XMUT220 Signals and Systems

Demo 5.1: Time and Frequency Characteristics

Section A. Time Domain Analysis of the System

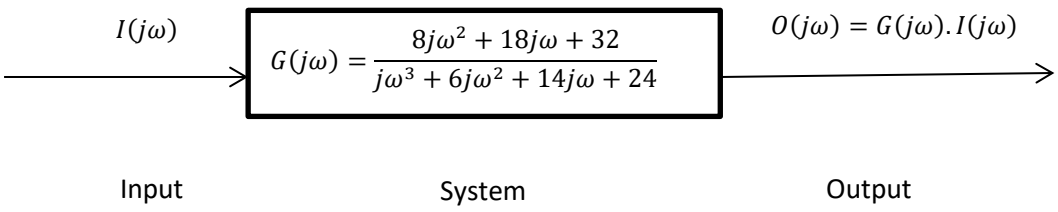
This demo exercise shows how to perform various time-domain analysis of a linear and time invariant systems in the MATLAB. The analysis techniques in the time domain enable you to study the characteristics and behaviours of a given system over time.

To perform the analysis, beside techniques that require observation of the system in real time, test signals such as impulse and step inputs can be injected into the system, so we could evaluate its response. In MATLAB there are various functions available e.g. `impulse()` and `step()` function commands that can realise this type of analysis. You have also function such as `lsim()` that enables you to create arbitrary test signal and also set up specific test conditions for analysing the system.

Notice that MATLAB works with matrices and equations and requires relevant information and data for performing system simulations. In the time and frequency domain analysis it is important that you have the details available before performing the analysis. As a result, some of these preparation works involve formulating the equation or matrix model of the system and getting the relevant operating condition information and data for analysing the system.

Exercise 1 - Time Responses of the System

For illustration purposes, in this exercise we create the time response plot of the following block diagram that illustrates a system with third-order transfer function:



In MATLAB, you type the following code to enter the equation that represents the system to the software and then subsequently assign it to a variable for its easy recall in the subsequent exercises:

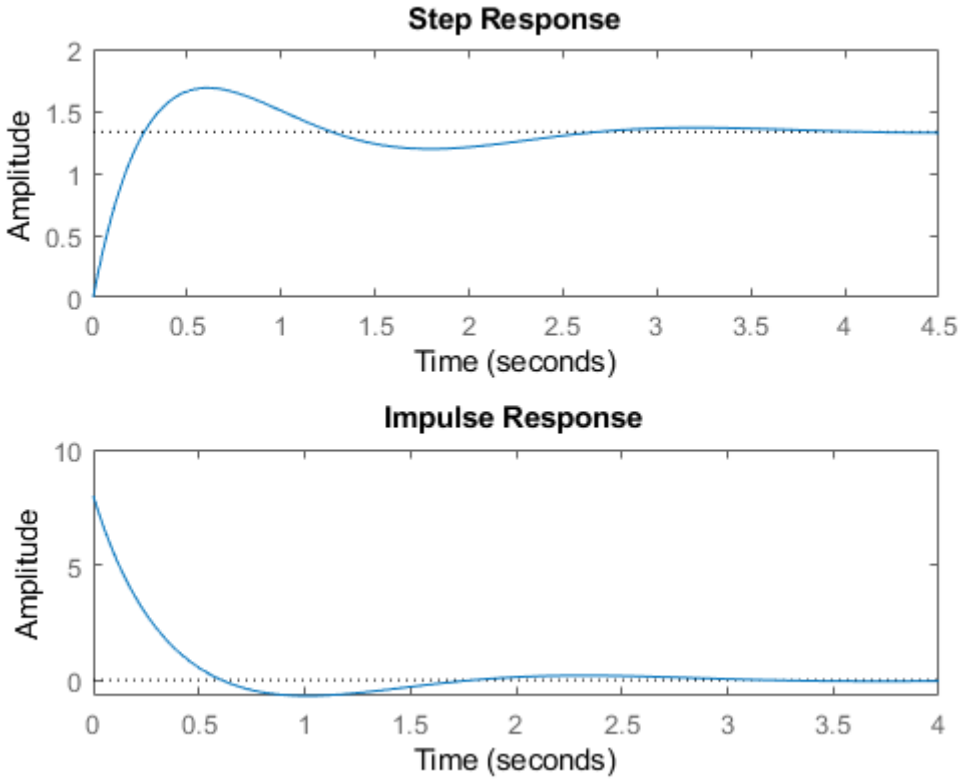
```
sys = tf([8 18 32],[1 6 14 24])
sys =
```

```
     8 s^2 + 18 s + 32
  ----------------------
  s^3 + 6 s^2 + 14 s + 24

Continuous-time transfer function.
```

You can plot the step and impulse responses of this system using `step()` and `impulse()` function commands:

```
subplot(2,1,1)
step(sys)
subplot(2,1,2)
impulse(sys)
```



Observed in the plots above are the time domain plots of the responses of the system when it is subjected to step and impulse signals.

It looks like in the step response plot that the system experiences a little bit oscillatory response to this impulse signal, before it settles down after a few seconds.

In the impulse response plot, the system experiences a reduction of amplitude whenever it is given impulse input. Shown on the plot the reaction of system towards this type of test signal also involves a slightly oscillatory response, but in this case, it is not as significant as in the step response case.

Exercise 2 - Detailed Characteristics of the System Response

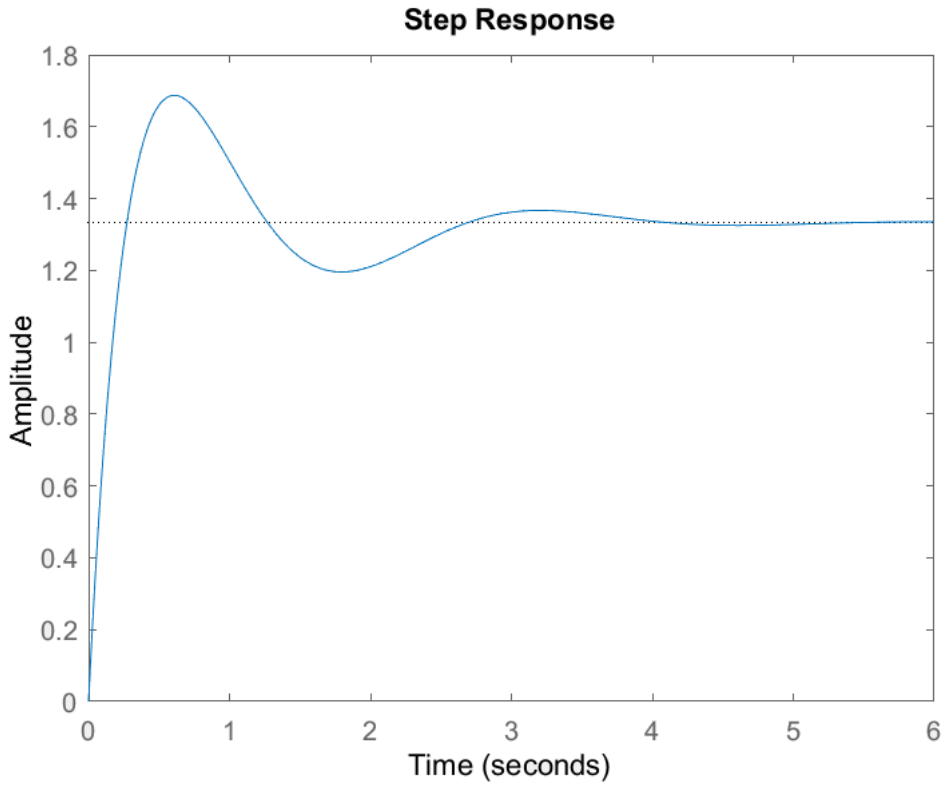
Obtaining the detailed information and data from the plot is very important when you are performing time domain analysis. In a lot of occasions time domain analysis requires retrieving these detailed information and data from the plot i.e. time constant, rise time, settling time, and so forth. You could get these details interactively on the plot in the MATLAB.

