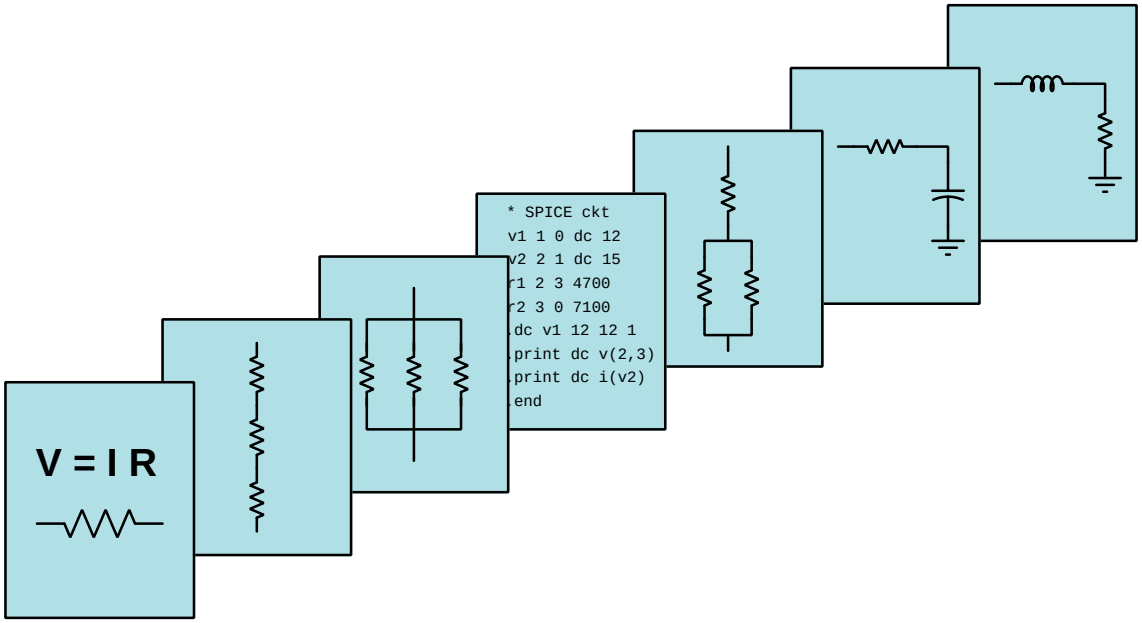
Modular Electronics Learning (ModEL) project



Elementary Filter Circuits

Last update = 9 May 2025

Contents

Chapter 1

Introduction

1.1 Recommendations for students

Many practical electrical and electronic circuit applications are characterized by a combination of signals which must be separated from one another. *Radio* communication works by broadcasting electromagnetic waves at high frequency, and all these broadcasts must be distinguished from one another by their set frequencies. Sensors used to measure physical variables such as temperature and position and convert them into electrical signals may suffer from interference by *noise* (i.e. random disturbances to the voltage or current signal) picked up from surrounding circuitry or mechanisms, and that noise must somehow be screened from the received signal in order to accurately interpret the sensor's measurement. Precision AC-to-DC power supply circuits must have their outputs conditioned to screen out any unwanted "ripple" or other AC disturbances in the otherwise continuous (DC) power sent to sensitive loads.

A *filter* circuit is one designed to perform any of these tasks, discriminating one signal from another based on frequency. A broad range of filter circuit designs exist, and their analysis can be quite mathematically complex. This tutorial seeks to introduce the topic in as general terms as possible, using as little math as possible for the sake of building a strong conceptual understanding of the subject.

Important concepts related to filters include **capacitive reactance**, **inductive reactance**, effects of **opens** versus **shorts**, **voltage divider** networks, **cutoff frequency**, **parasitic** properties, **resonance**, **Bode plots**, **quality factor**, **roll-off**, **decibels**, **crystals**, **fundamental frequency**, and **harmonic frequency**.

A problem-solving technique applied throughout the text is *limiting cases*, where we consider the behavior of a circuit at some extreme condition(s). In this case, the variable we take to these limiting cases is frequency of an AC signal, and we examine the reactance values of capacitors and inductors at those extreme frequency values. Like all limiting-case examples, this tends to simplify the circuit being analyzed, and in so doing provides us with a description of how the circuit will respond to smaller (less-extreme) changes in frequency.

