\theta) = 1 + c_1 q^{-1} + \cdots + c_{n_c} q^{-n_c}$$

$$D(q,\theta) = 1 + d_1 q^{-1} + \cdots + d_{n_d} q^{-n_d}.$$

- Very general form. Includes all others as special cases.
- Denote the noise-free system output by $w[k]$, and the overall disturbance by $v[k]$. Then, the difference equation is

$$w[k] + a_1 w[k-1] + \cdots + a_{n_a} w[k-n_a] = b_1 u[k-1] + \cdots + b_{n_b} u[k-n_b]$$

$$v[k] + d_1 v[k-1] + \cdots + d_{n_d} v[k-n_a] = e[k] + c_1 e[k-1] + \cdots + c_{n_c} e[k-n_c]$$

$$y[k] = w[k] + v[k].$$

Generalizing for longer input delay

- All these difference equations assume a single delay in $B(q,\theta)$ only.
- Often need n_k additional delays: $B(q,\theta) = \boldsymbol{q}^{\Gamma \boldsymbol{n_k}} (b_1 q^{-1} + \cdots + b_{n_b} q^{-n_b})$.
- Difference equations now of the form
  $\cdots = b_1 u[k \Gamma \boldsymbol{n_k} - 1] + b_2 u[k \Gamma \boldsymbol{n_k} - 2] + \cdots$.

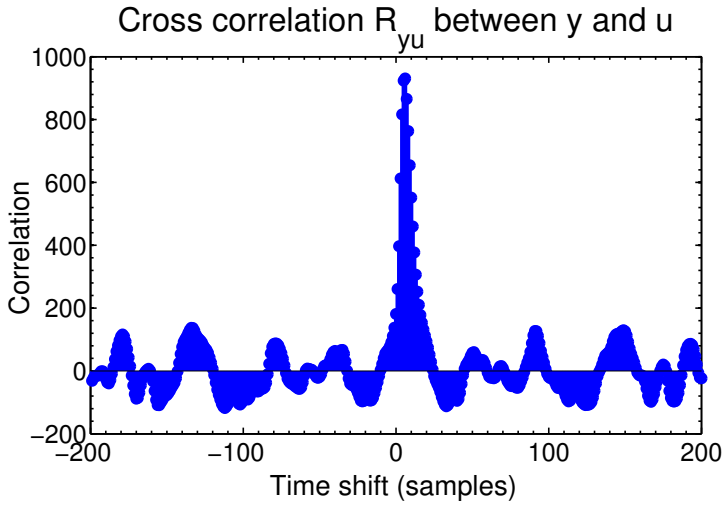
4.5: Initial model structure selection

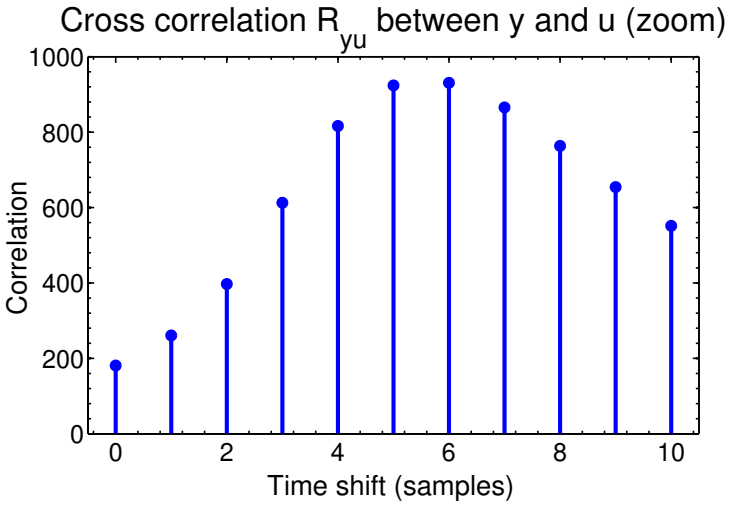
- Models are valid for different assumptions on the dynamics, noise.
- Often not clear which is the best to use!
  - Common approach is to try several structures and see if the fit improves (will look at how to compare them later on).
- In addition to the structures themselves, number of delays n_k and coefficient order n_a, n_b, n_c, and n_d not obvious either.

Delay estimation

- One quick way to estimate the delay is to plot the cross-correlation between the output and input. For the “dryer” data,

```
load dryer2
u2 = u2 - mean(u2); y2 = y2 - mean(y2);
x = (-length(u2)+1):(length(u2)-1);
Ryu = xcorr(y2,u2); stem(x,Ryu,'filled');
```





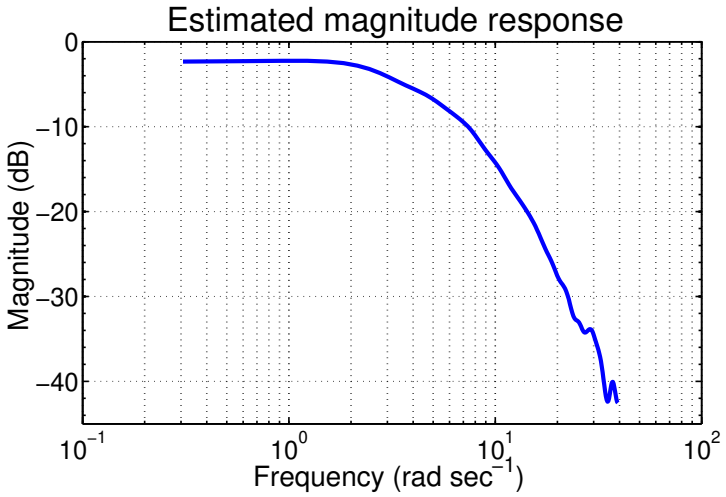
- We see a general noise floor of around 200. So, a time delay of two or three samples looks about right. Should probably try both.
- A second method is to look at unit-pulse- or step-response data (if available), looking for the first non-negligible response point.

- A third method is to use MATLAB's `delayest.m`. (cf. Topic 4.14)
  - Fits ARX models with different delays to the data, uses a "cost" criteria to determine which one looks most likely.
  - Since ARX is not guaranteed to be best structure, don't believe `delayest` too much. For above data, it suggests a delay of three samples, but a delay of two samples is probably better.

Coefficient order

- To estimate model order, can use frequency response identification (*e.g.*, `spa.m`) to get a feel for the Bode plot.
  - "Eyeball" fit poles and zeros to get an idea of how many of each.
  - Pay particular attention to peaks in the magnitude plot, which indicate resonances (lightly damped complex-conjugate pole pair).
  - Slope of magnitude plot of some use, but care must be taken due to frequency warping of discrete-time Bode plots, and uncertainty of whether there are regions where slope doesn't change as much as might be expected due to a near pole-zero cancelation.

```
load dryer2
u2 = u2 - mean(u2);
y2 = y2 - mean(y2);
z = iddata(y2,u2,0.08);
[m,p,w]=bode(spa(z));
w=squeeze(w); m=squeeze(m);
semilogx(w,20*log10(m));
```



- Slope ≈ -40 dB per decade: at least two more poles than zeros.

  - But, how many zeros? Phase plot is pretty useless due to delays.
  - Might try a range of model orders from 2 to 5 to see which fits best.
- Typically get a better fit as we increase the order of $A(q)$.
  - Avoid "over fitting" the data—the result looks good on this data set, but much poorer on any other.
- MATLAB `selstruc` can help determine good starting points.

Polynomial form

- Much of the MATLAB system identification toolbox deals with transfer functions given in a polynomial format.
- We enter polynomials by typing their coefficients in vector form as ascending powers of the delay operator q^{-k}, $k = 0, 1, \ldots$

KEY POINT: Delays are denoted by leading zeros in the polynomial. So,

$$\frac{B(q)}{A(q)} = \frac{q^{-3}}{1 - 1.5q^{-1} + 0.7q^{-2}}$$

is entered by `B = [0 0 0 1]` and `A = [1 -1.5 0.7]`.

- Note that the MATLAB control-systems toolbox does this differently:
  - In discrete-time (z form), would normally write this as

$$\frac{z^{-3}}{1 - 1.5z^{-1} + 0.7z^{-2}} = \frac{1}{z^3 - 1.5z^2 + 0.7z},$$

    entered as `num = 1` and `den = [1 -1.5 0.7 0]`.
  - The two different toolbox representations are equivalent (only) if the lengths of A and B are equal.
- MATLAB commands to use polynomials in system ID are `idpoly` and `polydata`.

4.6: Fitting parametrized model: Simulation or prediction?

- Having chosen n_k and n_a, n_b, n_c, and n_d, it's now time to determine the polynomials $A(q)$, $B(q)$, etc.
- This is an optimization problem:
  - First, much choose what objective we are trying to optimize,
  - Then, develop a "cost function" to achieve this objective,
  - Then, select an appropriate optimization method.

Optimization objectives: Simulation or prediction?

- A very fundamental question we must consider is: what will the final model be used for?
  - Simulation: Given knowledge only of $u[k]$ up to the present time, estimate the present output $y[k]$.
  - 1-step prediction: Given knowledge of $u[k]$ up to the present time, and $y[k]$ up to the prior time, $k-1$, estimate present output $y[k]$.
  - n-step prediction: Given knowledge of $u[k]$ up to the present time, and $y[k]$ up to a previous time, $k-n$, estimate present output $y[k]$.
- A model optimized for a simulation application will generally match the open-loop response best.
- A model optimized for prediction will use measured data as feedback, and will generally provide better estimates, especially when there is non-white noise impacting the system response.
- Our basic approach will be to find $G(q)$ and possibly $H(q)$ to minimize some measure of modeling error:

- For simulation, $\epsilon_s[k] = y[k] - \hat{y}_s[k]$, where $\hat{y}_s[k] = G(q)u[k]$, the simulated value of the output at time k given measurements of the input only. (We don't estimate $H(q)$ for a simulation application.)
- For 1-step prediction, $\epsilon_p[k] = y[k] - \hat{y}_p[k \mid k-1]$, where $\hat{y}_p[k \mid k-1]$ is the predicted value of the output at time k given measurements of the output up until (and including) time $k-1$.
- For n-step prediction, $\epsilon_n[k] = y[k] - \hat{y}_n[k \mid k-n]$, where $\hat{y}_n[k \mid k-n]$ is the predicted value of the output at time k given measurements of the output up until (and including) time $k-n$.

- The formulations for the predicted $\hat{y}_p[k \mid k-1]$ and $\hat{y}_n[k \mid k-n]$ deserve some attention.
- We start with the assumed form $y[k] = G(q)u[k] + H(q)e[k]$.
- Pre-multiply both sides by $H^{-1}(q)$, assuming that $H(q)$ is monic, minimum-phase, and hence is also stable. This isolates $e[k]$.

$$H^{-1}(q)y[k] = H^{-1}(q)G(q)u[k] + e[k].$$

- Now, add $y[k]$ to both sides, and rearrange to get

$$y[k] = \left(1 - H^{-1}(q)\right) y[k] + H^{-1}(q)G(q)u[k] + e[k].$$

- Note that the RHS requires knowledge of $u[k]$ up until the present, $e[k]$ at the present time only, and $y[k]$ up until the prior time only.
- To see this last point, let's look at the details of $H(q)$ more closely:

$$H(q) = \frac{C(q)}{D(q)} = \frac{1 + c_1 q^{-1} + c_2 q^{-2} + ...}{1 + d_1 q^{-1} + d_2 q^{-2} + \cdots}$$

$$1 - H^{-1}(q) = \frac{C(q) - D(q)}{C(q)} = \frac{(c_1 - d_1)q^{-1} + (c_2 - d_2)q^{-2} + \cdots}{1 + c_1 q^{-1} + c_2 q^{-2} + \cdots}.$$

- The numerator has a pure delay term, therefore, $\left(1 - H^{-1}(q)\right) y[k]$ contains only old values of the output $\{y[s],\ s \leq k-1\}$.
- Therefore, we can use this relationship to predict $y[k]$ from past values of $y[k]$ (etc).
  - Assume that $e[k]$ is white, so best estimate of $e[k]$, given information up to $k-1$, is $\hat{e}[k] = 0$.
  - Therefore, $\hat{y}_p[k \mid k-1] = \left(1 - H^{-1}(q)\right) y[k] + H^{-1}(q)G(q)u[k]$.
- Can also derive similar relationship for n-step prediction,

$$\hat{y}_n[k \mid k-n] = \left(1 - \overline{H}_n(q)H^{-1}(q)\right) y[k] + \overline{H}_n(q)H^{-1}(q)G(q)u[k],$$

where $\overline{H}_n(q) = \sum_{j=0}^{n-1} h[j]q^{-j}$, which is a truncated version of $H(q)$.

EXAMPLE: For output error, $H(q) = 1$, and $G(q) = B(q)/A(q)$.

- This gives $(1 - H^{-1}(q)) = 0$ and $H^{-1}(q)G(q) = G(q)$.
  - So, $\hat{y}_p[k \mid k-1] = \dfrac{B(q)}{A(q)}u[k]$, which is not a function of past outputs.

EXAMPLE: For ARX, $H(q) = 1/A(q)$, and $G(q) = B(q)/A(q)$.

- This gives $(1 - H^{-1}(q)) = 1 - A(q)$ and $H^{-1}(q)G(q) = B(q)$.
  - Therefore,

$$\begin{aligned}\hat{y}_p[k|k-1] &= (1 - A(q))y[k] + B(q)u[k] \\ &= -a_1 y[k-1] - \cdots - a_{n_a} y[k-n_a] + b_1 u[k-1] + \cdots + b_{n_b} u[k-n_b].\end{aligned}$$

  - ARX uses old values of $y[k]$ as well as $u[k]$ to predict $\hat{y}_p[k \mid k-1]$.
- Other cases are more complicated.

4.7: Fitting parametrized model: Cost function

- Now that we have defined three different forms of instantaneous modeling error $\epsilon[k]$, would like to define an optimization strategy to minimize that error in some sense.
  - Note that $\epsilon[k]$ is parametrized by θ, which we write explicitly in this section as $\epsilon[k;\theta]$.
- Can take either a time-domain or frequency-domain approach: both require defining a "cost function" to be minimized.
- In the time domain, might want to choose parameters θ to minimize $V_N(\theta) = \sum_{k=1}^{N} \epsilon^2[k;\theta]$.
  - Solution, $\hat{\theta} = \arg\min_{\theta} V_N(\theta)$ is called the "least squares" solution.
  - More generally, might want to minimize $V_N(\theta) = \sum_{k=1}^{N} L(\epsilon[k;\theta])$, where $L(\cdot)$ is a "loss function" where $L \geq 0$ and L is a scalar.
- In the frequency domain, might want to minimize weighted linear least-squares frequency response

$$V_N(\theta) = \sum_{\omega_i} \alpha_i \left| G(e^{j\omega_i};\theta) - \widehat{G}_N(e^{j\omega_i}) \right|^2,$$

where $G(e^{j\omega_i};\theta)$ is the model frequency response at frequency ω_i with parameters θ, and $\widehat{G}_N(e^{j\omega_i})$ is the estimated frequency response (using `spa.m`, for example).

  - One of the problems with the linear least-squares frequency fit is that zeros of the system are fit *very poorly*.

- Especially true for lightly damped zeros.

- Reason is that the index assigns a very small penalty to errors in the match of the data/model in the low-gain regions.
  - Near these zeros, the absolute difference in the frequency responses is small (relative errors are large).
- Linear least squares puts too much emphasis on fitting the poles.

- Logarithmic least squares is better for the fitting of the zeros since it weights the ratio of model gain to measurement gain.

$$V_N(\theta) = \sum_{\omega_i} \alpha_i \left| \log(G(e^{j\omega_i};\theta)) - \log(\widehat{G}_N(e^{j\omega_i})) \right|^2.$$

- Works much better for a system with large dynamic range[3].

EXAMPLE: Consider two points in the transfer function

- Measured data: $\widehat{G}(\omega_1) = 10$ and $\widehat{G}(\omega_2) = 0.1$.
- Our model estimates these as: $G(\omega_1;\theta) = 9$ and $G(\omega_2;\theta) = 0.09$.
- Check contribution to the cost functions:

$$V_{\text{lin}} = \underbrace{(10-9)^2}_{1} + \underbrace{(0.1-0.09)^2}_{0.0001 \ll 1}$$

$$V_{\text{lls}} = \underbrace{(\log(10)-\log(9))^2}_{0.0111} + \underbrace{(\log(0.1)-\log(0.09))^2}_{0.0111}.$$

Optimization method

- Most cost functions and models require nonlinear optimization methods, which we look at in the sequel.
- However, a simple solution to the linear least-squares ARX problem exists. We look at this first.

[3] Sidman, IEEE TAC, 36, p. 1065, 1991.