Right-clicking on response plots gives access to a variety of options and annotations. In particular, the **Characteristics** menu lets you display standard metrics such as rise time and settling time for step responses, or peak gain and stability margins for frequency response plots.

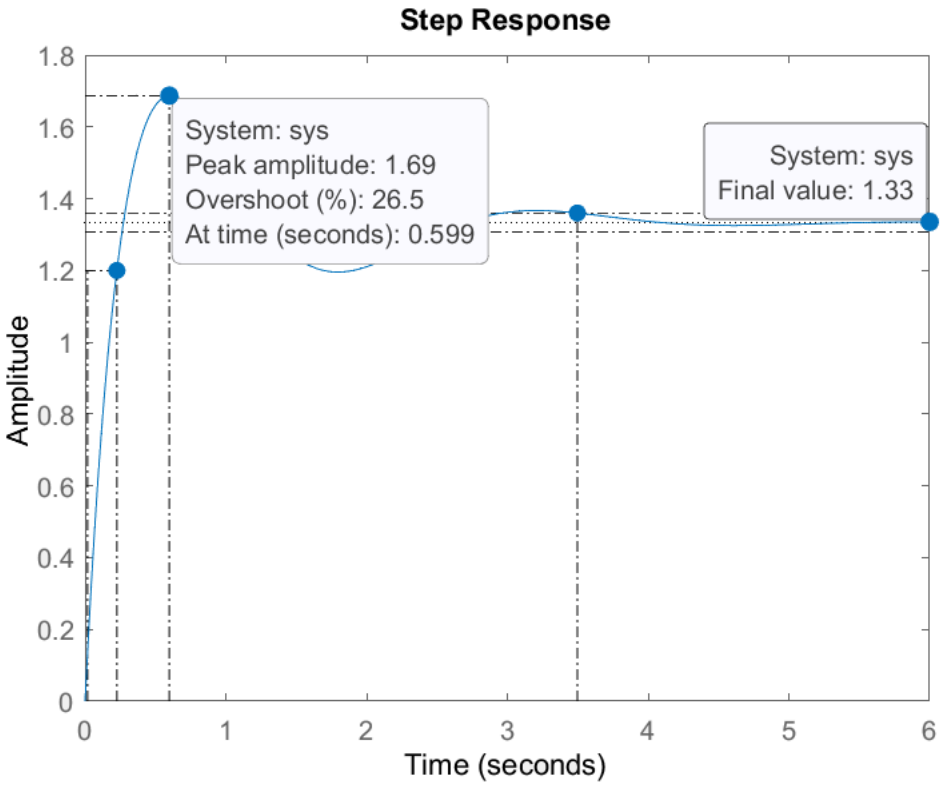
Using the example from the previous section, recall the equation of the system and plot the step response of the system:

```
sys=tf([8 18 32],[1 6 14 24])

step(sys)
```



Now, right-click on the plot to display the Rise Time, Peak Response, and Settling Time Characteristics inside the Characteristics option in the pop-up window, and click on the blue dots to read the corresponding overshoot and settling time values:



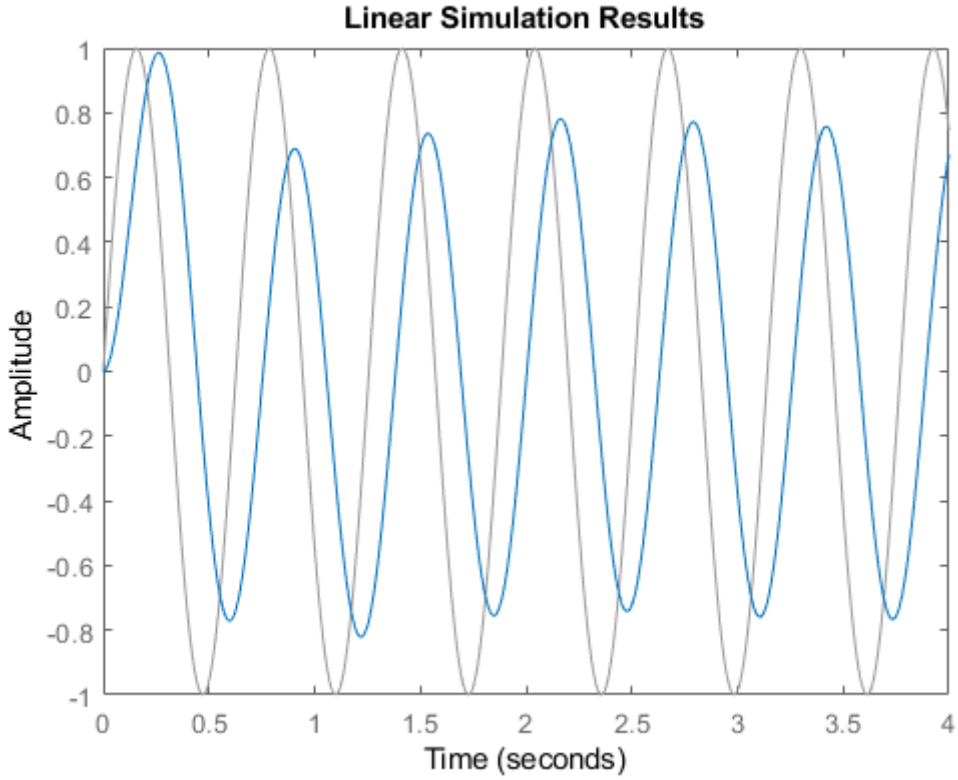
Exercise 3 - Response of Arbitrary Signal and Specific Analysis Conditions

Having the freedom to choose the type of test signals and specific conditions of analysis is very important in time domain analysis. Different systems could potentially require different test signals to evaluate their characteristics and behaviours.

You can simulate the response of a system to an arbitrary signal, for example, a sine wave, using the `lsim()` function command.

```
clf
t = 0:0.01:4;
u = sin(10*t);
lsim(sys,u,t)   % u,t define the input signal
```

Shown on the time domain plot below, the input signal appears in grey colour line and the system's response in blue line. Notice on the plot that the output signal of the system experiences some sort of waveform distortions which are typical responses of a third order system.



In term of phase shift, see the output signal is shifted to the right relative to its input signal. As far as the gain of the system is concerned, the amplitude of the output signal is less than the amplitude of the input signal.

Exercise 4 - Step-Response Details of the System

Step signal is used quite often to test a given system. Beside other test signals such as impulse, ramp and parabola, this test signal is versatile enough to be used for analysing the system in terms of its performance and stability. Step test could be from low level to a higher level or vice versa.