Here are some good questions to ask of yourself while studying this subject:

- How might an experiment be designed and conducted to prove the existence of harmonic frequencies within a non-sinusoidal waveform? What hypothesis (i.e. prediction) might you pose for that experiment, and what result(s) would either support or disprove that hypothesis?
- How might an experiment be designed and conducted to gather enough data to sketch a Bode plot for a filter network having unknown characteristics? What hypothesis (i.e. prediction) might you pose for that experiment, and what result(s) would either support or disprove that hypothesis?
- How might an experiment be designed and conducted to measure the input impedance of a filter network? What hypothesis (i.e. prediction) might you pose for that experiment, and what result(s) would either support or disprove that hypothesis?
- In what form do inductors store energy?
- In what form do capacitors store energy?
- How does the problem-solving technique of "limiting cases" help us understand filter networks?
- What are some practical applications of filter networks?
- How is "cutoff frequency" defined for a filter network?
- Why are capacitors usually favored over inductors for creating filter networks?
- What is "resonance" and how does it manifest in both electrical and mechanical systems?
- How does a Bode plot differ from an oscillograph?
- When might we prefer a filter network with a high quality factor?
- When might we prefer a filter network with a low quality factor?
- What does "roll-off" mean for a filter network?
- What does the phrase "brick wall" mean for the response of an ideal filter network?
- What advantage(s) do quartz crystals bring to filter networks?
- How does filtering affect the frequency-domain spectrum of an AC signal?
- How may filtering be used to re-shape the time-domain shape of an AC signal?

1.2 Challenging concepts related to elementary filter networks

The following list cites concepts related to this module's topic that are easily misunderstood, along with suggestions for properly understanding them:

- **Reasoning from trusted principles** – many students enter college-level study of electronics with an educational background stressing rote memorization at the expense of logical reasoning from trusted principles, and as such tend to find circuit analysis daunting where there is no single procedure or single formula always yielding the correct answer(s). These students also try to rote-memorize circuit configurations rather than use logic to determine what each of those configurations does. In the case of simple filter networks the tendency is to try to memorize the component positions and associate them with labels such as "high-pass", "low-pass", etc. A much better approach is to view each new filter network from the perspective of a voltage divider when subjected to signals of different frequency, using general principles of "opens" and "shorts" to conclude the effects on the output signal as frequency values go to extremes (i.e. DC versus super-high frequency). In other words, apply the problem-solving strategy of *limiting cases* to every filter circuit so as to figure out its function rather than try to memorize it!

- **Shorts versus Opens** – these are two distinctly different types of electrical conditions, and are not generic labels for *any* problem that might occur in a circuit[1]. Each of these conditions is characterized by a prohibition of some electrical measure: shorts prevent voltage from existing between the two points that are shorted together, and opens prevent current from passing through the conductors that used to be joined. A very common misconception is that shorts ensure current and opens ensure voltage, but it is more accurate to say that *shorts prohibit voltage* and *opens prohibit current*.

- **Decibels** – "decibels" are an attempt to express power ratios (i.e. power gains or attenuation factors) logarithmically rather than linearly, and as such they tend to generate confusion for students less familiar (or unfamiliar) with exponential and logarithmic functions.

- **Practical filter applications** – Bode plots show how filter circuits respond to inputs of changing frequency, but this is not how filters are typically used in real applications. Rarely does one find a filter circuit subjected to only one particular frequency at a time – usually a simultaneous mix of frequencies are seen at the input, and it is the filter's job to select a particular range of frequencies to pass through from that simultaneous mix. Understanding the superposition theorem is helpful for comprehending practical filter applications.

The *Case Tutorial* chapter section on decibels gives an overview of this mathematical concept.

[1] It is common for new students of electricity to assume, for example, that "short" means any type of fault whatsoever!

1.3 Recommendations for instructors

This section lists realistic student learning outcomes supported by the content of the module as well as suggested means of assessing (measuring) student learning. The outcomes state what learners should be able to do, and the assessments are specific challenges to prove students have learned.

- **Outcome** – Demonstrate effective technical reading and writing

  Assessment – Students present their outlines of this module's instructional chapters (e.g. Case Tutorial, Tutorial, Historical References, etc.) ideally as an entry to a larger Journal document chronicling their learning. These outlines should exhibit good-faith effort at summarizing major concepts explained in the text.

  Assessment – Students show how quantitative results were obtained by the author in the Tutorial chapter's examples.

- **Outcome** – Apply the "limiting cases" problem-solving strategy to the identification of filter network types

  Assessment – Identify types of filter networks based on qualitative analysis of their schematic diagrams; e.g. pose problems in the form of the "Identifying filter types" and "Identifying (more) filter types" and "Identifying (even more) filter types" Conceptual Reasoning questions.

- **Outcome** – Calculate cutoff frequency values for given filter networks

  Assessment – Calculate the cutoff frequency of a filter circuit given its schematic diagram and component values; e.g. pose problems in the form of the "Filter type and cutoff identifications" Quantitative Reasoning question.