4.8: Solving the linear least-squares ARX problem

- Consider quadratic case first, linear prediction form of ARX, where

$$\begin{aligned}\hat{y}_p[k|k-1] &= (1-A(q))y[k] + B(q)u[k] \\ &= -a_1 y[k-1] - \cdots - a_{n_a} y[k-n_a] + b_1 u[k-1] + \cdots + b_{n_b} u[k-n_b] \\ &= \theta^T \phi[k],\end{aligned}$$

where

$$\theta = \begin{bmatrix} a_1 & a_2 & \cdots & a_{n_a} & b_1 & b_2 & \cdots & b_{n_b} \end{bmatrix}^T$$

$$\phi[k] = \begin{bmatrix} -y[k-1] & \cdots & -y[k-n_a] & u[k-1] & \cdots & u[k-n_b] \end{bmatrix}^T.$$

- Note that the prediction error is linear in θ

$$\epsilon_p[k;\theta] = y[k] - \hat{y}_p[k;\theta] = y[k] - \theta^T \phi[k].$$

- So, we can use regression to solve for θ.[4] Define

$$\begin{aligned}V_N(\theta) &= \sum_{k=1}^{N} \left(y[k] - \theta^T \phi[k]\right)^2 \\ &= \sum_{k=1}^{N} \left(y^2[k] - 2\theta^T \phi[k] y[k] + \theta^T \phi[k] \phi^T[k] \theta\right) \\ &= \underbrace{\left(\sum_{k=1}^{N} y^2[k]\right)}_{y_N} - 2\theta^T \underbrace{\left(\sum_{k=1}^{N} \phi[k] y[k]\right)}_{f_N} + \theta^T \underbrace{\left(\sum_{k=1}^{N} \phi[k] \phi^T[k]\right)}_{R_N} \theta \\ &= y_N - 2\theta^T f_N + \theta^T R_N \theta.\end{aligned}$$

[4] Note, the term "regress" here alludes to the fact that we try to calculate or describe $y[k]$ by "going back" to $\phi[k]$. Also, models such as ARX where $\phi[k]$ contains old values $y[k-\tau]$ of the variable to be explained, $y[k]$, are then partly "auto-regression" models.

- We assume that R_N is invertible, giving

$$V_N(\theta) = \underbrace{y_N - f_N^T R_N^{-1} f_N}_{\text{not a function of } \theta} + \underbrace{(\theta - R_N^{-1} f_N)^T R_N (\theta - R_N^{-1} f_N)}_{\text{non-negative since } R_N \geq 0}.$$

- We get the smallest possible $V_N(\theta)$ when we select $\theta = \hat{\theta}_N = R_N^{-1} f_N$.
- Can also formulate using vectors,

$$X = \begin{bmatrix} \phi[1] & \phi[2] & \cdots & \phi[N] \end{bmatrix}^T$$

$$Y = \begin{bmatrix} y[1] & y[2] & \cdots & y[N] \end{bmatrix}^T,$$

  which allows us to write $R_N = X^T X$ and $f_N = X^T Y$.
- This also gives

$$V_N = (Y - X\theta)^T (Y - X\theta)$$

  and $\hat{\theta}_N = (X^T X)^{-1} X^T Y$ (the least-squares estimate).
- In MATLAB, `theta = X \ Y`.

Solving other parameter estimation optimizations

- The prediction ARX problem with quadratic cost function is the only system ID that can be done using linear least squares.
- Other problems require nonlinear optimization, which is quite tricky.
- One fundamental principle is that we know we are at a minima (or maxima) of an objective function when $\mathrm{d}V_N(\theta)/\mathrm{d}\theta = 0$.
- Comparing to the matrix form of the prior case, we get

$$\frac{\mathrm{d}}{\mathrm{d}\theta} V_N = 2R_N\theta - 2f_N = 0$$

$$\hat{\theta} = R_N^{-1} f_N,$$

  which is consistent with the result we obtained.

- In the more general case (*i.e.*, most other model forms),
  - $\epsilon[k;\theta]$ is not linear in θ, and $V_N(\theta)$ is not quadratic in $\epsilon[k;\theta]$.
  - Hence, $V_N(\theta)$ is not quadratic in θ.
  - Optimization is not as simple. No closed-form solutions are available. Nonlinear optimization is required.
- So, next substantial section of notes is an overview of nonlinear optimization, *in general,* using the Newton/quasi-Newton methods.
- We then return to the system identification problem, where we apply these general results to our problem.

4.9: Nonlinear optimization

- Here, we re-cast the problem into more generic notation.[5]
  - Instead of finding $\hat{\theta}$ that minimizes $V_N(\theta)$, we are find x^* that minimizes $F(x)$, where x is a vector and $F(x)$ is a scalar.

$$x^* = \arg\min_x F(x).$$

  - This scenario encompasses the ARMAX, OE, and BJ cases, plus many other unconstrained optimization problems.
- Numerical methods for nonlinear optimization are iterative. Typically, this is done using $\hat{x}_{k+1} = \hat{x}_k + \alpha_k p_k$, where,
  - $\hat{x}_k$ is the estimate of the optimizing x^* at algorithm iteration k,
  - p_k is a search direction to look for the minimum, and
  - α_k is a positive constant determined so that an appropriate decrease in the value of $F(x)$ is observed.
  - Note that I am using subscripts to denote iteration number at a specific sample number (*i.e.*, given a fixed set of data, length N). This is different from time sample number, for which I still use square brackets $[\cdot]$, when appropriate.
- Available methods primarily differ in how they find p_k and α_k.
  - Some methods use function values only;
  - Some methods use values of F as well as values of its gradient $g(\hat{x}_k)$ (the first derivative vector);

[5] Some references for this section include: L.E. Scales, Introduction to Non-Linear Optimization, Springer-Verlag, 1985, and M.A. Wolfe, Numerical Methods for Unconstrained Optimization: An Introduction, Van Nostrand Reinhold, 1978.

  - Others use values of F, its gradient, and its Hessian $G(\hat{x}_k)$ (the second derivative matrix).

- The typical member of the third group corresponds to Newton algorithms, where the search direction is computed as

$$p_k = -G^{-1}(\hat{x}_k)g(\hat{x}_k).$$

- The most important subclass of the second group consists of quasi-Newton algorithms, which somehow form an estimate of the Hessian and then compute

$$p_k = -\widehat{G}^{-1}(\hat{x}_k)g(\hat{x}_k).$$

- Methods from the first group generally form gradient estimates by difference approximations and proceed as quasi-Newton methods,

$$p_k = -\widehat{G}^{-1}(\hat{x}_k)\hat{g}(\hat{x}_k).$$

- The basic idea is that the minimum lies (more or less) in the direction of the negative gradient of $F(x)$ with respect to x.
- Via gradient descent, we eventually get to (at least a local) minimum.
  - However, for objective functions having both steep regions and flat regions (or dimensions), gradient descent is very slow.
  - Scaling by the local curvature allows us to take different step sizes in different dimensions, greatly speeding up the search.
  - Note that the scaling G^{-1} is large when the curvature is small (can go long distances before local gradient is invalid estimate of slope), and is small when the curvature is large (can go only a short distance before local gradient no longer reliable indicator of slope).

- Can use fixed value for $\alpha_k = \alpha$, but generally get better computational performance if we do a line search to find the value for α_k that minimizes $F(x)$ along the search direction p_k. Line search:
  - First looks for bracket $[a, b]$ in which there is a minimum; then
  - Iteratively reduces bracket length until "close enough" to minimum.
- Overall process: Compute p_k; compute α_k; update $\hat{x}_k$; repeat.

NOTE: $\hat{x}_0$ tends to be very important.

NOTE: In general, nonlinear optimization will find only a local minimum.

NOTE: Convergence can be very slow.

NOTE: Guarantees on getting a good final answer in a reasonable amount of time? No. But, very often do so anyway.

4.10: Generic nonlinear optimization example

EXAMPLE: Consider the Rosenbrock function, a non-convex problem that is difficult to minimize:

$$F(x) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2,$$

where the global minimum is at $x^* = [\ 1 \quad 1\ ]^T$.

- Start with $x_0 = [\ -1.2 \quad 1\ ]^T$ and see how different methods work.
- We will use `fminunc.m` from MATLAB's optimization toolbox to try different algorithm possibilities.
- First, create a MATLAB function to implement the Rosenbrock function

```
function F=rosen(x)
  global xpath;
  F=100*(x(1)^2-x(2))^2+(1-x(1))^2;
  xpath=[xpath;x'];
end
```

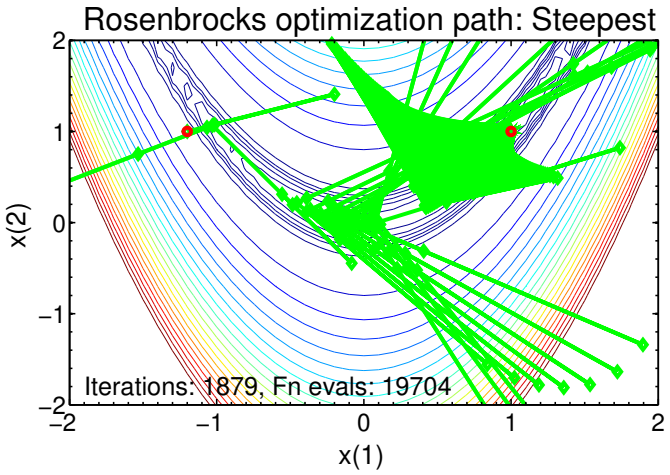
- Next, in a separate script, build up the surface we're trying to minimize

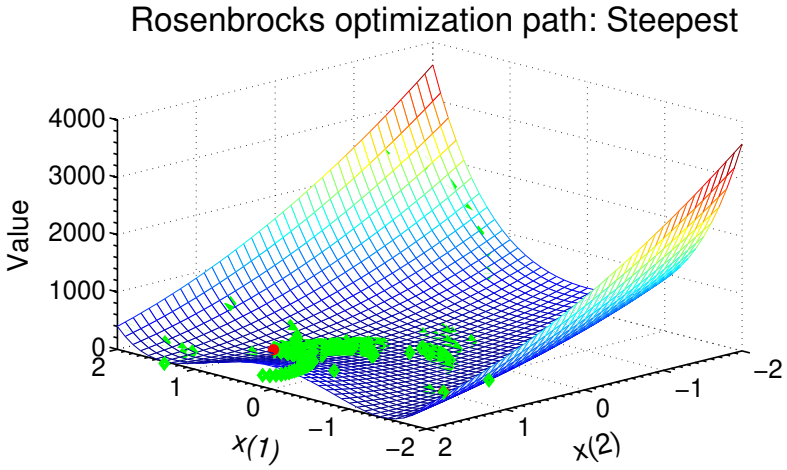
```
  global xpath;
  x1=[-2:.1:2]'; x2=x1; N=length(x1); FF = zeros(N,N);
  for ii=1:N,
    for jj=1:N,
      FF(ii,jj)=rosen([x1(ii) x2(jj)]');
    end
  end
```

- First, use steepest descent. Lots of iterations required to get solution.

```
xpath=[];
options = optimset('LargeScale','off','HessUpdate','steepdesc',...
      'MaxFunEvals',20000,'MaxIter',2000);
[X,FVAL,EXITFLAG,OUTPUT]=fminunc('rosen',[-1.2 1]',options);
```

```
figure; clf; contour(x1,x2,FF',[0:2:10 15:50:1000]); hold on
plot(xpath(:,1),xpath(:,2),'gd'); plot(xpath(:,1),xpath(:,2),'g-');
xlabel('x(1)'); ylabel('x(2)');
title('Rosenbrocks optimization path: Steepest')
plot(-1.2,1,'ro'); plot(1,1,'ro')
text(-1.8,-1.8,sprintf('Iterations: %d, Fn evals: %d',...
     OUTPUT.iterations,OUTPUT.funcCount),'fontsize',16);

figure;clf; mesh(x1,x2,FF'); hold on
for ii=1:length(xpath);
  plot3(xpath(ii,1),xpath(ii,2),0.1+rosen(xpath(ii,:)'),'gd',...
        'markersize',5)
end
plot3(-1.2,1,1+rosen([-1.2 1]'),'r.','markersize',25)
plot3(1,1,1+rosen([1 1]'),'r.','markersize',25)
xlabel('x(1)'); ylabel('x(2)'); zlabel('Value');
title('Rosenbrocks optimization path: Steepest')
campos([-22.248, 21.81, 17143]); xlim([-2 2]); ylim([-2 2]);
```

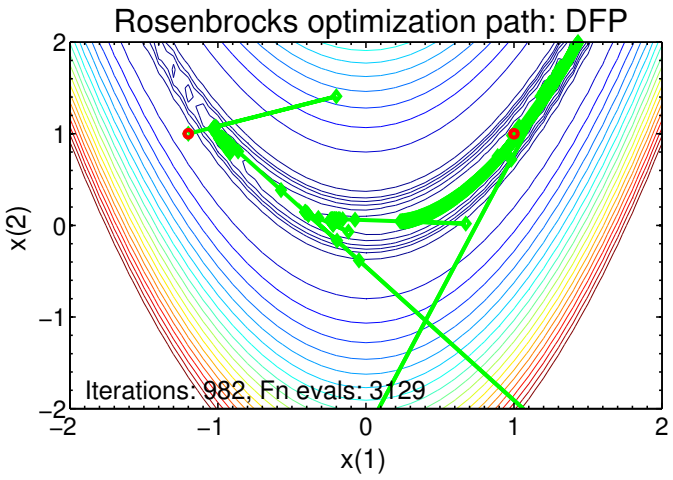


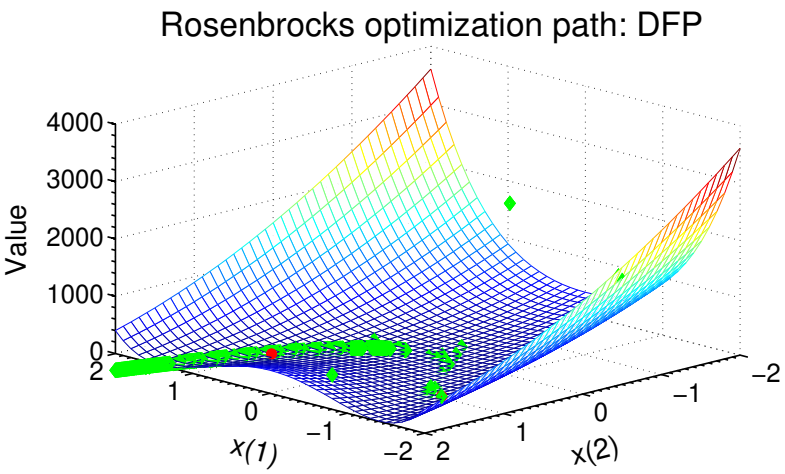


- Next, use a quasi-Newton method, with the “Davidson, Fletcher, Powell (DFP) approach” to recursively build up approximate Hessian inverse, using the function and its approximate gradients only.

```
xpath=[];
options = optimset('LargeScale','off','HessUpdate','dfp',...
    'MaxFunEvals',5000,'MaxIter',2000);
[X,FVAL,EXITFLAG,OUTPUT] = fminunc('rosen',[-1.2 1]',options);
```

```
figure; clf;  contour(x1,x2,FF',[0:2:10 15:50:1000]); hold on
plot(xpath(:,1),xpath(:,2),'gd'); plot(xpath(:,1),xpath(:,2),'g-');
xlabel('x(1)'); ylabel('x(2)');
title('Rosenbrocks optimization path: DFP')
plot(-1.2,1,'ro'); plot(1,1,'ro')
text(-1.8,-1.8,sprintf('Iterations: %d, Fn evals: %d',...
    OUTPUT.iterations,OUTPUT.funcCount),'fontsize',16);

figure; clf; mesh(x1,x2,FF'); hold on
for ii=1:length(xpath);
  plot3(xpath(ii,1),xpath(ii,2),0.1+rosen(xpath(ii,:)'),'gd',...
        'markersize',5)
end
plot3(-1.2,1,1+rosen([-1.2 1]'),'r.','markersize',25)
plot3(1,1,1+rosen([1 1]'),'r.','markersize',25)
xlabel('x(1)'); ylabel('x(2)'); zlabel('Value');
title('Rosenbrocks optimization path: DFP')
campos([-22.248, 21.81, 17143]); xlim([-2 2]); ylim([-2 2]);
```