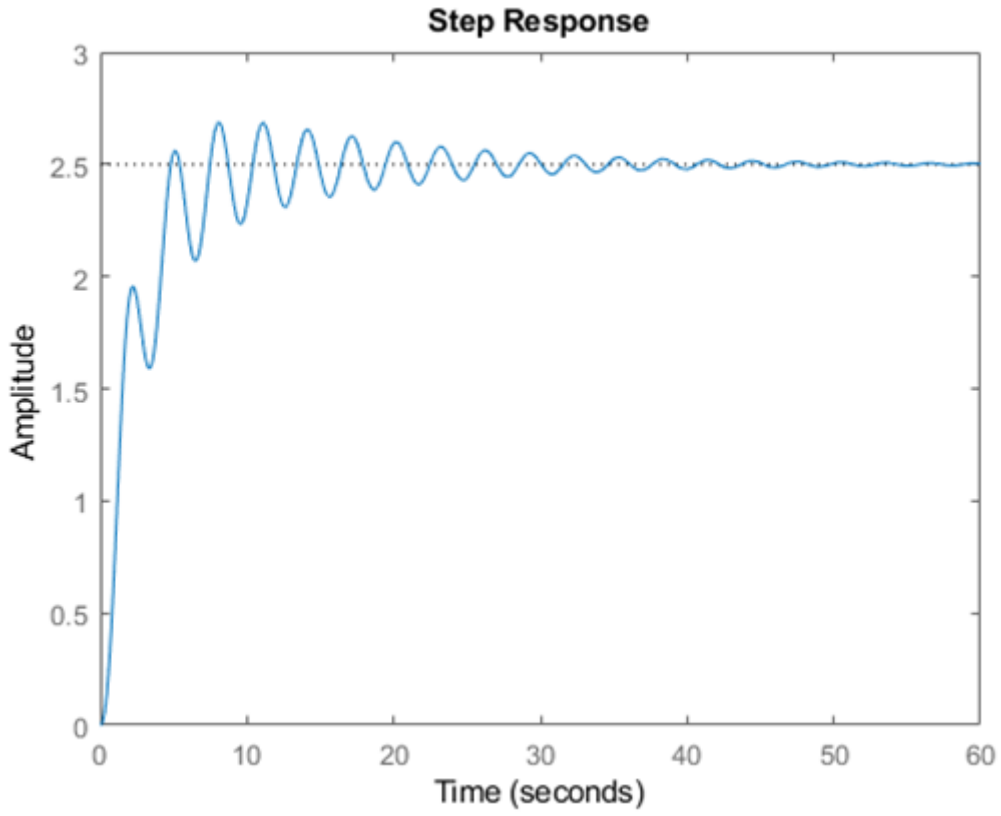
This exercise uses the `stepinfo()` function command in MATLAB. This function helps you to do time domain analysis e.g. rise time, settling time, and other step-response characteristics

Compute step-response characteristics such as rise time, settling time, and overshoot for a dynamic system model. For this example, use the continuous-time transfer function:

$$sys = \frac{s^2 + 5s + 5}{s^4 + 1.65s^3 + 6.5s + 2}$$

Create the transfer function and examine its step response.

```
sys = tf([1 5 5],[1 1.65 5 6.5 2]);
step(sys)
```



The plot shows that the response rises in a few seconds, and then rings down to a steady-state value of about 2.5. Compute the characteristics of this response using `stepinfo()` function command.

```
S = stepinfo(sys)
S = struct with fields:
        RiseTime: 3.8456
    SettlingTime: 27.9762
     SettlingMin: 2.0689
     SettlingMax: 2.6873
       Overshoot: 7.4915
      Undershoot: 0
            Peak: 2.6873
        PeakTime: 8.0530
```

By default, the settling time is the time it takes for $y(t)-y_{final}$ to fall below 2% of its peak value, where $y(t)$is the system response at time *t and* y_{final} is the steady-state response. The result `S.SettlingTime` shows that for `sys`, this condition occurs after about 28 seconds.

The default definition of rise time is the time it takes for the response to go from 10% of its steady-state value to 90% of that value. `S.RiseTime` shows that for `sys`, this rise occurs in less than 4 seconds.

The maximum overshoot is returned in `S.Overshoot`.

For this system, the peak value `S.Peak`, which occurs at the time `S.PeakTime`, overshoots the steady-state value by about 7.5% of the steady-state value.

Section B. Frequency Domain Analysis of the System

Frequency-domain analysis is key to understanding stability and performance properties of a system in the control systems. Bode diagram, Nyquist diagram, and Nichols chart are three standard ways to plot and analyse the frequency response of a linear system. You can create these plots using the `bode()`, `nichols()`, and `nyquist()` function commands respectively in the MATLAB.

In the last exercise in this section we will use `fft()` function in MATLAB for creating the frequency spectrum of a given signal and noise. We will also perform frequency domain analysis when this signal is convoluted noise created from a random function in MATLAB.

Exercise 5 – Frequency Responses of the System (Bode & Nyquist Diagrams and Nichols Chart)

By referring to the exercises from the previous section, given for example the same example system with the following transfer function equation:

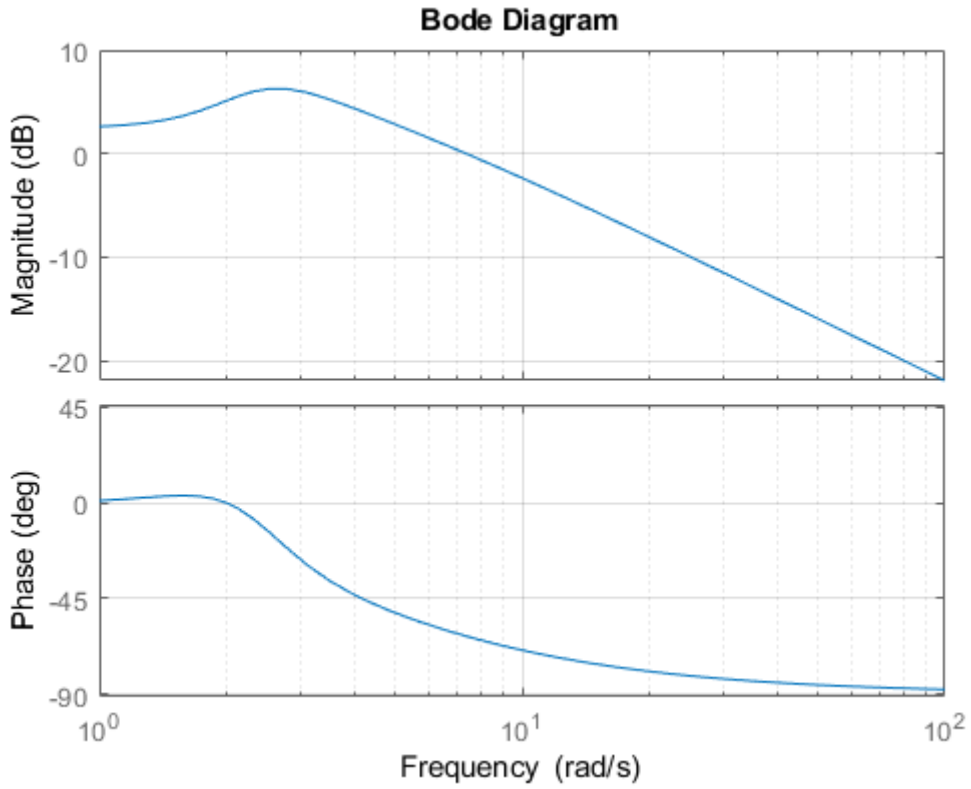
```
sys = tf([8 18 32],[1 6 14 24])
sys =

     8 s^2 + 18 s + 32
  -----------------------
  s^3 + 6 s^2 + 14 s + 24


Continuous-time transfer function.
```