  Assessment – Design a filter network for an application with a specified cutoff frequency and filtering type; e.g. pose problems in the form of the "Designing filters using IEC standard component values" Quantitative Reasoning question.

- **Outcome** – Independent research

  Assessment – Locate filter network datasheets (e.g. SAW-type filters) and properly interpret some of the information contained in those documents including pass characteristic(s), frequency ranges, power ratings, etc.

Chapter 2

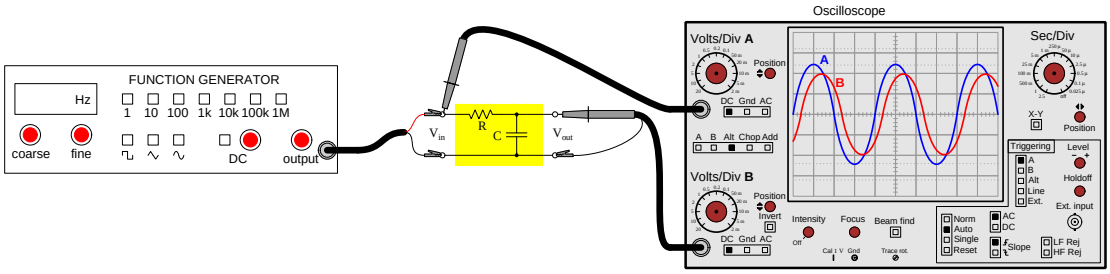
Case Tutorial

The idea behind a *Case Tutorial* is to explore new concepts by way of example. In this chapter you will read less presentation of theory compared to other Tutorial chapters, but by close observation and comparison of the given examples be able to discern patterns and principles much the same way as a scientific experimenter. Hopefully you will find these cases illuminating, and a good supplement to text-based tutorials.

These examples also serve well as challenges following your reading of the other Tutorial(s) in this module – can you explain *why* the circuits behave as they do?

2.1 Example: filter network testing

A simple test arrangement for any passive filter network is to energize its input using a function generator while measuring its output using an oscilloscope. Here we see a diagram of a low-pass resistor-capacitor filter network set up for testing:



Ideally, you could use an AC voltmeter to measure both V_{in} and V_{out}, but most affordable voltmeters have rather limited accuracy over wide ranges in AC frequency, and so an oscilloscope is a more suitable voltage-measuring instrument for this application.

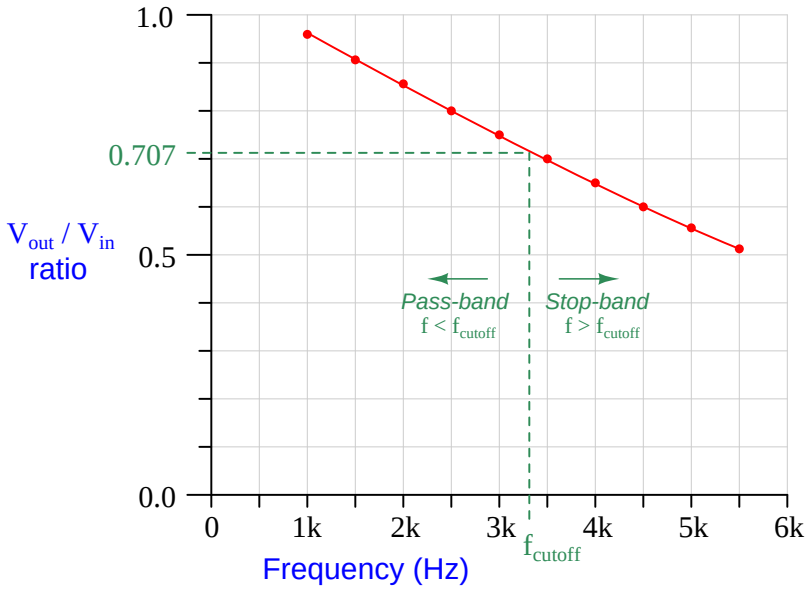
As you manually set the function generator to output sine-wave AC voltages at different frequencies, the oscilloscope will register sine-wave voltage signals at those same frequencies but at different amplitudes. Note how channel A of the oscilloscope measures V_{in} at the filter's input terminals, while channel B measures the filter's V_{out}. The filter network *attenuates* the signal, which explains why V_{out} (channel B) has less amplitude than V_{in} (channel A).

This attenuation factor varies with frequency: for a low-pass filter such as the one being tested here, the output signal weakens with respect to the input signal as frequency rises. If we were testing a high-pass filter instead (just swap the positions of R and C in the circuit diagram!) the V_{out} signal would grow in strength (approaching V_{in}) with increasing frequency. Using the oscilloscope as a two-channel AC voltmeter, you can gather V_{in} and V_{out} data at different frequencies to plot their ratios. Tracking the value of the ratio $\frac{V_{out}}{V_{in}}$ at different frequencies will show us the characteristic behavior of this filter network.