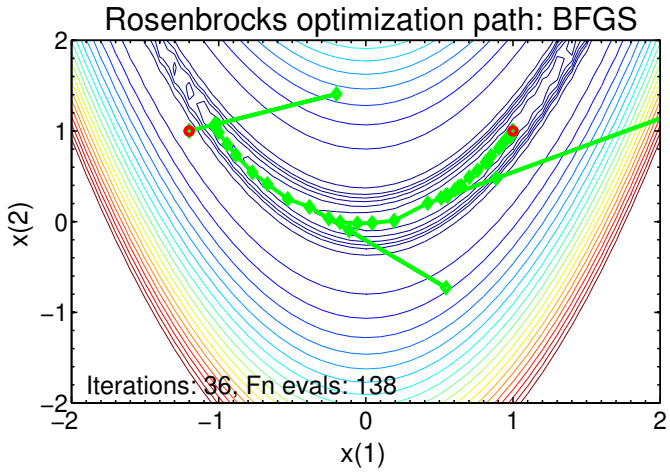


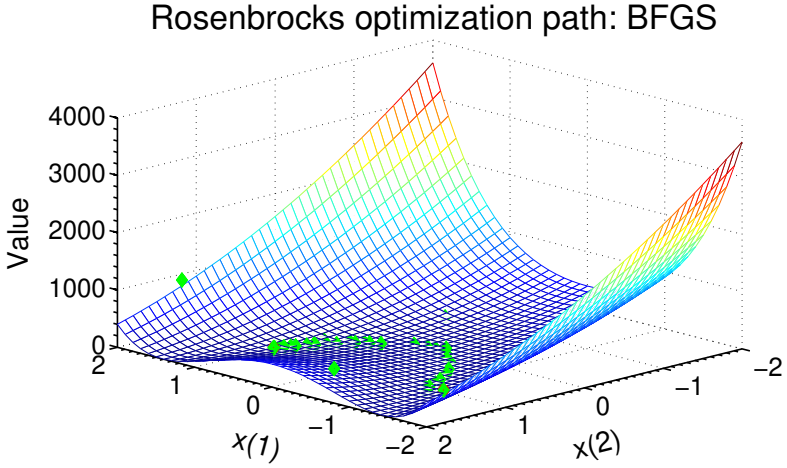


- Finally, use a quasi-Newton method, with the “Broyden, Fletcher, Goldfarb, Shano (BFGS) approach” to building up the Hessian.

```
xpath=[];
options = optimset('LargeScale','off'); % select quasi-Newton; BFGS
[X,FVAL,EXITFLAG,OUTPUT]=fminunc('rosen',[-1.2 1]',options);

figure; clf; contour(x1,x2,FF',[0:2:10 15:50:1000]); hold on
plot(xpath(:,1),xpath(:,2),'gd'); plot(xpath(:,1),xpath(:,2),'g-');
xlabel('x(1)'); ylabel('x(2)');
```

```
title('Rosenbrocks optimization path: BFGS')
plot(-1.2,1,'ro'); plot(1,1,'ro')
text(-1.8,-1.8,sprintf('Iterations: %d, Fn evals: %d',...
    OUTPUT.iterations,OUTPUT.funcCount),'fontsize',16);

figure; clf; mesh(x1,x2,FF'); hold on
for ii=1:length(xpath);
  plot3(xpath(ii,1),xpath(ii,2),0.1+rosen(xpath(ii,:)'),'gd',...
         'markersize',5)
end
plot3(-1.2,1,1+rosen([-1.2 1]'),'r.','markersize',25)
plot3(1,1,1+rosen([1 1]'),'r.','markersize',25)
xlabel('x(1)'); ylabel('x(2)'); zlabel('Value');
title('Rosenbrocks optimization path: BFGS')
campos([-22.248, 21.81, 17143]); xlim([-2 2]); ylim([-2 2]);
```





- Quasi-Newton (BFGS) by far the most efficient of those tried. (It's MATLAB's default for medium-scale problems.)
- It would be fun to spend a lot more time looking at methods for nonlinear optimization, but that isn't really the purpose of this course.
  - So, we press on, switching our attention back to system ID;
  - If you are interested in methods of optimization, take *ECE 5570: Optimization Methods for Systems and Control*.

4.11: Toolbox methods (1): Frequency response

- MATLAB's system identification toolbox contains commands `arx.m`, `armax.m`, `bj.m`, and `oe.m`, which use measured system input–output data to produce optimized polynomials $A(q)$, $B(q)$, $C(q)$, and $D(q)$, as appropriate, for each model type.
  - Must supply model delay n_k and model size n_a, n_b, n_c, and n_d.
- The best way to see these functions in action is via example.

EXAMPLE: Consider a continuous-time system

$$G(s) = \frac{1}{s^2 + 0.5s + 1}.$$

- Actual system configured with OE structure (*i.e.*, $H(s) = 1$).
- First step in simulation is to convert model from continuous time to discrete time using ZOH, sampled at 2 Hz

  $$y[k] = G(q)u[k] + e[k],$$

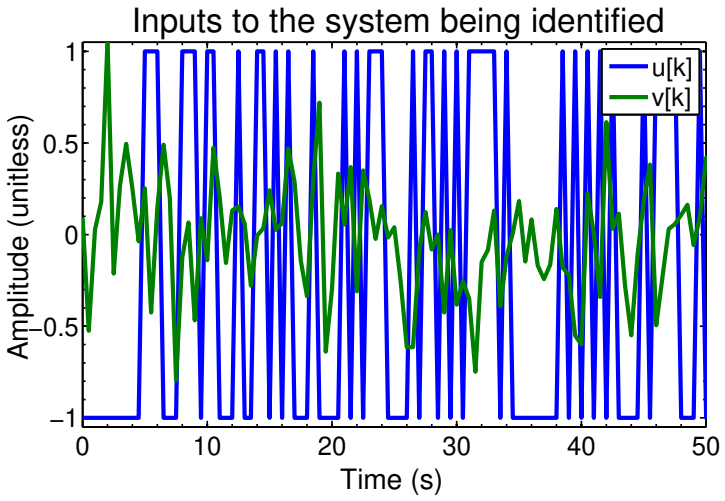
  giving the discrete-time transfer function

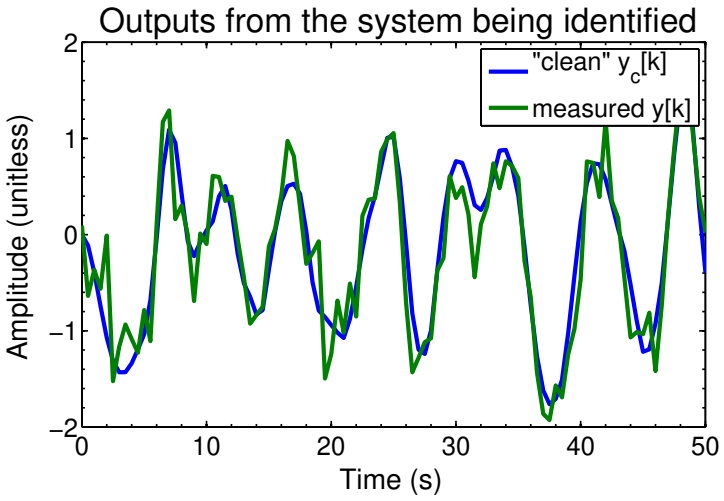
  $$G(q) = \frac{0.1129q^{-1} + 0.1038q^{-2}}{1 - 1.5622q^{-1} + 0.7788q^{-2}}.$$

- Used noise with $\mathbb{E}[e^2[k]] = \sigma^2 \approx 0.15$, but I scaled its power automatically in code to be equal to 1/4 of signal power.
- Simulate and plot random input–output sequence:

```
%% Modified from code originally by Jonathan P. How
%% Part 1: Setup
clear; clc; close all
Npts=512; T = 0.5; t=(0:1:Npts-1)*T; % # data points, sample period
RandStream.setGlobalStream(RandStream('mcg16807','Seed', 15));
```

```
nn=1; dd=[1 .5 1]; % continuous-time system dynamics
[num,den]=c2dm(nn,dd,T,'zoh'); % discrete-time system dynamics
w_arx=logspace(-2,0,300)*pi/T;

% compute input sequence for identification
u = idinput(Npts,'prbs');  % input signal for identification
yc = dlsim(num,den,u);     % compute "clean" output of G
v = randn(Npts,1);         % noise - note that this is white, gaussian
LL = 0.25*(yc'*yc)/(v'*v); % scale so energy in sensor noise 1/4 times
v = sqrt(LL)*v;                    %   the energy in the "clean" signal
y = yc+v;                  % actual output y=Gu+v
Z = iddata(y,u,T);         % data available to identification

% compute input sequence for validation
u_val = idinput(Npts,'prbs');   % input signal for validation
yc_val = dlsim(num,den,u_val);  % "clean" output of G
v_val = sqrt(LL)*randn(Npts,1); % noise - white, gaussian
y_val = yc_val+v_val;           % actual output y=Gu+v
Z_val = iddata(y_val,u_val,T);  % data available for validation

% plot portions of input and output signals after initial transient
figure; plot(t,[u v]); legend('u[k]','v[k]');
title('Inputs to the system being identified');
xlabel('Time (s)'); ylabel('Amplitude (unitless)');

figure; plot(t,[yc y]); legend('"clean" y_c[k]','measured y[k]');
title('Outputs from the system being identified');
xlabel('Time (s)'); ylabel('Amplitude (unitless)');
```

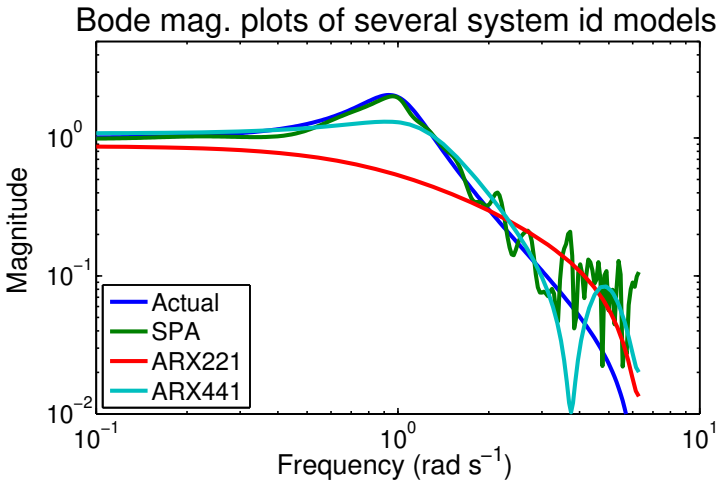


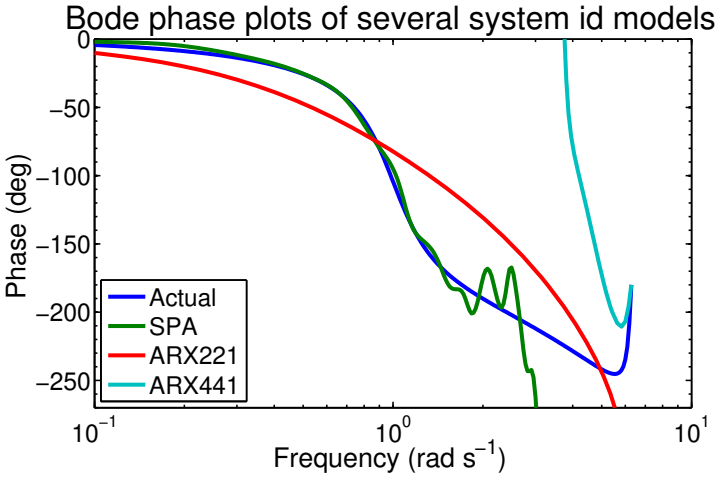


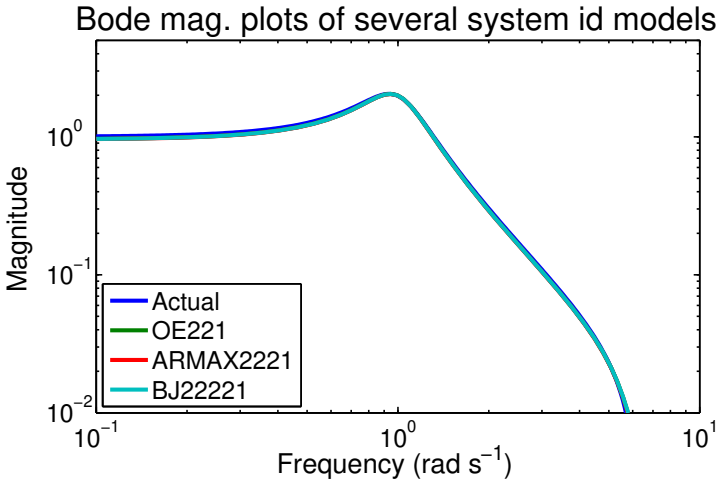
- Identify system models for several different model structures, and show Bode plots of the results:

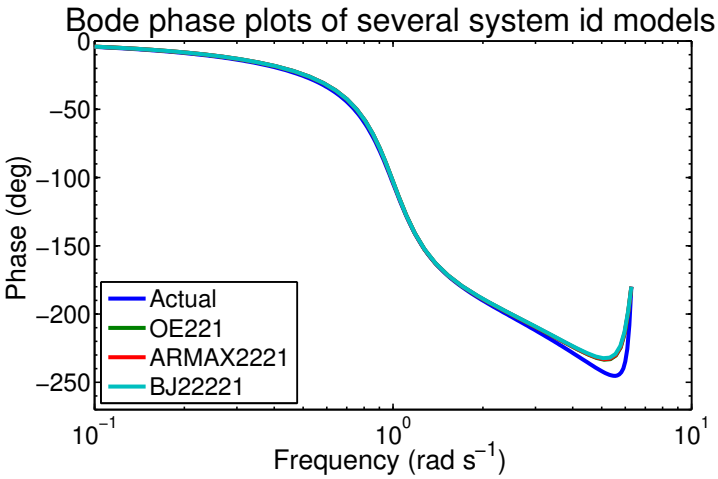
```
% frequency response of actual system, and "SPA" frequency resp model
[mag,ph,w]=dbode(num,den,T,w_arx); % get "true" magnitude and phase resp
G = spa(Z,64,w,[],T); [amp,phas,w]=bode(G); w = squeeze(w);
amp = squeeze(amp); phas = squeeze(phas);

% ARX model with na=2; nb=2; nk=1 (ARX221)
M_arx221 = arx(Z,'na',2,'nb',2,'nk',1);
[m_arx221,p_arx221,w_arx221]=bode(M_arx221); w_arx221 = squeeze(w_arx221);
m_arx221 = squeeze(m_arx221); p_arx221 = squeeze(p_arx221);
[a_arx221,b_arx221,c_arx221,d_arx221,f_arx221] = polydata(M_arx221);

% ARX model with na=4; nb=4; nk=1 (ARX441)
M_arx441 = arx(Z,'na',4,'nb',4,'nk',1);
[m_arx441,p_arx441,w_arx441]=bode(M_arx441); w_arx441 = squeeze(w_arx441);
m_arx441 = squeeze(m_arx441); p_arx441 = squeeze(p_arx441);
[a_arx441,b_arx441,c_arx441,d_arx441,f_arx441] = polydata(M_arx441);

% ARMAX model with na=2; nb=2; nc=2; nk=1 (ARMAX2221)
M_armax=armax(Z,'na',2,'nb',2,'nc',2,'nk',1);
[a_armax,b_armax,c_armax,d_armax,f_armax]=polydata(M_armax);
[m_armax,p_armax,w_armax]=bode(M_armax); w_armax = squeeze(w_armax);
m_armax = squeeze(m_armax); p_armax = squeeze(p_armax);

% Box-Jenkins model with nb=2; nc=2; nd=2; nf=2; nk=1 (BJ22221)
% y(t) = [B(q)/F(q)] u(t-nk) + [C(q)/D(q)] e(t)
M_bj=bj(Z,'nb',2,'nc',2,'nd',2,'nf',2,'nk',1);
[m_bj,p_bj,w_bj]=bode(M_bj); w_bj = squeeze(w_bj);
m_bj = squeeze(m_bj); p_bj = squeeze(p_bj);
[a_bj,b_bj,c_bj,d_bj,f_bj]=polydata(M_bj);

% OE model with nb=2; nf=2; nk=1;
M_oe = oe(Z,'nb',2,'nf',2,'nk',1);
[m_oe,p_oe,w_oe]=bode(M_oe); w_oe = squeeze(w_oe);
m_oe = squeeze(m_oe); p_oe = squeeze(p_oe);
[a_oe,b_oe,c_oe,d_oe,f_oe]=polydata(M_oe);

% Now, plot Bode plots
figure; loglog(w,mag,w,amp,w_arx221,m_arx221,w_arx441,m_arx441);
title('Bode mag. plots of several system id models');
```

```
ylabel('Magnitude'); xlabel('Frequency (rad s^{-1})');
legend('Actual','SPA','ARX221','ARX441'); axis([.08 8 1e-2 5]);

figure; semilogx(w,ph,w,phas,w_arx221,p_arx221,w_arx441,p_arx441);
title('Bode phase plots of several system id models');
xlabel('Frequency (rad s^{-1})'); ylabel('Phase (deg)');
legend('Actual','SPA','ARX221','ARX441'); axis([.08 8 -270 0]);

figure; loglog(w,mag,w_oe,m_oe,w_armax,m_armax,w_bj,m_bj);
title('Bode mag. plots of several system id models');
ylabel('Magnitude'); xlabel('Frequency (rad s^{-1})');
legend('Actual','OE221','ARMAX2221','BJ22221'); axis([.08 8 1e-2 5]);

figure; semilogx(w,ph,w_oe,p_oe,w_armax,p_armax,w_bj,p_bj);
title('Bode phase plots of several system id models');
xlabel('Frequency (rad s^{-1})'); ylabel('Phase (deg)');
legend('Actual','OE221','ARMAX2221','BJ22221'); axis([.08 8 -270 0]);
```