You can create the frequency response of the system by using `bode()` function in the MATLAB. Bode diagram enables frequency domain analysis of signals and/or systems by looking at their magnitude and phase plots over a range of frequencies.

```
bode(sys)
grid
```

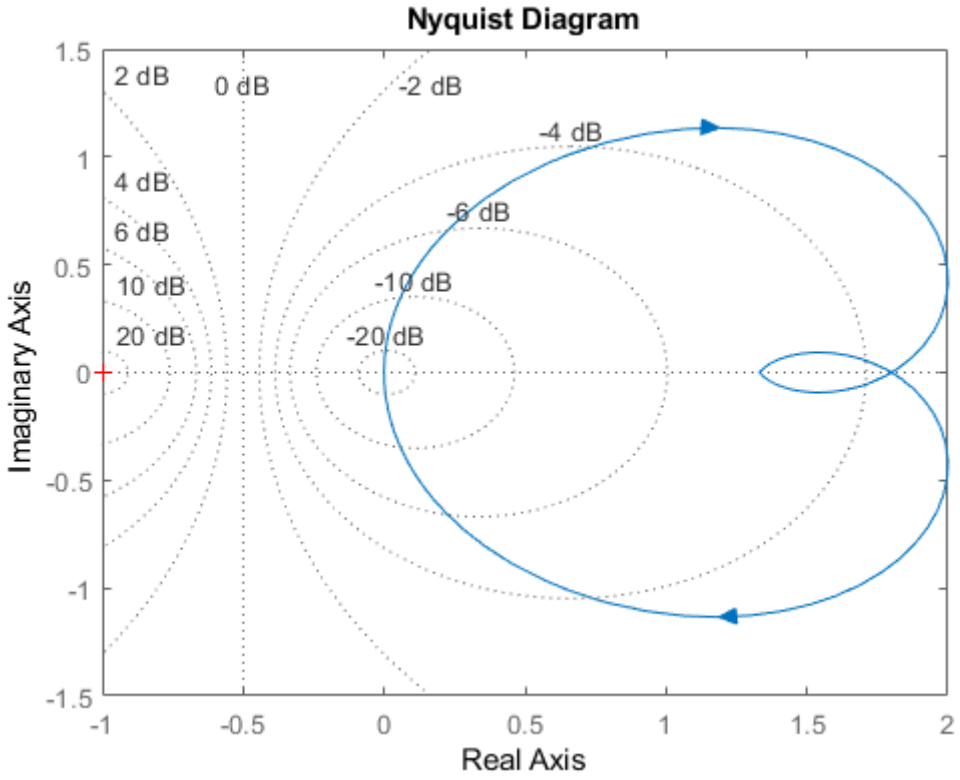


Shown in the Bode plots above are magnitude and phase responses of the system. In the magnitude plot, the amplitude of the system is near zero at lower frequency and is having a peak value at frequency near 5 Hz, before it gradually rolls off at higher frequencies. In the phase plot, the phase of the system is near zero at low frequency and is approaching -90° at higher frequencies. Typically, these plots are the characteristics of a third order system which is basically the type of the system in this simulation.

Although not covered in the 220 Systems and Signal course, for illustration purposes, the following sections show how to create other plots of frequency domain analysis of the systems in MATLAB. In this case two further frequency domain analysis techniques e.g. Nyquist diagram and Nichols chart plots are illustrated.

To create the frequency response of the system as a Nyquist diagram, use the `nyquist()` function command in MATLAB. Engineers typically use the plot to study the stability of the systems throughout their working frequencies.

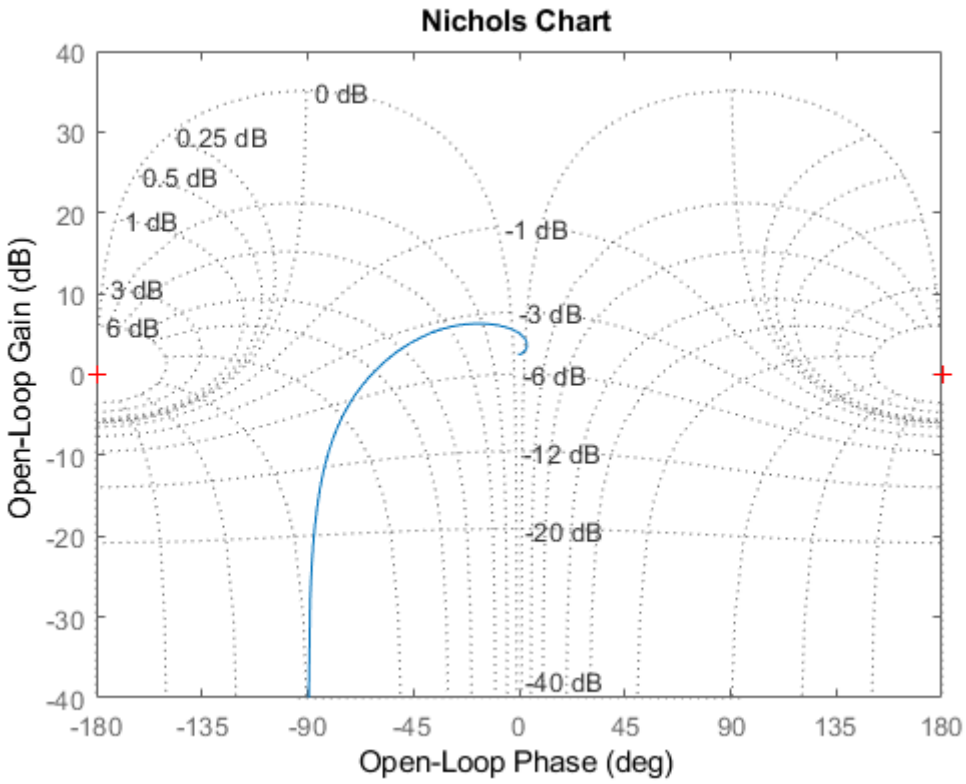
```
nyquist(sys)
grid
```



Shown in the Nyquist diagram above is that the system is stable at low frequency but it experiences some form of instability at higher frequencies.

To create the frequency response of the system in the Nichols chart, use the `nichols()` function command in MATLAB. This plot could be used further to analyse in more details the stability of a given system.

```
nichols(sys)
grid
```

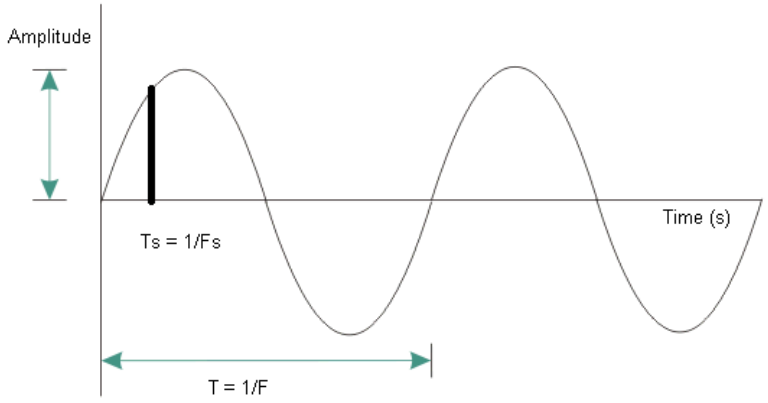


On the Nichols chart above, the stability of the system is observed to be marginal when its working frequency is near -90° phase shift.