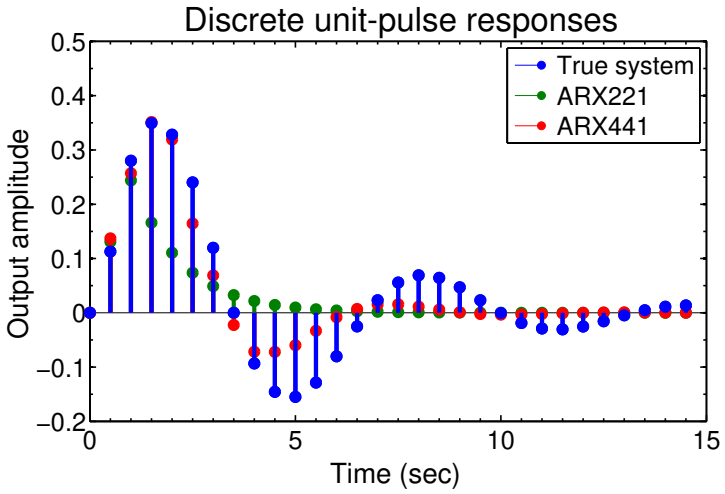


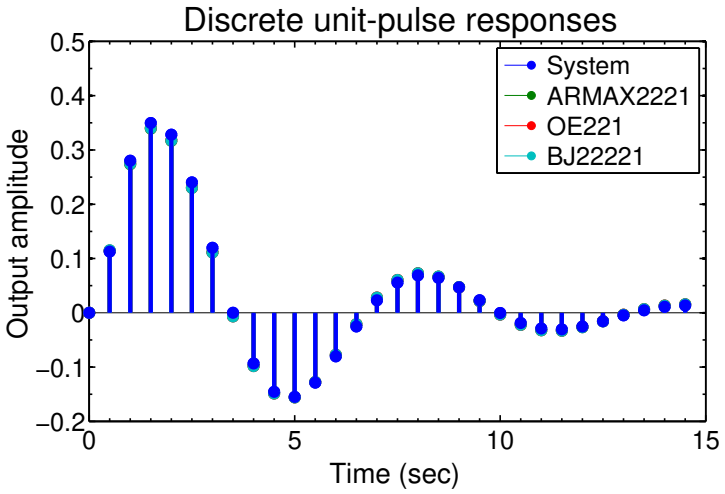
4.12: Toolbox methods (2): Unit-pulse response, residuals

- We continue the prior example by looking at the true and estimated system discrete-time unit-pulse responses.

```
Ntime=30;
y_act    = dimpulse(num,den,Ntime);
y_arx221 = dimpulse(b_arx221,a_arx221,Ntime);
y_arx441 = dimpulse(b_arx441,a_arx441,Ntime);
y_armax  = dimpulse(b_armax,a_armax,Ntime);
y_oe     = dimpulse(b_oe,f_oe,Ntime);
y_bj     = dimpulse(b_bj,f_bj,Ntime);

figure; stem([0:Ntime-1]*T,[y_act y_arx221 y_arx441],'filled'); hold on
stem([0:Ntime-1]*T,y_act,'filled');
legend('True system','ARX221','ARX441');
title('Discrete impulse responses')
xlabel('Time (sec)'); ylabel('Output amplitude')

figure; stem([0:Ntime-1]*T,[y_act y_armax y_oe y_bj],'filled'); hold on
stem([0:Ntime-1]*T,y_act,'filled');
legend('System','ARMAX2221','OE221','BJ22221');
title('Discrete impulse responses')
xlabel('Time (sec)'); ylabel('Output amplitude')
```

- Here are the impulse responses of the various methods attempted.





- We can also tabulate transfer-function coefficients:

```
[['Act  B    ',num2str(num,4)]       [' Act  A    ',num2str(den,4)]; ...
 ['ARX221 B ',num2str(b_arx221,4)] [' ARX221 A ',num2str(a_arx221,4)];...
 ['ARMAX B  ',num2str(b_armax,4)]  [' ARM  A    ',num2str(a_armax,4)]; ...
 ['BJ   B    ',num2str(b_bj,4)]      [' BJ   A    ',num2str(f_bj,4)]; ...
 ['OE   B    ',num2str(b_oe,4)]      [' OE   A    ',num2str(f_oe,4)]]
```

- Values for this example are tabulated below:
  - Second-order ARX isn't even close.
  - Fourth-order ARX isn't directly comparable.
  - Other three provide good estimates of $\hat{G}(q)$.

	b_0	b_1	b_2	a_0	a_1	a_2
Actual	0	0.1129	0.1038	1	−1.562	0.7788
ARX221	0	0.1309	0.1538	1	−0.689	0.0154
ARMAX2221	0	0.1144	0.0941	1	−1.569	0.7874
BJ22221	0	0.1153	0.0940	1	−1.569	0.7872
OE221	0	0.1148	0.0948	1	−1.568	0.7869

Model validation: A first step

- Usually we do not know the "actual system" dynamics, so how do we establish if our model is good?
- Various types of tests can be performed:
  - Prediction and simulation errors,
  - Frequency-response fit.
- Make sure you use different data to validate (if possible).
- Can also perform very detailed analysis of the residuals.

$$\epsilon[k] = y[k] - \hat{y}[k \mid k-1]$$

$$= y[k] - \left(1 - H^{-1}(q)\right) y[k] - H^{-1}(q)G(q)u[k]$$
$$= H^{-1}(q)\left(y[k] - G(q)u[k]\right).$$

- Called the innovations process and it contains a lot of information about the quality of our fit.
- A first desirable property of the residuals is that they be zero mean and normally distributed (at least symmetric).
  - Analyze using a histogram of $\epsilon[k]$.
- We can also use the residuals $\epsilon^2[k]$ to estimate σ^2

$$\hat{\sigma}^2 = \frac{1}{N}\sum_{k=1}^{N} \epsilon^2[k].$$

  - Natural if $H(q) = 1$ as $\epsilon[k] = y[k] - \hat{y}[k]$ is a good estimate of $e[k]$.
- Residuals can be computed as:

```
e_arx2 = resid(M_arx221,Z); e_arx2 = e_arx2.OutputData;
e_arx4 = resid(M_arx441,Z); e_arx4 = e_arx4.OutputData;
e_arm  = resid(M_armax,Z);  e_arm  = e_arm.OutputData;
e_bj   = resid(M_bj,Z);     e_bj   = e_bj.OutputData;
e_oe   = resid(M_oe,Z);     e_oe   = e_oe.OutputData;
mean([v e_arx2 e_arx4 e_arm e_bj e_oe])'
```

- Approximate noise variance can be computed as:

```
[['sigma^2 arx  ',sprintf('%1.4f',M_arx221.NoiseVariance)];...
 ['sigma^2 arm  ',sprintf('%1.4f',M_armax.NoiseVariance)];...
 ['sigma^2 BJ   ',sprintf('%1.4f',M_bj.NoiseVariance)];...
 ['sigma^2 OE   ',sprintf('%1.4f',M_oe.NoiseVariance)];...
 ['sigma^2 act  ',sprintf('%1.4f',LL)]]
```

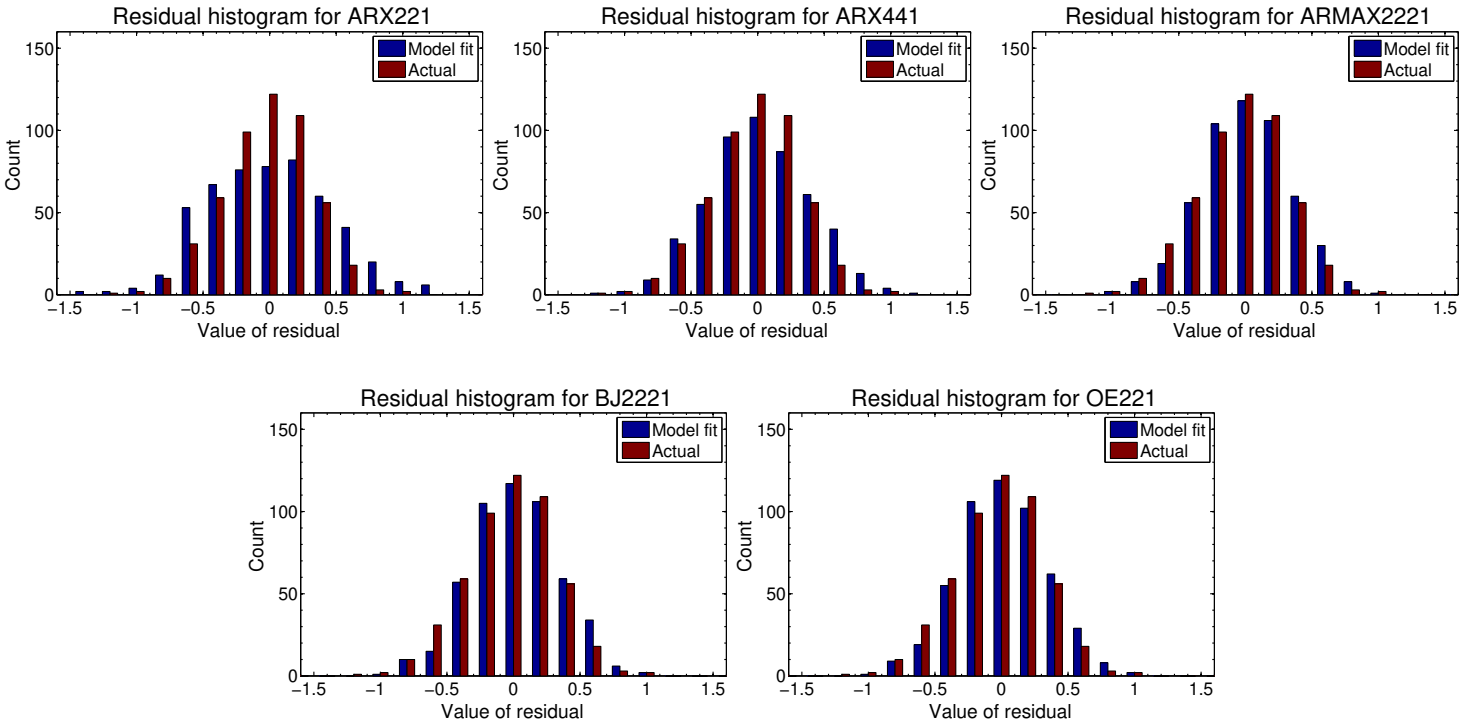
- Results shown below for noise-mean and noise-power estimation
  - Again, second-order ARX not even close.

- Other three provide good estimates.

	Actual	ARX221	ARMAX	BJ2221	OE221
Mean	-0.0221	-0.0087	-0.0272	-0.0281	-0.0249
σ^2	0.1118	0.2206	0.1085	0.1091	0.1083

- Look for symmetry and normal distribution in histograms of residuals:

```
% Plot a histogram of the residuals for ARX221
figure; hist([e_arx2 v],-2:0.2:2); axis([-1.6 1.6 0 160])
title('Residual histogram for ARX221');
ylabel('Count');xlabel('Value of residual')
legend('Model fit','Actual');

% Similar for the other cases. Omitting code for figure formatting...
figure; hist([e_arx4 v],-2:0.2:2);
figure; hist([e_arm v],-2:0.2:2);
figure; hist([e_bj v],-2:0.2:2);
figure; hist([e_oe v],-2:0.2:2);
```



- Again, ARX models do not match data very well; ARMAX, BJ and OE much better.

4.13: Toolbox methods (3): Model validation using correlations

- Two other desirable properties of residuals are:
  - White: We want $\epsilon[k]$ to look like what we assumed for $e[k]$.
  - Residuals uncorrelated with past inputs: If there are traces of past inputs in the residuals, then a part of $y[k]$ that originates from the input was not captured well in our model (bad).
- Analyze for whiteness by computing residual autocorrelation

$$\widehat{R}_{\epsilon}^{N}[\tau] = \frac{1}{N}\sum_{k=\tau}^{N} \epsilon[k]\epsilon[k-\tau],$$

  which we desire to be (roughly) zero everywhere except at $\tau = 0$.
- Analyze the second by computing cross-correlation between residuals and input

$$\widehat{R}_{\epsilon u}^{N}[\tau] = \frac{1}{N}\sum_{k=\tau}^{N} \epsilon[k]u[k-\tau],$$

  where $\tau > 0$ correlates $\epsilon[k]$ with old $u[k-\tau]$, and so we desire $\widehat{R}_{\epsilon u}^{N}[\tau]$ to be (roughly) zero for $\tau > 0$.
- Both analysis tests of the correlation graph need a measure of "small enough," which must be developed from the data as well.
- Can develop this by analyzing the statistics of the residuals.

WHITENESS: The numbers $\widehat{R}_{\epsilon}^{N}$ carry information regarding whether the residuals can be regarded as white.

- We can test for whiteness by first defining

$$r_{N,M} = \frac{\sqrt{N}}{\widehat{R}_{\epsilon}[0]}\left[\begin{array}{ccc} \widehat{R}_{\epsilon}[1] & \cdots & \widehat{R}_{\epsilon}[M] \end{array}\right]^{T}.$$

- Then, according to the central limit theorem, as $N \to \infty$, $r_{N,M}$ will be normally distributed with zero mean, and unit variance.
- Thus, if we sum together squares of $r_{N,M}$,

$$\zeta_{N,M} = \frac{N}{\left(\widehat{R}_\epsilon^N[0]\right)^2} \sum_{\tau=1}^{M} \left(\widehat{R}_\epsilon^N[\tau]\right)^2$$

  should be asymptotically $\chi^2(M)$ distributed.
- So, we develop an *overall* test on the residuals by checking whether $\zeta_{N,M}$ passes the test of being $\chi^2(M)$ distributed. That is, by checking that $\zeta_{N,M} < \chi^2_\alpha(M)$.
- More instructive is to look at the residuals *individually*, using the confidence intervals for a normal distribution.
  - For a 95% confidence level, we can use the ± 1.96 bounds on each element of $r_{N,M}[\tau]$ to decide if the autocorrelation is small for $\tau > 0$.
    - Plot $r_{N,M}[k]$ for $1 \leq k \leq M$.
    - Test for normality by ensuring that $r_{N,M}[k]$ within the confidence interval for all k.

CROSSCORRELATION TEST: As $N \to \infty$ can show $\sqrt{N}\widehat{R}_{\epsilon u}[\tau]$ is normally distributed, with zero mean and variance $P_r = \sum_{k=-\infty}^{\infty} R_\epsilon[k] R_u[k]$.

- Can perform a normality test on $\widehat{R}_{\epsilon u}[\tau]$ by checking if $|\widehat{R}_{\epsilon u}[\tau]| \leq 1.96\sqrt{P_r/N}$ for all τ.
- If $\widehat{R}_{\epsilon u}[\tau]$ is outside these bounds, then for those values of τ, $\epsilon[k]$ and $u[k-\tau]$ are probably dependent.
- Dependency for small τ could imply the need for smaller n_k.

OTHER TESTS: MATLAB system ID toolbox has validation functions.

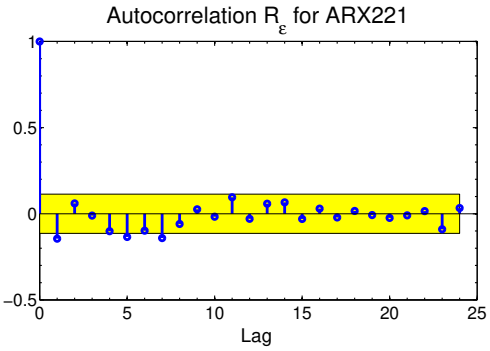
- `compare.m` - compare model's output with actual output.
- `sim.m` - simulate a model (old function = `idsim.m`).
- `pe.m` - compute prediction errors (the longer the prediction horizon, the more demanding the modeling task).
- `predict.m` - predict future outputs.
- `resid.m` - compute and test residuals.

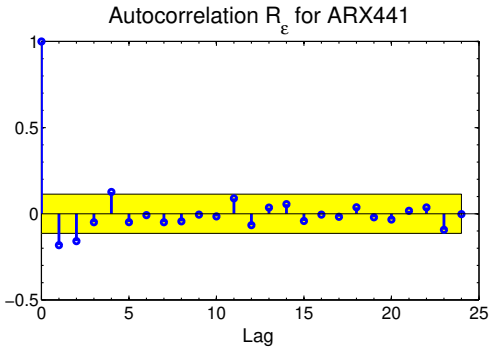
- Try at least one of these, never using same data to create and validate a model.
  - A larger model will always give a better fit (a lower value of $V_N(\theta)$).
  - Must use new data to compare. Good models will still give good predictions on the new data as well.
- Problem with using same data for model creation and validation is that model will attempt to fit the randomness of the noise.
  - Since noise varies from run to run, this is counterproductive.
  - If only one data set is available, split in half: one half for training, one for validation.

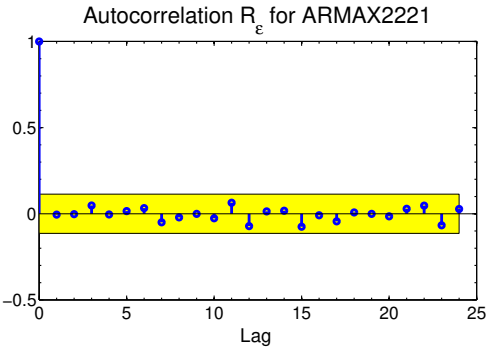
EXAMPLE: Calculated residuals for the previous system ID example.

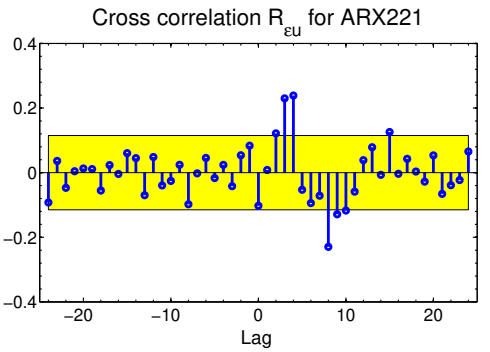
```
% Create a new figure; call "resid" with no outputs to plot residuals
figure; resid(M_arx221,Z);

% Do some fancy MATLAB handle graphics calls to relabel axes
h = get(gcf,'children'); xlabel(h(1),'Lag'); xlabel(h(2),'Lag');
title(h(1),'Cross correlation R_{\epsilonu} for ARX221');
title(h(2),'Autocorrelation R_{\epsilon} for ARX221');

% For the following, I have omitted the axes-relabeling code
```

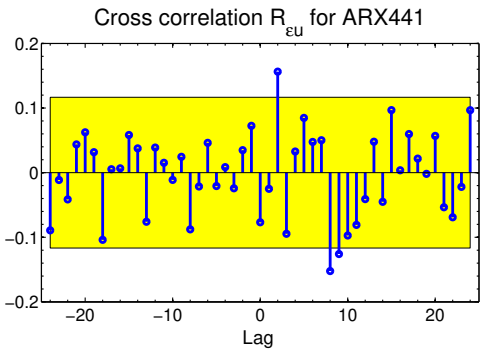
```
figure; resid(M_arx441,Z); figure; resid(M_armax,Z);
figure; resid(M_bj,Z);     figure; resid(M_oe,Z);
```

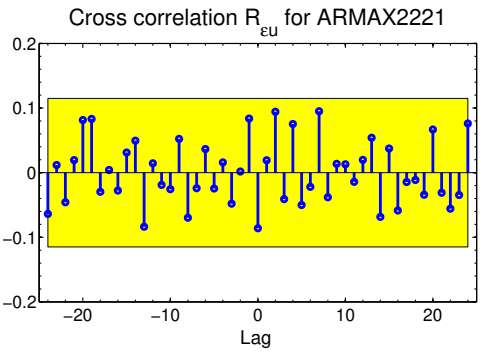




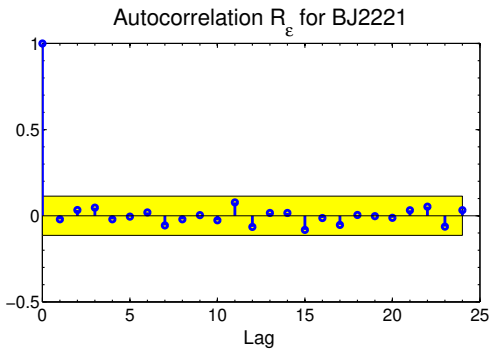


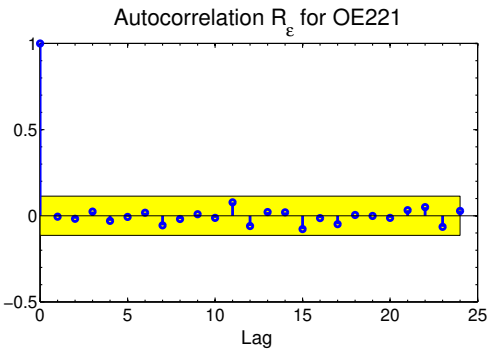


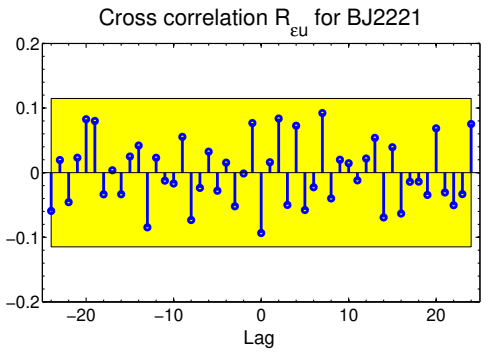


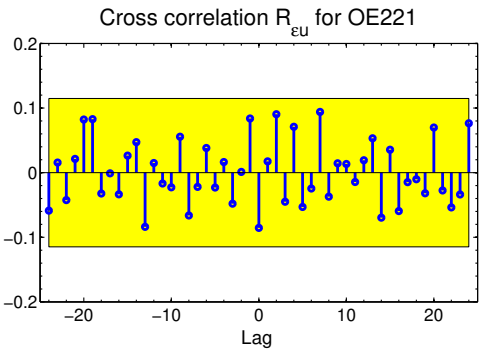


- Both ARX cases fail the auto-correlation and crosscorrelation tests.
- ARMAX, BJ and OE pass. Big improvement over ARX.









4.14: Toolbox methods (4): ARX model size

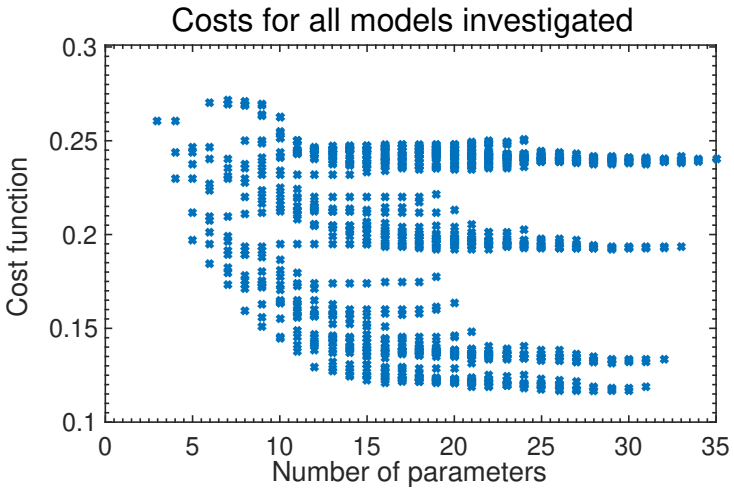
- MATLAB toolbox also has routines to help determine a good model size for the ARX structure.
- There is a tradeoff between model fidelity and accuracy, so researchers have developed modified cost functions of the form $J(\theta, d) = V_N(\theta)(1 + U_N(d))$.
  - $V_N(\theta)$ is the standard cost: decreases with increase in model size.
  - $U_N(d)$ provides a measure of the complexity of the model: increases with increase in model size.
- Two common criteria for $U_N(d)$ are (where $d = \dim(\theta)$):
  - Akaike information criterion (AIC): $U_N(d) = 2d/N$.
  - Minimum description length (MDL): $U_N(d) = d\log(N)/N$.
- Both have strong information-theoretic background, which we won't discuss here. They generally give (somewhat) different "optimum" answers, so the "best" model turns out to be somewhat subjective.
- The objective now is to minimize J over all available d and θ.
  - This is a hybrid optimization problem, since d is integer, and θ is a vector of real numbers.
  - To make tractable, must first select a set of candidate model structures, for which d is known.
  - Then, for each structure, find the optimum $V_N(\theta)$.
    - Note that $U_N(d)$ will be a constant in this optimization, so it does not play a direct role.
    - Can use any optimization method from before without change.

- For each d, select the model structure having the lowest $V_N(\theta)$.
- Finally, plot $J(\theta, d)$; select model with overall lowest value.

- Note that this can require *lots* of optimization time.
  - Most facile for ARX structure due to speed of ARX parameter optimization, so MATLAB has this built in (see example below).
    - Guidance: If validation set `Z_val` is different from training set `Z`, then choose "best fit" structure. Otherwise, choose either MDL or AIC structure.
  - However, can also be done (manually) for any set of structures, where you compute `V` using your own methods, then use `selstruc(V)` for final selection.

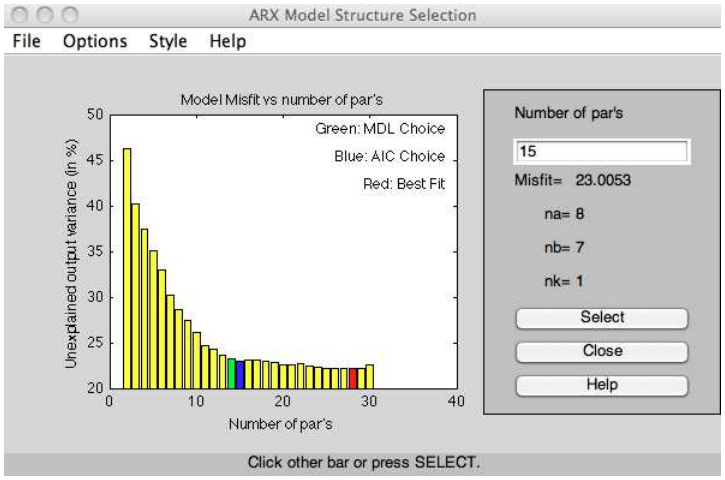
EXAMPLE: In prior example, we found that ARX221 and ARX441 did not do very good job of modeling system.

- Probably need a larger model: Typical problem with ARX.
- Optimize ARX models where:
  - Numerator: $n_b \in \{1 \ldots 15\}$,
  - Denominator: $n_a \in \{1 \ldots 15\}$,
  - Delays: $n_k \in \{1 \ldots 5\}$.
- Plot of $V_N(\theta)$ versus number of parameters.

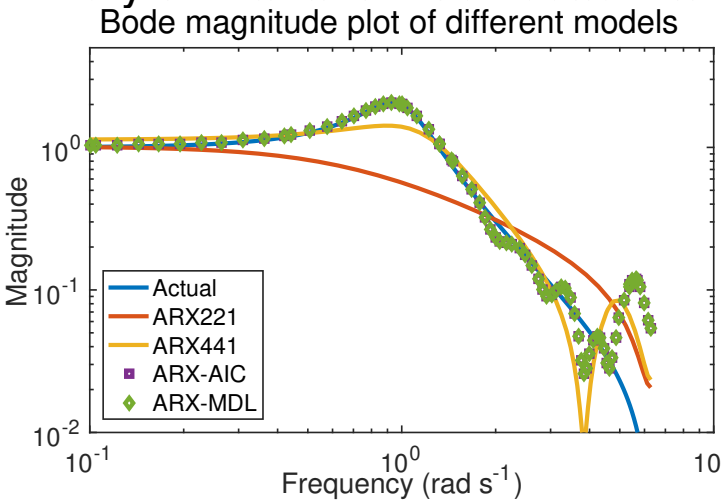


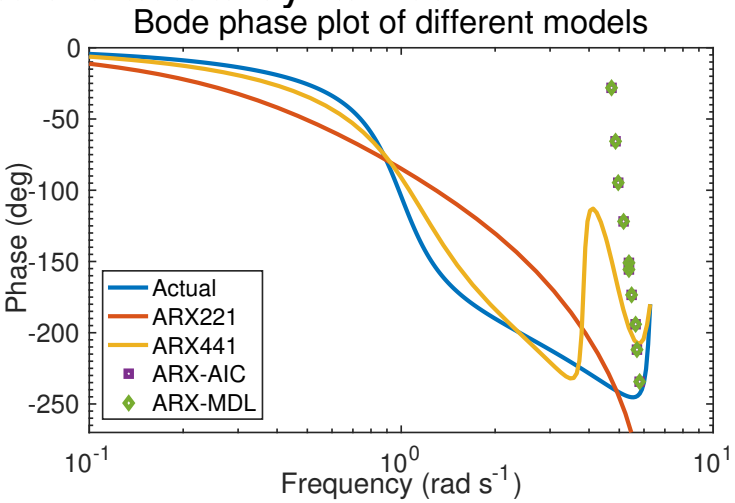

```
NN = struc(1:15,1:15,1:5); % define range of structures
V = arxstruc(Z,Z_val,NN);  % compute cost for each one
plot(sum(V(2:4,1:end-1)),V(1,1:end-1),'x');
xlabel('Number of parameters'); ylabel('Cost function');
title('Costs for all models investigated');
```

- See right away that there are many poor choices! With only five parameters, would get very poor fit.
- Can also use `selstruc` GUI to manually select best model for each overall size.
  - For MDL, fifteen parameters: 8,6,1;
  - For AIC, sixteen parameters: 8,7,1.

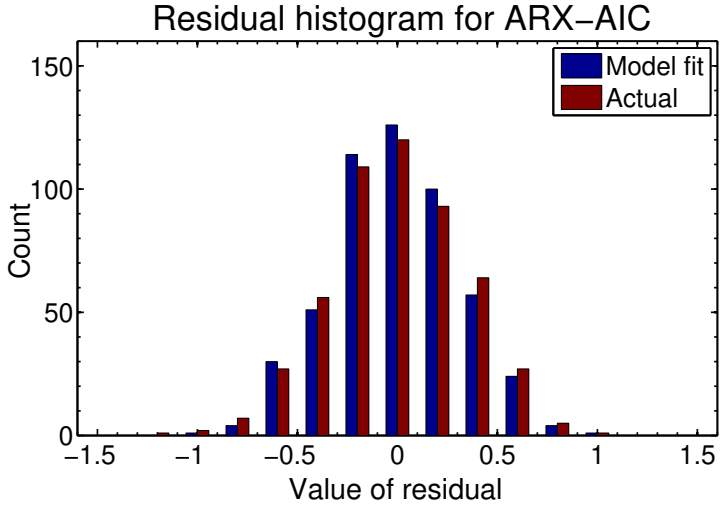


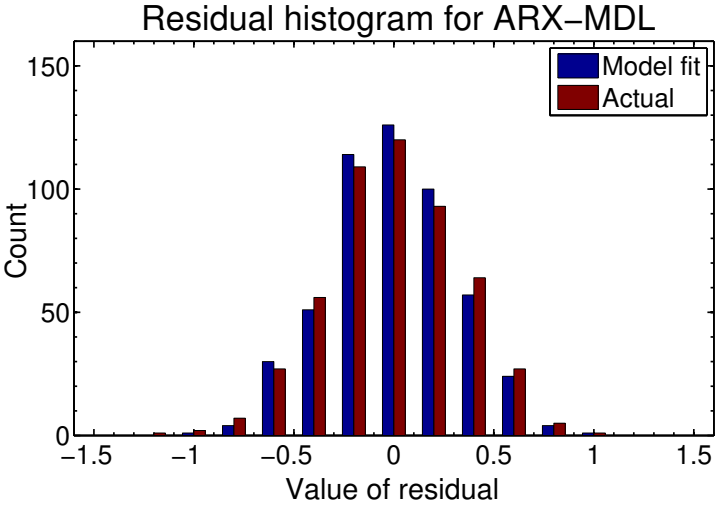
- Frequency response plots show that ARX-AIC and ARX-MDL are very similar and much better fits to the actual dynamics.