Exercise 6 - Frequency Spectra of Signals

In this exercise we will create both time domain plot and frequency spectrum of example sinusoidal signals in MATLAB (one signal is at f = 50 Hz and the other is at f = 120 Hz). Both signals are sampled at 1 kHz for a time duration of 1.5 seconds.

We will also include in the time-domain and frequency-spectrum plots noise using random function (i.e. `rand()`. Then we will see the overall signal in the time domain plot and frequency spectra when this noise is being superimposed onto the sinusoidal signals. Finally, we compare the spectra of the signals with the noise with the spectra of the signals without the noise included.



The transformation from the time-domain to the frequency-domain modelling is carried out via the Fourier transform. In MATLAB, Fourier transform is realised with `fft()` function command. This function enables the transformation of a signal or system in the time-domain representation to its equivalent frequency-domain representation.

In this exercise, we use Fourier transforms to find the frequency components of a signal buried in noise.

Specify the parameters of a signal with a sampling frequency of 1 kHz and a signal duration of 1.5 seconds.

```
Fs = 1000;            % Sampling frequency
T = 1/Fs;             % Sampling period
L = 1500;             % Length of signal
t = (0:L-1)*T;        % Time vector
```

Form a signal containing a 50 Hz sinusoid of amplitude 0.7 and a 120 Hz sinusoid of amplitude 1.

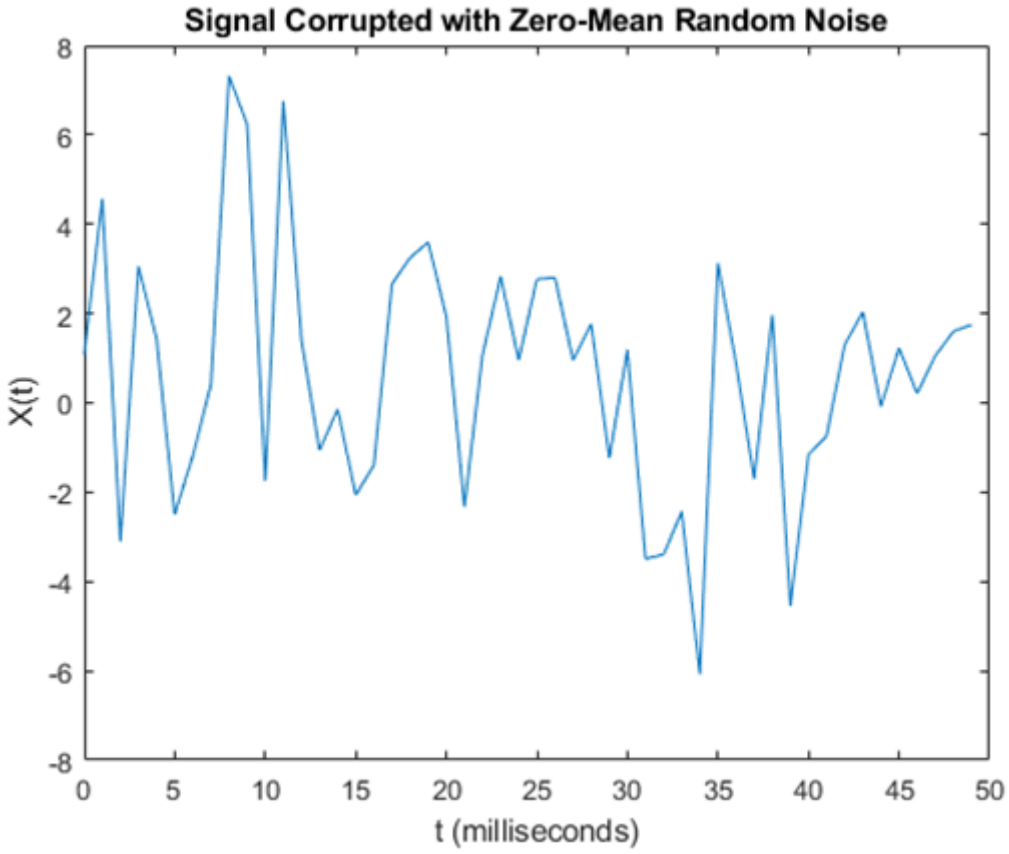
```
S = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t);
```

Corrupt the signal with zero-mean white noise with a variance of 4.

```
X = S + 2*randn(size(t));
```

Plot the noisy signal in the time domain. It is difficult to identify the frequency components by looking at the signal `X(t)`.

```
plot(1000*t(1:50),X(1:50))
title('Signal Corrupted with Zero-Mean Random Noise')
xlabel('t (milliseconds)')
ylabel('X(t)')
```



Notice that the waveform of the signal is distorted by the noise and also observe zero-mean nature of the random noise added into the signal.

Compute the Fourier transform of the signal.

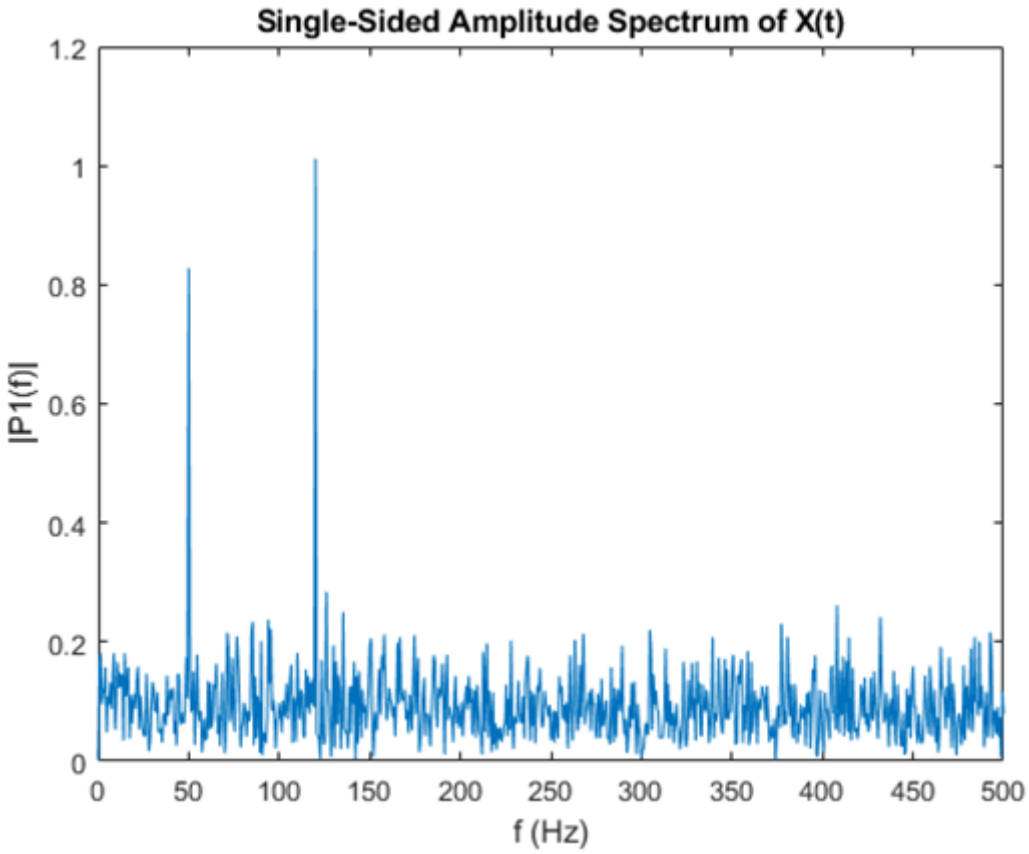
```
Y = fft(X);
```

Compute the two-sided spectrum `P2`. Then compute the single-sided spectrum `P1` based on `P2` and the even-valued signal length `L`.

```
P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
```

Define the frequency domain `f` and plot the single-sided amplitude spectrum `P1`. The amplitudes are not exactly at 0.7 and 1, as expected, because of the added noise. On average, longer signals produce better frequency approximations.