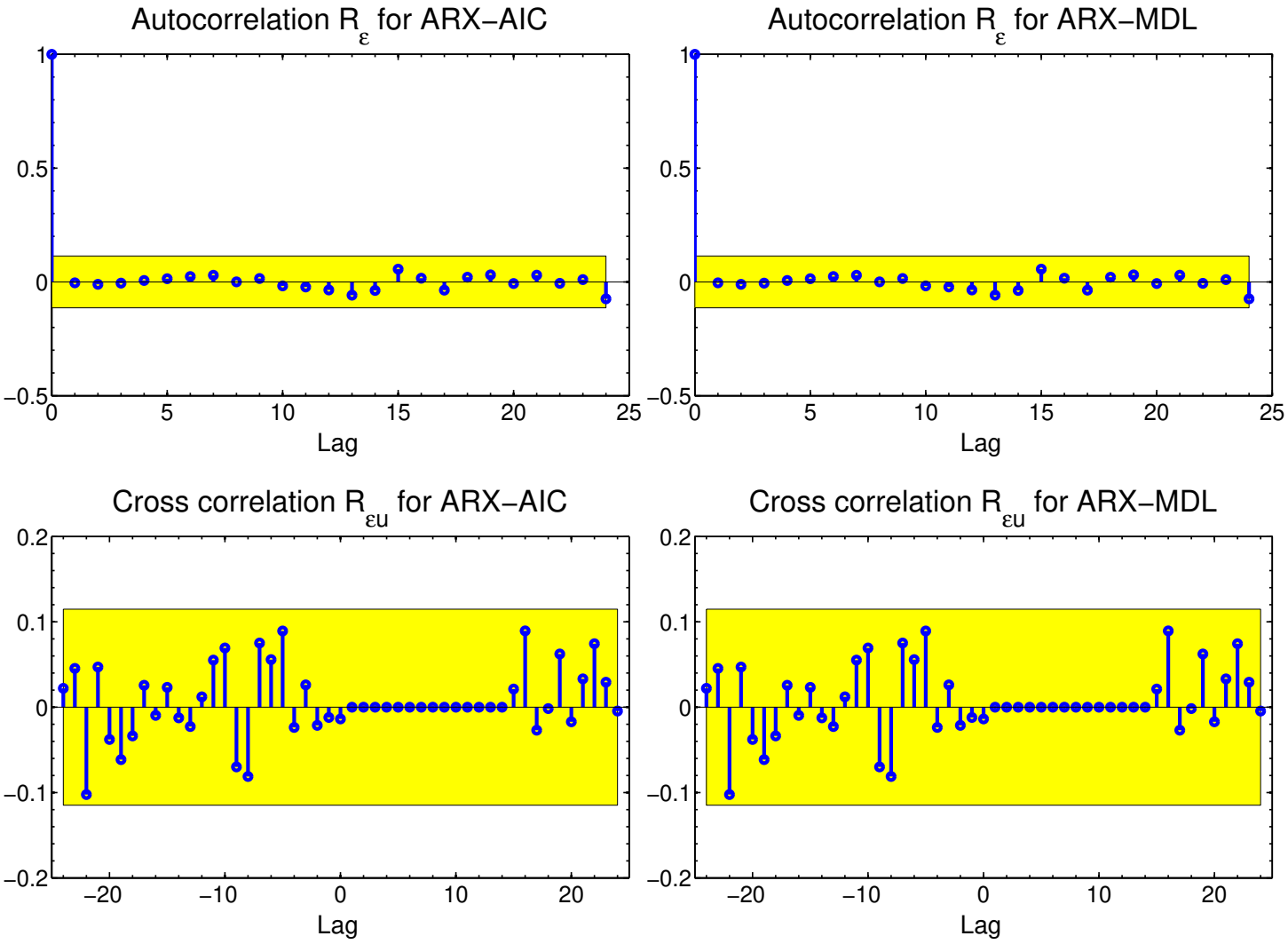


- Histograms look much better than ARX221 and ARX441.





- Both pass the correlation tests with flying colors.



- Note: This approach automatically takes care of the delay estimation and model-order estimation problems that we discussed earlier.
- So, these give much better models, but much larger than what we have used for the other approaches.
  - Very typical problem: ARX is not the ideal structure.
  - So, will generally be better off trying this approach with multiple non-ARX model structures.

4.15: Example with nonwhite noise

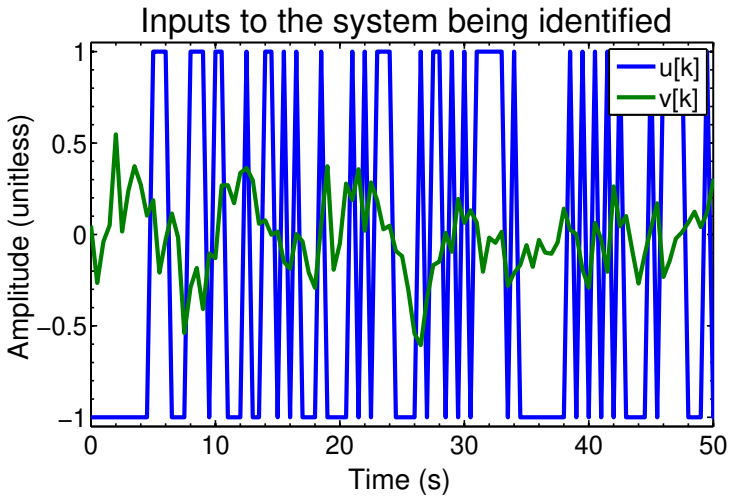
- Previous examples had white Gaussian noise, added directly in OE form. Will now try something harder.
- For valid comparison, try same experiment, but with nonwhite noise.
- Try more complicated scenario with

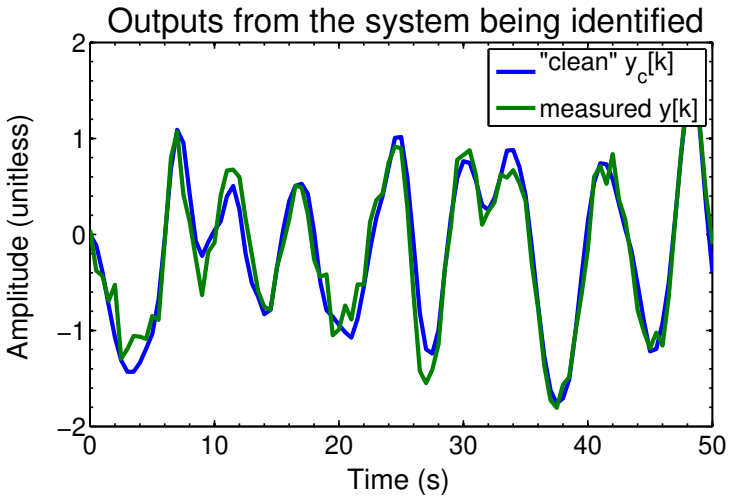
$$H(q) = \frac{1 - 1.633q^{-1} + 0.7567q^{-2}}{1 - 1.8623q^{-1} + 0.9851q^{-2}}.$$

  - Introduces broadband noise plus narrowband noise at 0.7 rad s^{-1}.
- Now must match both $G(q)$ and $H(q)$ to do a good job.
  - $H(q)$ and $G(q)$ have a different denominator, so expect both ARX and ARMAX to struggle.
  - Also $|H(q)| \neq 1$ so OE should also be poor.
- Noise $v[k]$ scaled to be 1/4 input signal's power, as before.

```
nne=[1 .5 .5]; dde=[1 .03 .5];% cts H dynamics
[nume,dene] = c2dm(nne,dde,T);%ZOH conversion is default for TF's
nume = nume/nume(1);        % scaling to make the polynomial monic
u = idinput(Npts,'prbs');   % input signal for identification
yc = dlsim(num,den,u);      % compute "clean" output of G
e = randn(Npts,1);          % dist
v = dlsim(nume,dene,e);     % filtered dist
LL = 0.25*(yc'*yc)/(v'*v); % scale so energy in sensor noise 1/4 times
e = sqrt(LL)*e;             % scale both
v = sqrt(LL)*v;             % the energy in the "clean" signal
y = yc+v;                   % actual output y=Gu+v
Z = iddata(y,u,T);          % data available to identification
% similar process for validation set
```

- Correlation in noise not necessarily obvious.

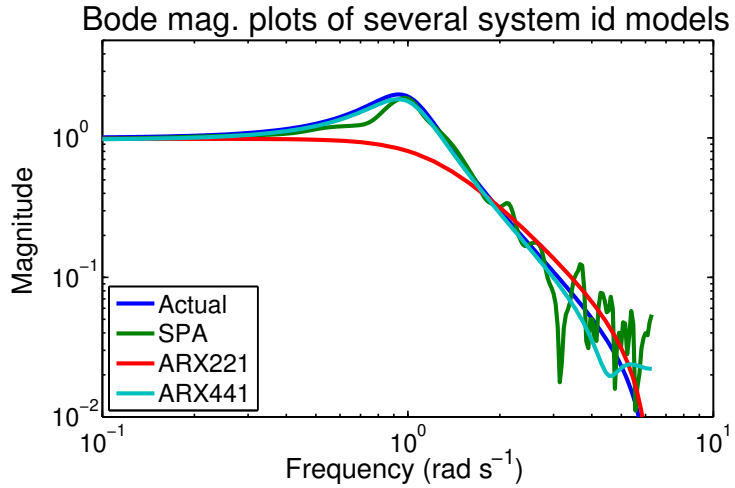


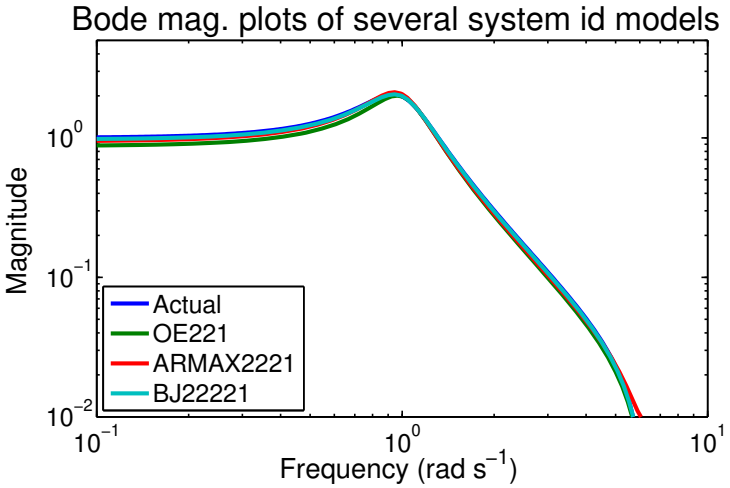


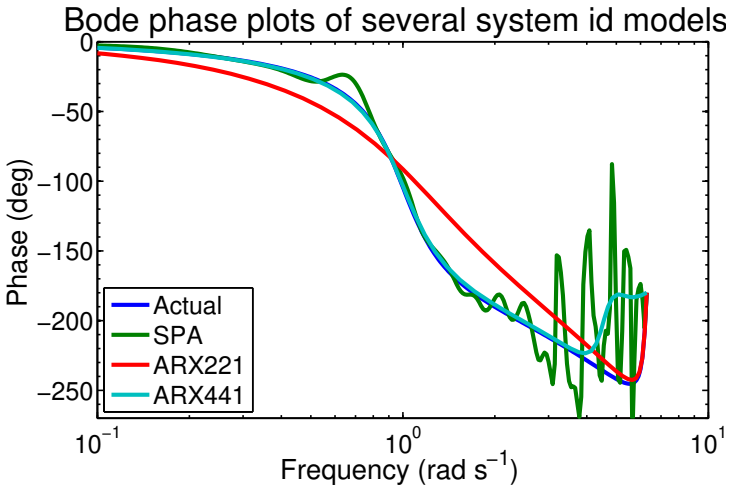
- Some numerical results for the transfer-function coefficients of $G(q)$ are tabulated below: ARMAX, OE good; BJ better.

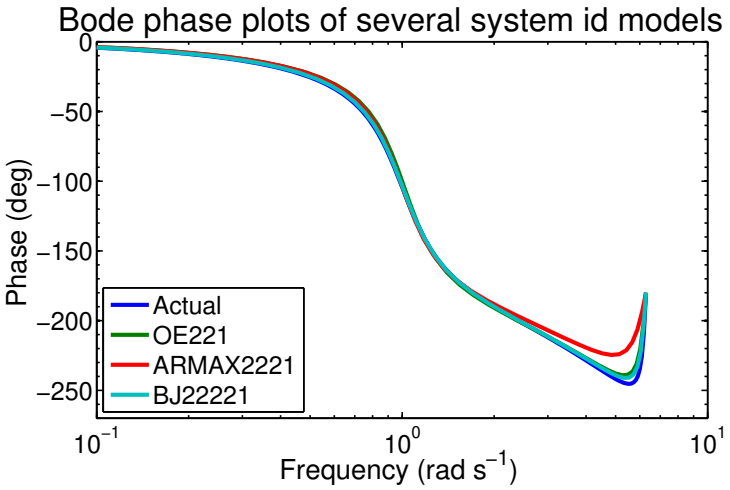
	b_0	b_1	b_2	a_0	a_1	a_2
Actual	0	0.1129	0.1038	1	−1.562	0.7788
ARX221	0	0.1200	0.1106	1	−1.199	0.4338
ARMAX2221	0	0.1181	0.0868	1	−1.581	0.7984
BJ22221	0	0.1125	0.1002	1	−1.566	0.7841
OE221	0	0.1053	0.0920	1	−1.573	0.7989

- ARX441 gives reasonable fit to $G(q)$, but BJ, ARMAX, OE look better.

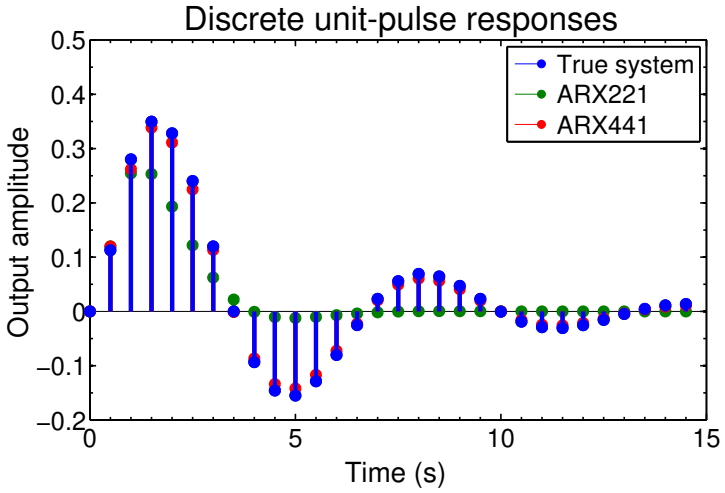


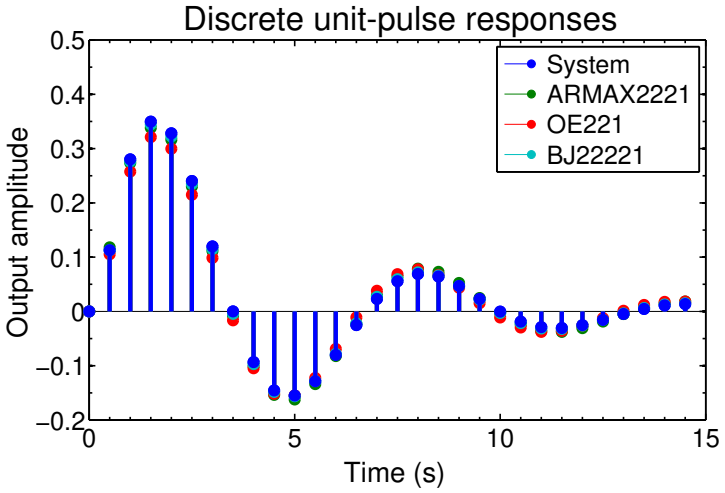






- ARX441 gives reasonable impulse response; BJ, ARMAX, OE better.

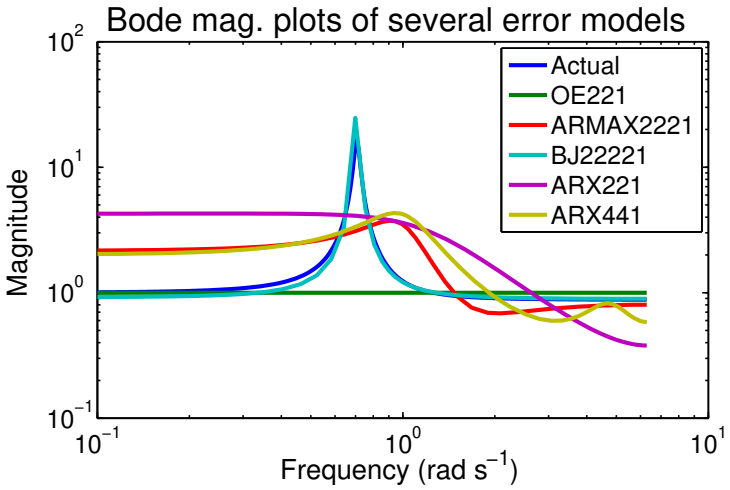


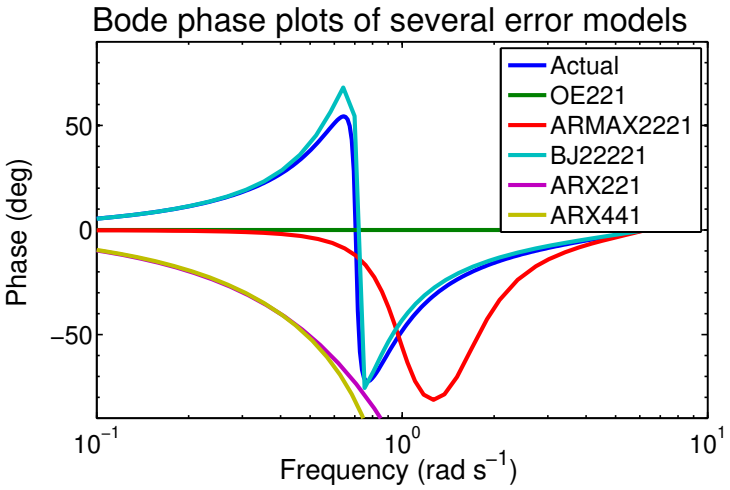


- Some numerical results for the transfer-function coefficients of $H(q)$ are tabulated below: ARMAX and BJ only ones even close.