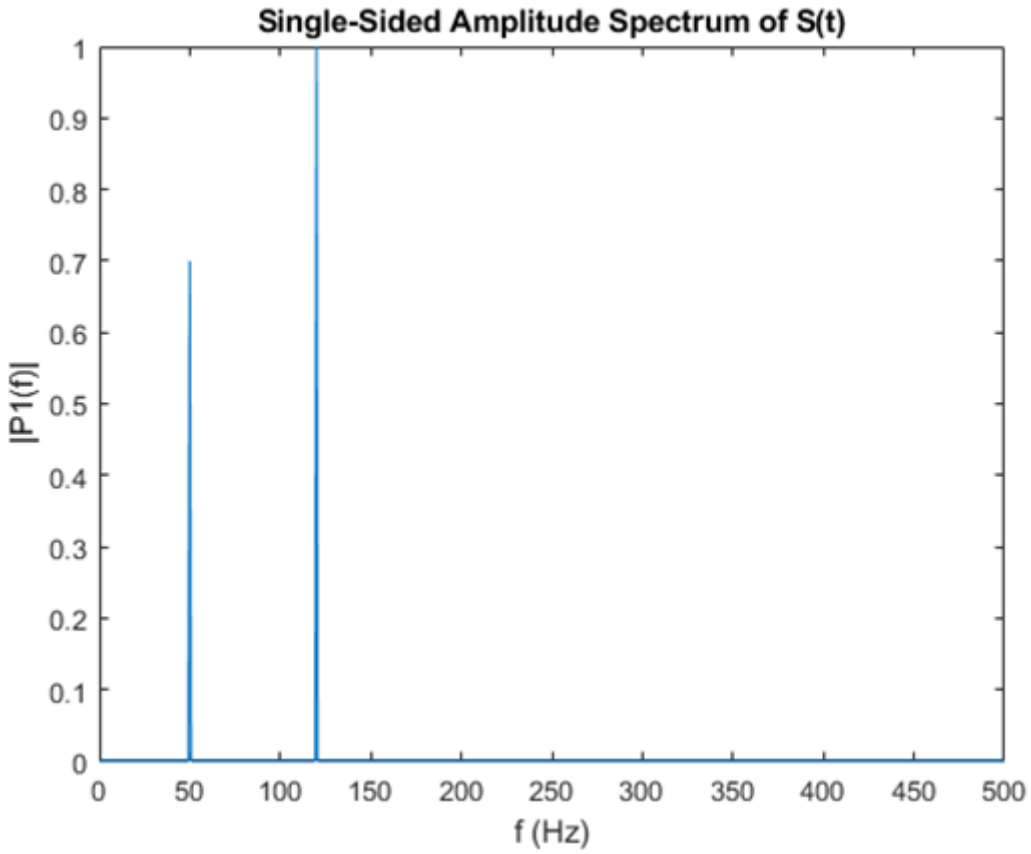
```
f = Fs*(0:(L/2))/L;
plot(f,P1)
title('Single-Sided Amplitude Spectrum of X(t)')
xlabel('f (Hz)')
ylabel('|P1(f)|')
```



As shown on the frequency spectrum plot, that the frequency spectrum of the noise dominates the graph. Although the noise magnitude is very small, but its spectrum spans the whole plot. Notice also the two frequency spectra of the two signals at 50 Hz and 120 Hz. Their amplitudes are about 0.8 and 1.0 respectively.

Now, take the Fourier transform of the original, uncorrupted signal and retrieve the exact amplitudes, 0.7 and 1.0.

```
Y = fft(S);
P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
plot(f,P1)
title('Single-Sided Amplitude Spectrum of S(t)')
xlabel('f (Hz)')
ylabel('|P1(f)|')
```



Compared with the previous frequency spectrum plot, now the graph above has no noise signal existing in the plot. Notice the presence of the two signals at 50 Hz and 120 Hz.

Appendix A – Useful Functions for Time Domain Analysis in MATLAB

A.1. `impulse()` function

This function command computes the impulse response of LTI models

Syntax:

```
impulse(sys)
impulse(sys,t)
impulse(sys1,sys2,...,sysN)
impulse(sys1,sys2,...,sysN,t)
impulse(sys1,'PlotStyle1',...,sysN,'PlotStyleN')
[y,t,x] = impulse(sys)
```

Description:

The impulse function command calculates the unit impulse response of a linear system. The impulse response is the response to a Dirac input for continuous-time systems and to a unit pulse for discrete-time systems. Zero initial state is assumed in the state-space case. When invoked without left-hand arguments, this function plots the impulse response on the screen.

`impulse(sys)` plots the impulse response of an arbitrary LTI model sys. This model can be continuous or discrete simple system. The impulse response of multi-input systems is the collection of impulse responses for each input channel. The duration of simulation is determined automatically to display the transient behaviour of the response.

`impulse(sys,t)` sets the simulation horizon explicitly. You can specify either a final time `t = Tfinal` (in seconds), or a vector of evenly spaced time samples of the form

```
t = 0:dt:Tfinal
```

For discrete systems, the spacing `dt` should match the sample period. For continuous systems, `dt` becomes the sample time of the discretized simulation model, so make sure to choose `dt` small enough to capture transient phenomena.

To plot the impulse responses of several LTI models sys1,..., sysN on a single figure, use

```
impulse(sys1,sys2,...,sysN)
impulse(sys1,sys2,...,sysN,t)
```

As with bode or plot, you can specify a particular colour, line style, and/or marker for each system, for example:

```
impulse(sys1,'y:',sys2,'g--')
```

When invoked with left-side arguments,

```
[y,t] = impulse(sys)
```

```
[y,t,x] = impulse(sys)        % for state-space models only
y = impulse(sys,t)
```

return the output response y, the time vector t used for simulation, and the state trajectories x (for state-space models only). No plot is drawn on the screen. For single-input systems, y has as many rows as time samples (length of t), and as many columns as outputs. In the multi-input case, the impulse responses of each input channel are stacked up along the third dimension of y. The dimensions of y are then:

```
(length of t)×(number of outputs)×(number of inputs)
```

and y(:,:,j) gives the response to an impulse disturbance entering the jth input channel. Similarly, the dimensions of x are:

```
(length of t)×(number of states)×(number of inputs)
```

A.2. `step()` function

This function command creates step response plot of dynamic system. It also provides step response data.

Syntax:

```
step(sys)
step(sys,Tfinal)
step(sys,t)
step(sys1,sys2,...,sysN)
step(sys1,sys2,...,sysN,Tfinal)
step(sys1,sys2,...,sysN,t)
y = step(sys,t)
[y,t] = step(sys)
[y,t] = step(sys,Tfinal)
[y,t,x] = step(sys)
[y,t,x,ysd] = step(sys)
[y,...] = step(sys,...,options)
```

Description:

`step` calculates the step response of a dynamic system. For the state-space case, zero initial state is assumed. When it is invoked with no output arguments, this function plots the step response on the screen.

`step(sys)` plots the step response of an arbitrary dynamic system model, `sys`. This model can be continuous- or discrete-time simple system. The step response of multi-input systems is the collection of step responses for each input channel. The duration of simulation is determined automatically, based on the system poles and zeros.

`step(sys,Tfinal)` simulates the step response from `t = 0` to the final time `t = Tfinal`. Express `Tfinal` in the system time units, specified in the `TimeUnit` property of `sys`. For discrete-time

systems with unspecified sample time (`Ts = -1`), `step` interprets `Tfinal` as the number of sampling periods to simulate.

`step(sys,t)` uses the user-supplied time vector `t` for simulation. Express `t` in the system time units, specified in the `TimeUnit` property of `sys`. For discrete-time models, `t` should be of the form `Ti:Ts:Tf`, where `Ts` is the sample time. For continuous-time models, `t` should be of the form `Ti:dt:Tf`, where `dt` becomes the sample time of a discrete approximation to the continuous system. The `step` command always applies the step input at `t=0`, regardless of `Ti`.

To plot the step response of several models `sys1`,..., `sysN` on a single figure, use

```
step(sys1,sys2,...,sysN)
step(sys1,sys2,...,sysN,Tfinal)
step(sys1,sys2,...,sysN,t)
```

All of the systems plotted on a single plot must have the same number of inputs and outputs. You can, however, plot a mix of continuous- and discrete-time systems on a single plot. This syntax is useful to compare the step responses of multiple systems.

You can also specify a distinctive colour, line style, marker, or all three for each system. For example,

```
step(sys1,'y:',sys2,'g--')
```

plots the step response of `sys1` with a dotted yellow line and the step response of `sys2` with a green dashed line.

When invoked with output arguments:

```
y = step(sys,t)
[y,t] = step(sys)
[y,t] = step(sys,Tfinal)
[y,t,x] = step(sys)
```

`step` returns the output response `y`, the time vector `t` used for simulation (if not supplied as an input argument), and the state trajectories `x` (for state-space models only). No plot generates on the screen. For single-input systems, `y` has as many rows as time samples (length of `t`), and as many columns as outputs. In the multi-input case, the step responses of each input channel are stacked up along the third dimension of `y`. The dimensions of `y` are then

```
(length of t)×(number of outputs)×(number of inputs)
```

and `y(:,:,j)` gives the response to a unit step command injected in the `j`th input channel. Similarly, the dimensions of `x` are

```
(length of t)×(number of states)×(number of inputs)
```

For identified models (see `idlti` and `idnlmodlel`) `[y,t,x,ysd] = step(sys)` also computes the standard deviation `ysd` of the response `y` (`ysd` is empty if `sys` does not contain parameter covariance information).

`[y,...] = step(sys,...,options)` specifies additional options for computing the step response, such as the step amplitude or input offset. Use `stepDataOptions` to create the option set `options`.

A.3. `lsim()` function

This function command simulates time response of dynamic system to arbitrary inputs.

Syntax:

```
lsim(sys,u,t)
lsim(sys,u,t,x0)
lsim(sys,u,t,x0,method)
lsim(sys1,...,sysn,u,t)
lsim(sys1,LineSpec1,...,sysN,LineSpecN,u,t)
y = lsim(__)
[y,t,x] = lsim(__)
lsim(sys)
```

Description:

`lsim` simulates the (time) response of continuous or discrete linear systems to arbitrary inputs. When invoked without left-hand arguments, `lsim` plots the response on the screen.

`lsim(sys,u,t)` produces a plot of the time response of the dynamic system model `sys` to the input history, `t`,`u`. The vector `t` specifies the time samples for the simulation (in system time units, specified in the `TimeUnit` property of `sys`), and consists of regularly spaced time samples:

```
t = 0:dt:Tfinal
```

The input `u` is an array having as many rows as time samples (`length(t)`) and as many columns as system inputs. For instance, if `sys` is a simple system, then `u` is a t-by-1 vector. If `sys` has three inputs, then `u` is a t-by-3 array. Each row `u(i,:)` specifies the input value(s) at the time sample `t(i)`. The signal `u` also appears on the plot.

The model `sys` can be continuous or discrete simple system. In discrete time, `u` must be sampled at the same rate as the system. In this case, the input `t` is redundant and can be omitted or set to an empty matrix. In continuous time, the time sampling `dt = t(2)-t(1)` is used to discretize the continuous model. If `dt` is too large (undersampling), `lsim` issues a warning suggesting that you use a more appropriate sample time, but will use the specified sample time.

`lsim(sys,u,t,x0)` further specifies an initial condition `x0` for the system states. This syntax applies only when `sys` is a state-space model. `x0` is a vector whose entries are the initial values of the corresponding states of `sys`.

`lsim(sys,u,t,x0,method)` explicitly specifies how the input values should be interpolated between samples, when `sys` is a continuous-time system. Specify `method` as one of the following values:

- `'zoh'` — Use zero-order hold
- `'foh'` — Use linear interpolation (first-order hold)

If you do not specify a method, `lsim` selects the interpolation method automatically based on the smoothness of the signal `u`.

`lsim(sys1,...,sysn,u,t)` simulates the responses of several dynamic system models to the same input history `t`,`u` and plots these responses on a single figure. You can also use the `x0` and `method` input arguments when computing the responses of multiple models.

`lsim(sys1,LineSpec1,...,sysN,LineSpecN,u,t)` specifies the line style, marker, and colour of each of the system responses in the plot. You can also use the `x0` and `method` input arguments with this syntax. Each `LineSpec` argument is specified as a vector of one, two, or three characters. The characters can appear in any order. For example, the following code plots the response of `sys1` as a yellow dotted line and the response of `sys2` as a green dashed line:

```
lsim(sys1,'y:',sys2,'g--',u,t,x0)
```

For more information about configuring this argument, see the `LineSpec` input argument of the `plot` function.

`y = lsim(__)` returns the system response `y`, sampled at the same times as the input (`t`). The output `y` is an array having as many rows as time samples (`length(t)`) and as many columns as system outputs. No plot is drawn on the screen. You can use this syntax with any of the input arguments described in previous syntaxes except the `LineSpec` arguments.

`[y,t,x] = lsim(__)` also returns the time vector `t` used for simulation and the state trajectories `x` (for state-space models only). The output `x` has as many rows as time samples (`length(t)`) and as many columns as system states. You can use this syntax with any of the input arguments described in previous syntaxes except the `LineSpec` arguments.

A.4. `stepinfo()` function

This function command computes the step-response characteristics for a dynamic system model `sys`.