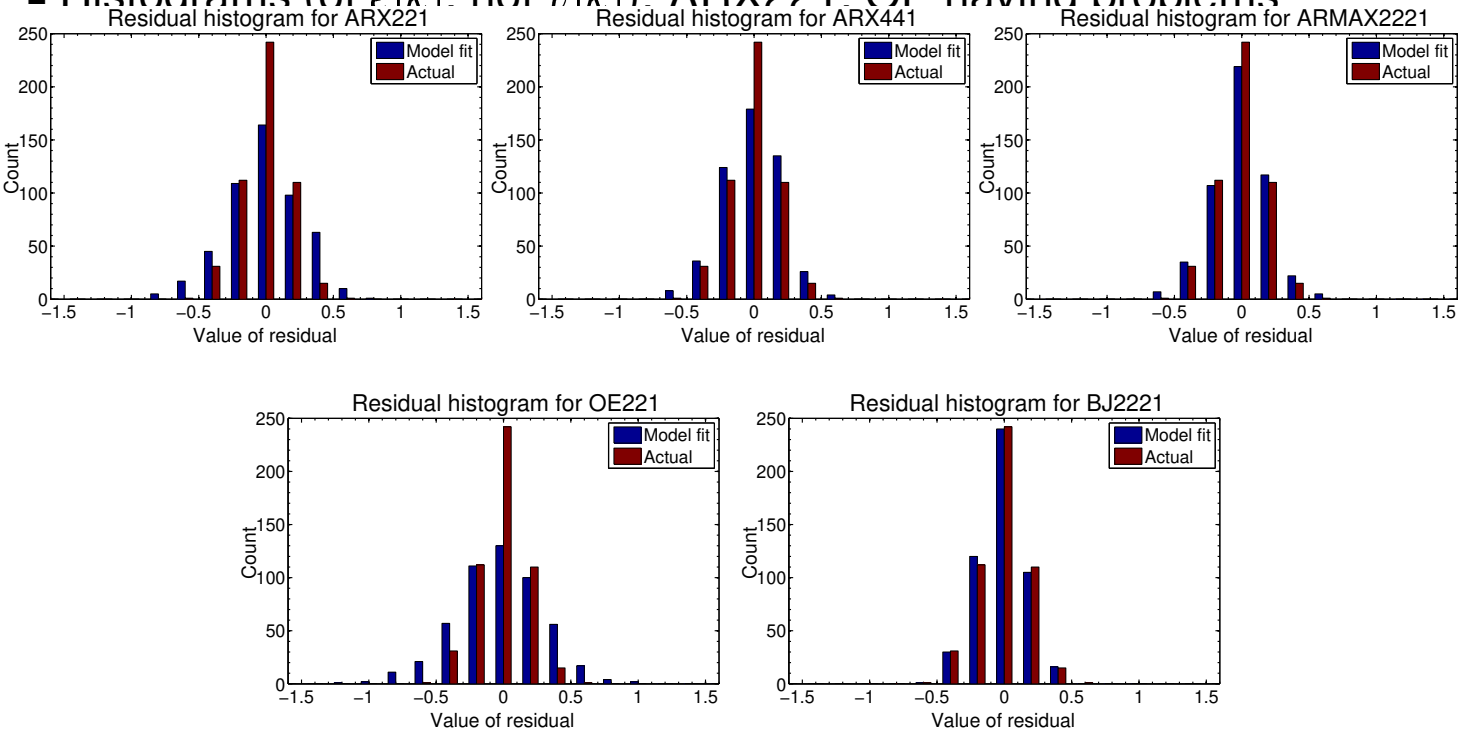
	c_0	c_1	c_2	d_0	d_1	d_2
Actual	1	-1.634	0.7567	1	-1.862	0.9851
ARX221	1	0	0	1	-1.199	0.4338
ARMAX2221	1	-1.122	0.5905	1	-1.581	0.7984
BJ22221	1	-1.673	0.7865	1	-1.872	0.9955
OE221	1	0	0	1	0	0

- Bode plots of $\widehat{H}(q)$ also show BJ to be far superior to other methods.





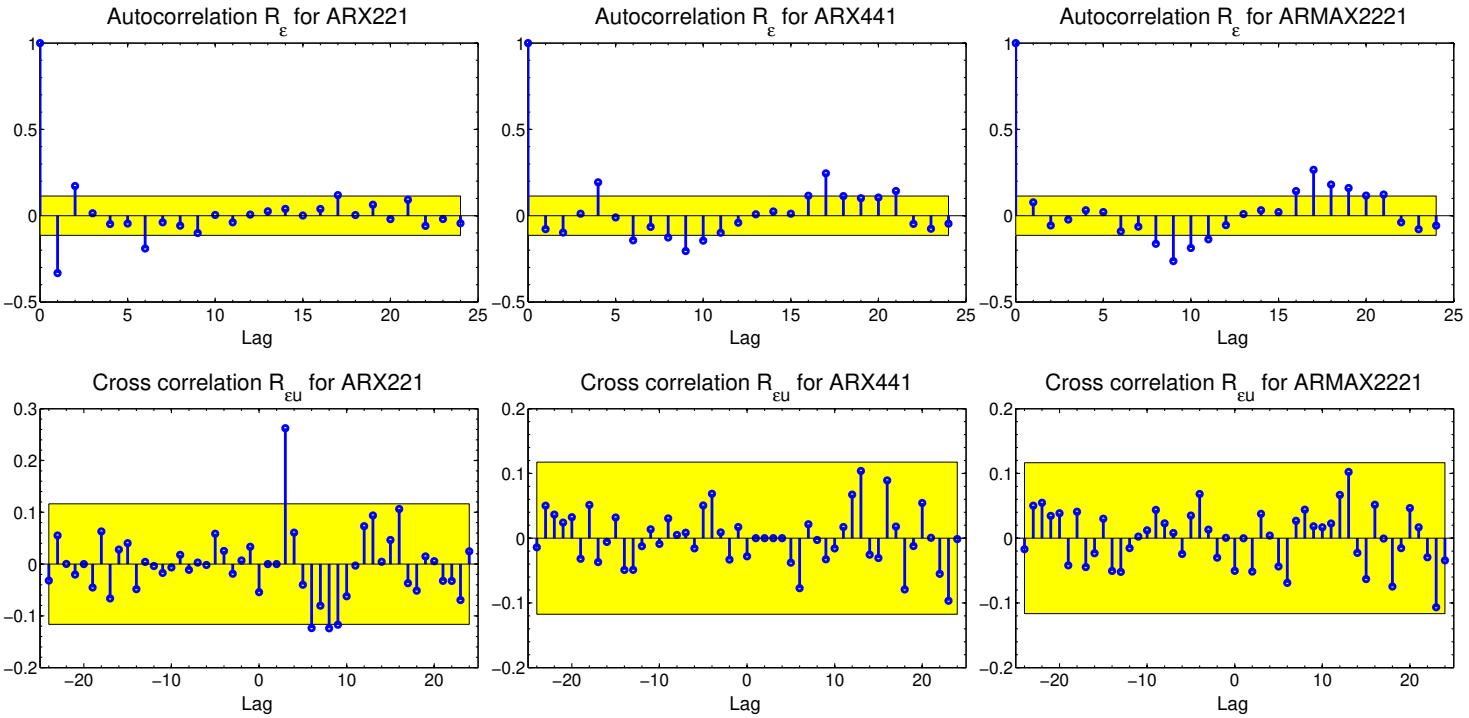
- Histograms (of $e[k]$, not $v[k]$): ARX221, OE having problems

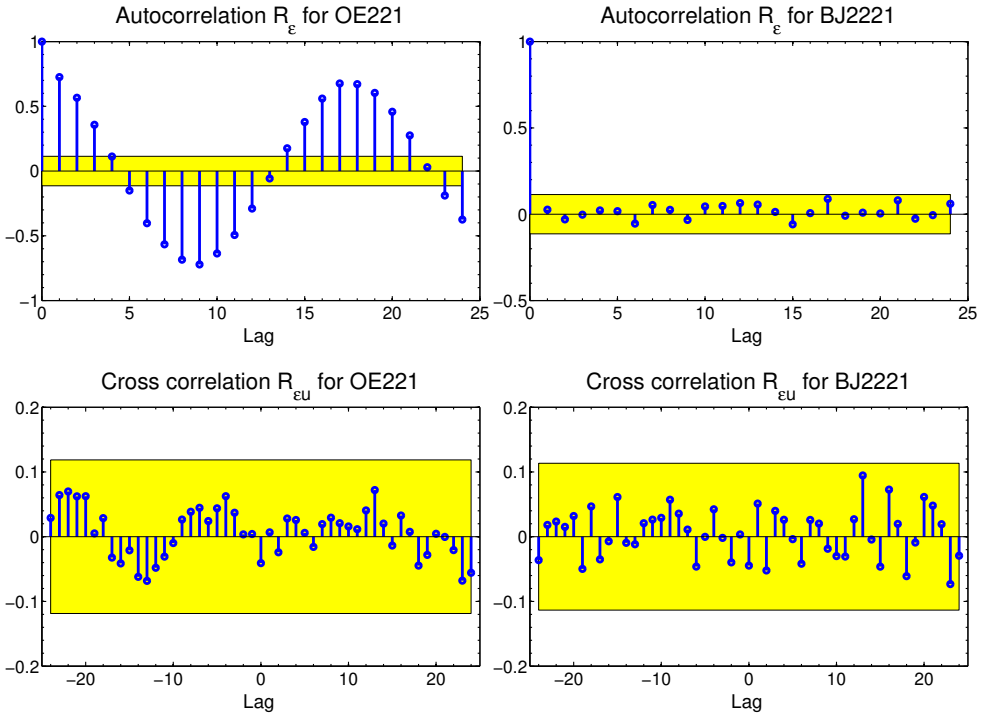


- Results for noise mean and noise-power estimation of $e[k]$ (not $v[k]$)

	Actual	ARX221	ARMAX	BJ2221	OE221
Mean	-0.0117	-0.0026	-0.0071	-0.0155	-0.0197
σ^2	0.0310	0.0750	0.0418	0.0302	0.1038

- Residuals: ARX221, ARX441, ARMAX, OE(!) fail. BJ passes well.





4.16: Model quality: Bias error

- What do we mean by a good model?
  - Close to the "true description," but that usually doesn't exist.
- Factors to keep in mind:
  - "Model quality" depends on intended use: simulation model, or prediction model for control design.
- Model quality associated with "model stability"
  - Does it change much as we look at different segments of the data?
- Need to quantify these model errors: bias and variance.
- Want a model with low bias and low variance!
- All model types have bias/variance problems, but ARX easiest (only?) case to analyze.
  - Also most pronounced.

Types of model errors

1. Variance errors:
   - Repeat experiment with same deterministic input $u[k]$ but different disturbance sequence $v[k]$.
   - Solve for model parameters: will get different values each time because the noise changes.
   - Variance errors can typically be reduced by collecting more data (want $N \to \infty$).
2. Bias errors:

- Systematic errors that occur because the model chosen is not capable of describing the system dynamics.
- If we repeat the experiment, will get same (expected) values each time because of deficiency of model structure.
  - Even with no noise.
  - Model structure too small or wrong type.

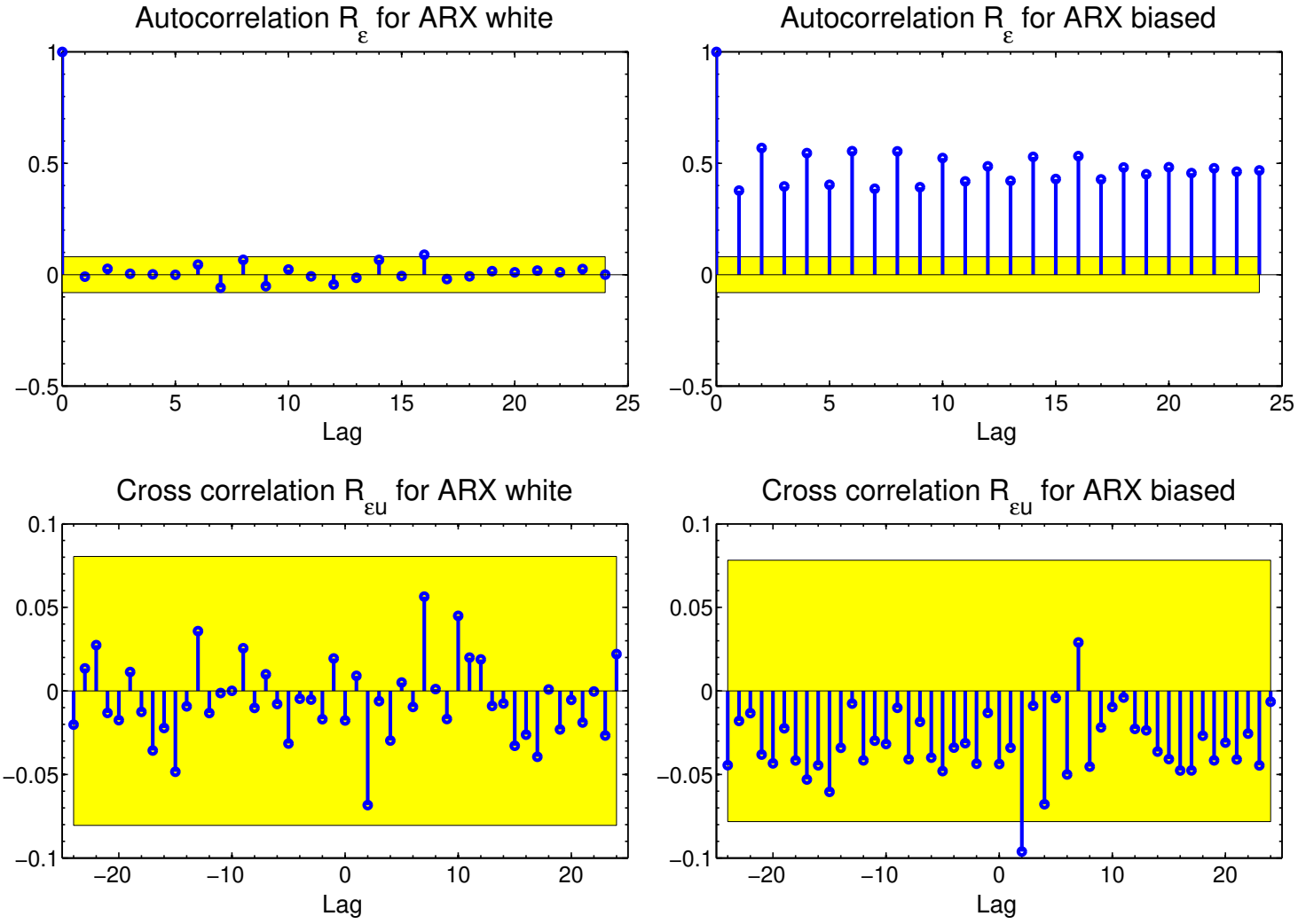
Typical problems

- Consider what happens when the linear regression analysis has a non-zero mean plus white noise for $e[k]$.
- Turns out that ARX performance is very sensitive to this type of "coloring" in the noise (bias).

SIMPLE EXAMPLE: Let $y[k] + ay[k-1] = bu[k-1] + e[k]$.

- Let the input be white Gaussian noise: $u[k] \sim \mathcal{N}(0,1)$.
- Consider two disturbance cases:
  1. Let the disturbance be white Gaussian noise: $e[k] \sim \mathcal{N}(0,1)$.
  2. Let the disturbance have an unknown bias: $e[k] \sim \mathcal{N}(1,1)$.
- We use $a = 0.9$, $b = 0.25$, and $N = 1024$.
- After averaging over ten different input-output sequences,
  1. Case 1 results: $\hat{a} = 0.9028$, $\hat{b} = 0.2563$.
  2. Case 2 results: $\hat{a} = 0.8128$, $\hat{b} = 0.2652$.
     - ARX performance with the non-white noise is quite poor (even though the "true" system is ARX).