Syntax:

```
S = stepinfo(sys)
S = stepinfo(y,t)
S = stepinfo(y,t,yfinal)
S = stepinfo(___,'SettlingTimeThreshold',ST)
S = stepinfo(___,'RiseTimeLimits',RT)
```

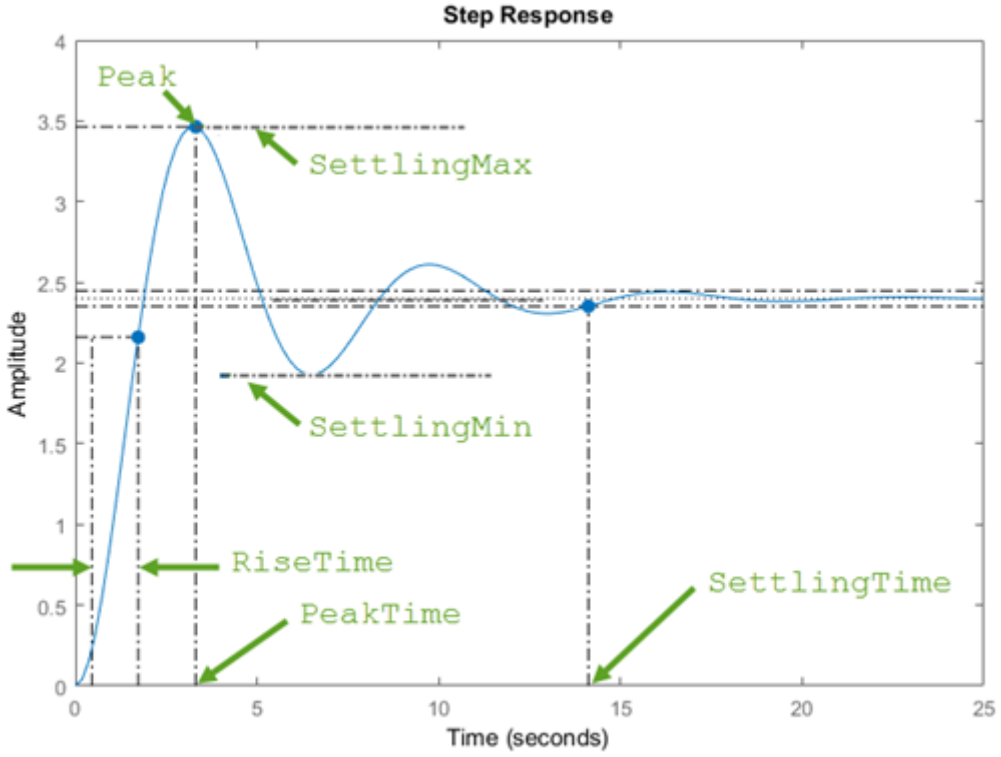
Description:

The function returns the characteristics in a structure containing the fields:

- `RiseTime` — Time it takes for the response to rise from 10% to 90% of the steady-state response.
- `SettlingTime` — Time it takes for the error $|y(t) - y_{final}|$ between the response $y(t)$ and the steady-state response y_{final} to fall to within 2% of y_{final}.

- `SettlingMin` — Minimum value of $y(t)$ once the response has risen.
- `SettlingMax` — Maximum value of $y(t)$ once the response has risen.
- `Overshoot` — Percentage overshoot, relative to y_{final}).
- `Undershoot` — Percentage undershoot.
- `Peak` — Peak absolute value of $y(t)$
- `PeakTime` — Time at which the peak value occurs.

The following figure illustrates some of these quantities on a typical second-order response.



`S = stepinfo(y,t)` computes step-response characteristics from an array of step-response data `y` and corresponding time vector `t`. For simple system responses, `y` is a vector with the same number of entries as `t`. This syntax uses the last value in `y` (or the last value in each channel's corresponding response data) as the steady-state value for computing characteristics that depend on that value.

`S = stepinfo(y,t,yfinal)` computes step-response characteristics relative to the steady-state value `yfinal`. This syntax is useful when you know that the expected steady-state system response differs from the last value in `y` for reasons such as measurement noise. For simple system responses, `t` and `y` are vectors with the same length.

`stepinfo(__,'SettlingTimeThreshold',ST)` lets you specify the threshold `ST` used in the definition of settling time. The response has settled when the error `|y(t) - yfinal|` becomes smaller than a fraction `ST` of its peak value. The default value is `ST = 0.02` (2%). You can use this syntax with any of the previous input-argument combinations.

`S = stepinfo(__,'RiseTimeLimits',RT)` lets you specify the lower and upper thresholds used in the definition of rise time. By default, the rise time is defined as the time the response takes to rise from 10 to 90% of the steady-state value (`RT = [0.1 0.9]`). The upper threshold `RT(2)` is also used to calculate `SettlingMin` and `SettlingMax`. These values are the minimum and maximum values of the response occurring after the response has reached the upper threshold. You can use this syntax with any of the previous input-argument combinations.

Appendix B – Frequency Domain Analysis Functions in MATLAB

B.1. `bode()` function

This function command create Bode plot of frequency response, or magnitude and phase data

Syntax:

```
bode(sys)
bode(sys1,sys2,...,sysN)
bode(sys1,LineSpec1,...,sysN,LineSpecN)
bode(__,w)
[mag,phase,wout] = bode(sys)
[mag,phase,wout] = bode(sys,w)
[mag,phase,wout,sdmag,sdphase] = bode(sys,w)
```

Description:

`bode(sys)` creates a Bode plot of the frequency response of a dynamic system model `sys`. The plot displays the magnitude (in dB) and phase (in degrees) of the system response as a function of frequency. `bode` automatically determines frequencies to plot based on system dynamics.

If `sys` is a multi-input, multi-output (MIMO) model, then `bode` produces an array of Bode plots, each plot showing the frequency response of one I/O pair.

`bode(sys1,sys2,...,sysN)` plots the frequency response of multiple dynamic systems on the same plot. All systems must have the same number of inputs and outputs.

`bode(sys1,LineSpec1,...,sysN,LineSpecN)` specifies a colour, linestyle, and marker for each system in the plot.

`bode(__,w)` plots system responses for frequencies specified by `w`.

- If `w` is a cell array of the form `{wmin,wmax}`, then `bode` plots the response at frequencies ranging between `wmin` and `wmax`.
- If `w` is a vector of frequencies, then `bode` plots the response at each specified frequency.

You can use `w` with any of the input-argument combinations in previous syntaxes.

`[mag,phase,wout] = bode(sys)` returns the magnitude and phase of the response at each frequency in the vector `wout`. The function automatically determines frequencies in `wout` based on system dynamics. This syntax does not draw a plot.

`[mag,phase,wout] = bode(sys,w)` returns the response data at the frequencies specified by `w`.

- If `w` is a cell array of the form `{wmin,wmax}`, then `wout` contains frequencies ranging between `wmin` and `wmax`.
- If `w` is a vector of frequencies, then `wout` = `w`.

`[mag,phase,wout,sdmag,sdphase] = bode(sys,w)` also returns the estimated standard deviation of the magnitude and phase values for the identified model `sys`. If you omit `w`, then the function automatically determines frequencies in `wout` based on system dynamics.

B.2. `fft()` function

This function command performs Fast Fourier transform on a given signal or system.

Syntax:

```
Y = fft(X)
Y = fft(X,n)
Y = fft(X,n,dim)
```

Description:

`Y = fft(X)` computes the discrete Fourier transform (DFT) of `X` using a fast Fourier transform (FFT) algorithm.

- If `X` is a vector, then `fft(X)` returns the Fourier transform of the vector.
- If `X` is a matrix, then `fft(X)` treats the columns of `X` as vectors and returns the Fourier transform of each column.
- If `X` is a multidimensional array, then `fft(X)` treats the values along the first array dimension whose size does not equal 1 as vectors and returns the Fourier transform of each vector.
- `Y = fft(X,n)` returns the `n`-point DFT. If no value is specified, `Y` is the same size as `X`.
- If `X` is a vector and the length of `X` is less than `n`, then `X` is padded with trailing zeros to length `n`.
- If `X` is a vector and the length of `X` is greater than `n`, then `X` is truncated to length `n`.
- If `X` is a matrix, then each column is treated as in the vector case.
- If `X` is a multidimensional array, then the first array dimension whose size does not equal 1 is treated as in the vector case.

`Y = fft(X,n,dim)` returns the Fourier transform along the dimension `dim`. For example, if `X` is a matrix, then `fft(X,n,2)` returns the n-point Fourier transform of each row.