- Residual analysis shows us that there is a real problem here.

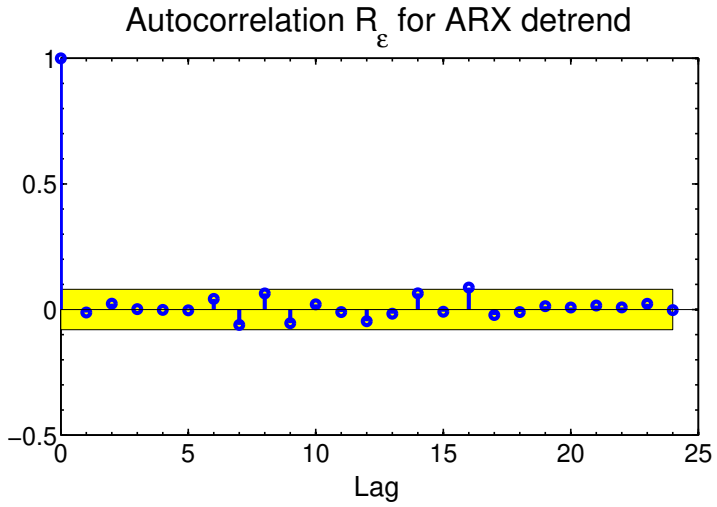


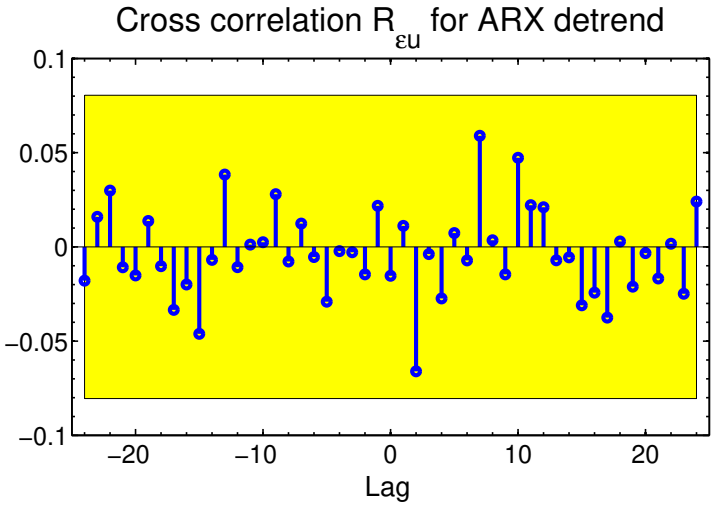
- Two possible solutions: de-trend or estimate the offset.
- `detrend(x,'constant')` is an *ad hoc* approach that works well.
  - Work with modified model:

$$A(q)(y[k]-\bar{y}) = B(q)(u[k]-\bar{u}) + \tilde{e}[k].$$

  - Use the modified data $(u[k]-\bar{u}, y[k]-\bar{y})$ to estimate the parameters in $A(q)$ and $B(q)$.

- For our example, case 2, we get $\hat{a} = 0.9029$ and $\hat{b} = 0.2565$ using de-trended data. Very similar to the results for case 1.
  - You should nearly always detrend $y[k]$ and $u[k]$ data.
- The second solution is to estimate the offset.
- Write $e[k] = e_0 + w[k]$, where e_0 is a constant offset and $w[k]$ is zero-mean white noise.
- Modify the estimation scheme so that we can estimate e_0 also.





$$y[k] = -ay[k-1] + bu[k-1] + e_0 + w[k]$$

$$= \begin{bmatrix} -y[k-1] & u[k-1] & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ e_0 \end{bmatrix} + w[k].$$

- This is a standard form, but now $\theta = \begin{bmatrix} a & b & e_0 \end{bmatrix}^T$.
- For the example given above, I solved the least-squares problem with handwritten code, averaged over ten runs, and got $\hat{a} = 0.9029$, $\hat{b} = 0.2565$, and $\hat{e}_0 = 1.0005$.

4.17: Bias problems in ARX WLS solution

- Consider the least-squares ARX optimization in Topic 4.8.
- Wrote prediction model as $\hat{Y} = X\theta$.
- The weighted cost function was $V_N = (Y - \hat{Y})^T W (Y - \hat{Y})$.
- This gives $\hat{\theta} = (X^T W X)^{-1} X^T W Y$.
- What is the bias in this estimate? Measure this by computing $\mathbb{E}[\theta - \hat{\theta}]$. Want bias to be zero.

CALCULATION: Compute

$$\theta - \hat{\theta} = \theta - (X^T W X)^{-1} X^T W Y.$$

- Assume that Y has the form $Y = X\theta + e$. Then

$$\theta - \hat{\theta} = \theta - (X^T W X)^{-1} X^T W X \theta - \underbrace{(X^T W X)^{-1} X^T W}_{-X_W^L} e$$

$$= -X_W^{-L} e,$$

so

$$\mathbb{E}[\theta - \hat{\theta}] = -\mathbb{E}[X_W^{-L} e].$$

- If the matrices X and W are deterministic and known, then $\mathbb{E}[\theta - \hat{\theta}] = -X_W^{-L} \mathbb{E}[e]$.
  - So, WLSE is unbiased for zero-mean disturbances in this case.
- Big problem: For the WLSE we care about in the ARX case, $X = \left[\begin{array}{ccc} \phi[1] & \cdots & \phi[N] \end{array} \right]^T$ so is explicitly a function of measured data.
- So, we cannot pull out the X_W^{-L} term. We are stuck with the bias.

$$\mathbb{E}[\theta - \hat{\theta}] = -\mathbb{E}[(X^T W X)^{-1} X^T W e] \neq 0 \quad \text{in general.}$$

- One last exception: If $e[k]$ is white, it will not be correlated with past data in X.
  - Therefore, $\mathbb{E}[\theta - \hat{\theta}] = -\mathbb{E}[X_W^{-L}]\mathbb{E}[e] = 0$.
- In general, WLSE tends to be biased, but the bias tends to be quite small with high SNRs.

EXAMPLE: Another example of problems with bias in parameter estimates.

- Actual system has $A(q)y[k] = B(q)u[k] + C(q)e[k]$, with
  - $A(q) = 1 + a_0 q^{-1}$, $B(q) = b_0 q^{-1}$, and $C(q) = 1 + c_0 q^{-1}$.
  - $u[k]$ and $e[k]$ independent white noises.
- Model this with ARX form

$$\begin{aligned}\hat{y}[k;\theta] &= -ay[k-1] + bu[k-1] \\ &= \theta^T \phi[k],\end{aligned}$$

  which ignores the dynamics in the error input (assumes $c_0 = 0$).
- Now, compute the "prediction error variance." Since
  - $V_N = \dfrac{1}{N}\displaystyle\sum_{k=1}^{N} \epsilon^2[k;\theta]$, which is approximately $\mathbb{E}[\epsilon^2[k;\theta]]$ for large N,
  - As $N \to \infty$, V_N is a good estimate of the prediction-error variance.
  - That is, as $N \to \infty$, we can work with either V_N or $\overline{V} = \mathbb{E}[\epsilon^2]$.
  - The latter is directly computable, which allows for interesting analytical comparison.

- New cost

$$\begin{aligned}\overline{V} = \mathbb{E}[\epsilon^2] &= \mathbb{E}[\,(y[k] - \hat{y}[k;\theta])^2\,] \\ &= \mathbb{E}[\,(y[k] + ay[k-1] - bu[k-1])^2\,] \\ &\vdots \quad \text{(see appendix)} \\ &= r_0(1 + a^2 - 2aa_0) + b^2 - 2bb_0 + 2ac_0,\end{aligned}$$

  with $r_0 = \mathbb{E}[y^2[k]] = \left(b_0^2 + c_0^2 - 2a_0c_0 + 1\right)\left(1 - a_0^2\right)^{-1}$.

- To optimize, $\theta^* = \arg\min_\theta \overline{V}$,

$$\begin{aligned}\frac{\partial \overline{V}}{\partial a} &= r_0\,(2a - 2a_0) + 2c_0 = 0 \\ &\Rightarrow a^* = a_0 - c_0/r_0,\end{aligned}$$

  where $c_0/r_0 \approx 1/\text{SNR}$. Also

$$\begin{aligned}\frac{\partial \overline{V}}{\partial b} &= 2b - 2b_0 = 0 \\ &\Rightarrow b^* = b_0.\end{aligned}$$

- The costs are

$$\begin{aligned}\overline{V}(\theta^*) &= 1 + c_0^2(1 - 1/r_0) \\ \overline{V}(\theta_0) &= 1 + c_0^2,\end{aligned}$$

  so $\overline{V}(\theta^*) < \overline{V}(\theta_0)$.

- By minimizing V_N for large N we expect our estimates to converge to θ^* because of the lower value of $\overline{V}$.

- But, these are biased since $\theta^* \neq \theta_0$.

- But, for this assumed model class, θ^* gives a better predictor since $V(\theta^*) < V(\theta_0)$.

NUMERICAL EXAMPLE: Confirmation of the above calculations.

- System has $y[k] + 0.9y[k-1] = 0.25u[k-1] + e[k] + 0.7e[k-1]$. So,

$$\theta_0 = \begin{bmatrix} a_0 \\ b_0 \end{bmatrix} = \begin{bmatrix} 0.9 \\ 0.25 \end{bmatrix}.$$

- We can compute $r_0 = 1.54$, so we expect $a^* = a_0 - c_0/r_0 = 0.4453$ and $b^* = b_0$. That is,

$$\theta^* = \begin{bmatrix} a^* \\ b^* \end{bmatrix} = \begin{bmatrix} 0.4453 \\ 0.25 \end{bmatrix}.$$

- Averaged ARX results over ten runs for several values of N.

$$\hat{\theta}_{1024} = \begin{bmatrix} 0.4580 \\ 0.2530 \end{bmatrix}, \quad \hat{\theta}_{4096} = \begin{bmatrix} 0.4501 \\ 0.2508 \end{bmatrix}, \quad \hat{\theta}_{16384} = \begin{bmatrix} 0.4485 \\ 0.2525 \end{bmatrix}.$$

- Note that $\overline{V}(\theta_0) = 1.49$ and $\overline{V}(\theta^*) = 1.17$.
- Algorithm gives us the best possible predictor, but this does not necessarily mean we get a good model.
- Average ten ARMAX models: $\theta^*_{\text{ARMAX}} = \begin{bmatrix} 0.9025 & 0.2520 & 0.7038 \end{bmatrix}^T$, which gives a good fit with low bias.
- Conclusion: If the model type or size is wrong, we will get a bias in the parameters.

Variance errors

- Assume that our estimate has zero bias.
- Can show that, if θ_0 is the value of the actual parameters, then

$$P_N = \mathbb{E}[(\hat{\theta}_N - \theta_0)(\hat{\theta}_N - \theta_0)^T] \approx \frac{\sigma^2}{N}\overline{R}^{-1},$$

where σ^2 is the variance of the added noise $e[k]$, and $\overline{R} = \mathbb{E}[\psi(k,\theta_0)\psi^T(k,\theta_0)]$ where

$$\psi(k,\theta_0) = \left.\frac{\mathrm{d}}{\mathrm{d}\theta}\hat{y}[k,\theta]\right|_{\theta_0}$$

which is the gradient of prediction with respect to θ.

- Covariance decreases with less noise and/or more data.
- Quality of $\hat{\theta}_N$ depends on sensitivity of prediction to θ (ψ).
  - Small sensitivity means that ψ is small, so $\overline{R}$ is small, and $\overline{R}^{-1}$ is large, so variance is large.

- Of course, to be of any use, we need to estimate $\overline{R}$ and σ^2.

$$\hat{R}_N = \frac{1}{N}\sum_{k=1}^{N}\psi(k;\hat{\theta}_N)\psi^T(k;\hat{\theta}_N)$$

$$\hat{\sigma}_N^2 = \frac{1}{N}\sum_{k=1}^{N}\epsilon^2[k;\hat{\theta}_N]$$

$$\hat{P}_N = \frac{1}{N}\hat{\sigma}_N^2\hat{R}_N^{-1}.$$

- For ARX, $\hat{y}[k] = \theta^T\phi[k]$ and $\frac{\mathrm{d}}{\mathrm{d}\theta}\hat{y}[k] = \phi[k]$, so

$$\phi[k] = \begin{bmatrix} -y[k-1] \\ u[k-1] \end{bmatrix}.$$

- This gives $\overline{R} = \mathbb{E}[\psi\psi^T] = \begin{bmatrix} R_y[0] & R_{yu}[0] \\ R_{yu}[0] & R_u[0] \end{bmatrix}$.
- We now see that the selection of $u[k]$ explicitly plays a role in the accuracy of our estimates through R_u and R_{yu}.

- These tools allow us to display confidence bounds for the models we develop.

Where from here?

- Transfer functions great for analysis and design of predictors for and control of LTI SISO systems.
- But, much harder to use for MIMO systems, and much harder to identify transfer-function models for MIMO systems.
- Also, nonlinear optimization required for all but simplest transfer-function approaches, which can get "stuck" in local minima, yielding sub-optimal models.
- "State space" models are an alternate way to describe system dynamics.
  - Work great for multi-input, multi-output systems.
  - Can provide access to what goes on inside a system in addition to an input–output mapping only (however, a system-ID model will not automatically provide this insight).
  - Allow new analysis and synthesis tools for estimation and control that are *very* powerful.
- Furthermore, deterministic globally optimal solutions exist to the system-ID problem. No local minima.
- So, our next topic is a preview of state-space systems (prior exposure helpful, but not necessary).
- From there, we will continue to explore a number of system-ID approaches for state-space models.

Appendix: Variance calculations for bias example

- These calculations are for the example in Topic 4.17.

$$\mathbb{E}[y^2[k]] = \mathbb{E}[\,(-a_0 y[k-1] + b_0 u[k-1] + e[k] + c_0 e[k-1])^2\,]$$

$$(-a_0)^2\mathbb{E}[y^2[k-1]] = a_0^2\mathbb{E}[y^2[k]]$$

$$-2a_0b_0\mathbb{E}[y[k-1]u[k-1]] = 0$$

$$-2a_0\mathbb{E}[y[k-1]e[k]] = 0$$

$$-2a_0c_0\mathbb{E}[y[k-1]e[k-1]] = -2a_0c_0 \times$$

$$\mathbb{E}[\,(-a_0 y[k-2]+b_0 u[k-2]+e[k-1]+c_0 e[k-2])\, e[k-1]]$$

$$= -2a_0c_0$$

$$b_0^2\mathbb{E}[u^2[k-1]] = b_0^2$$

$$\mathbb{E}[u[k-1]e[k]] = \mathbb{E}[u[k-1]e[k-1]] = 0$$

$$\mathbb{E}[e^2[k]] + 2c_0\mathbb{E}[e[k]e[k-1]] + c_0^2\mathbb{E}[e^2[k-1]]$$

$$= 1 + c_0^2$$

$$\text{so, } \mathbb{E}[y^2[k]] = a_0^2\mathbb{E}[y^2[k]] - 2a_0c_0 + b_0^2 + 1 + c_0^2$$

$$= (b_0^2 + 1 + c_0^2 - 2a_0c_0)(1 - a_0^2)^{-1}.$$

- Also

$$\mathbb{E}[(y[k] - \hat{y}[k])^2] = \mathbb{E}[\,(y[k] + ay[k-1] - bu[k-1])^2]$$

$$\mathbb{E}[y^2[k]] = r_0$$

$$2a\mathbb{E}[y[k]y[k-1]] = 2a\mathbb{E}[(-a_0 y[k-1]+b_0 u[k-1]+e[k]+c_0 e[k-1])y[k-1]]$$

$$= 2(-aa_0\mathbb{E}[y^2[k]] + ac_0)^2$$

$$
\begin{aligned}
2\mathbb{E}[y[k](-bu[k-1])] &= -2b\mathbb{E}[(-a_0y[k-1]+b_0u[k-1]+e[k]+c_0e[k-1])u[k-1]] \\
&= -2bb_0 \\
b^2\mathbb{E}[u^2[k-1]] &= b^2 \\
\mathbb{E}[a[y[k-1]bu[k-1]] &= 0 \\
\mathbb{E}[(ay[k-1])^2] &= a^2\mathbb{E}[y^2[k]]
\end{aligned}
$$

so, $\mathbb{E}[(y[k]-\hat{y}[k])^2] = r_0(1+a^2-2aa_0)+b^2+2ac_0-2bb_0$.