Here is an example of data collected from such a filter network where R is 4.7 kΩ and C is 0.01 μF. From these values we expect the filter to "cut off" at approximately 3386 Hz following the formula $f_{cutoff} = \frac{1}{2\pi RC}$:

Frequency	V_{in} (measured)	V_{out} (measured)	$\frac{V_{out}}{V_{in}}$ (calculated)
1000 Hz	3.00 VAC	2.88 VAC	0.96
1500 Hz	3.00 VAC	2.74 VAC	0.91
2000 Hz	3.00 VAC	2.58 VAC	0.86
2500 Hz	3.00 VAC	2.41 VAC	0.80
3000 Hz	3.00 VAC	2.25 VAC	0.75
3500 Hz	3.00 VAC	2.09 VAC	0.70
4000 Hz	3.00 VAC	1.94 VAC	0.65
4500 Hz	3.00 VAC	1.80 VAC	0.60
5000 Hz	3.00 VAC	1.68 VAC	0.56
5500 Hz	3.00 VAC	1.57 VAC	0.52

Plotting these $\frac{V_{out}}{V_{in}}$ ratio and frequency values on a graph yields a *Bode plot*:



Cutoff frequency is commonly known to be that signal frequency at which the filter network's attenuation is 70.7% (expressed as a ratio). The more fundamental definition, though, is that cutoff frequency is the *half-power point* where the output signal's *power* is exactly one-half that of the

input signal's. Since for any fixed load resistance, voltage is proportional to the square root of power ($V = \sqrt{\frac{P}{R}}$), a power attenuation of one-half is equivalent to a voltage attenuation of $\sqrt{\frac{1}{2}}$, which is where we get the 70.7% value from (0.70711).

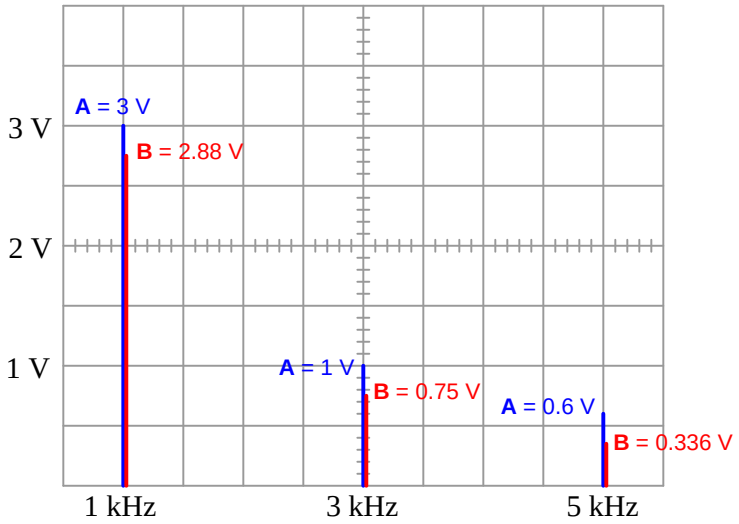
If we convert the "half-power" figure into a decibel value, we see that the "cutoff" point for a filter network is equivalent to a power attenuation of approximately -3 decibels:

$$10 \log (0.5) = -3.01 \text{ dB}$$

If the function generator you are using does not output a "clean" sinusoidal waveform, experimental determination of input and output voltages will be complicated by the fact that the shape of the output waveform will not exactly match the shape of the input waveform. This is because any non-sinusoidal waveform is actually composed of multiple sinusoids superimposed on each other, and the filter network you're testing will attenuate all of these harmonic frequencies to different degrees, effectively changing the shape of the output waveform.

One solution to this problem, assuming you cannot obtain a better-quality signal generator, is to use the spectrum-analyzer function on a digital oscilloscope[1] to measure the input and output voltage spectra. By comparing the respective heights of same-frequency peaks in the input and output spectra, you can easily compute $\frac{V_{out}}{V_{in}}$ ratios. Additionally, if the input waveform is "impure" you will have several pairs of peaks to compare against each other with each frequency setting of the function generator, which will make your testing faster!

For example, the spectrum display shown below (channel A is input, channel B is output) shows a 1 kHz square-wave signal passed through this same filter network, with the same $\frac{V_{out}}{V_{in}}$ ratios as obtained by sweeping a sinusoidal signal and taking several measurements every 500 Hz:



The peak heights of the input signal (3 Volts, 1 Volt, 0.6 Volts) is simply the result of the Fourier series for a square wave and has nothing to do with the filter network:

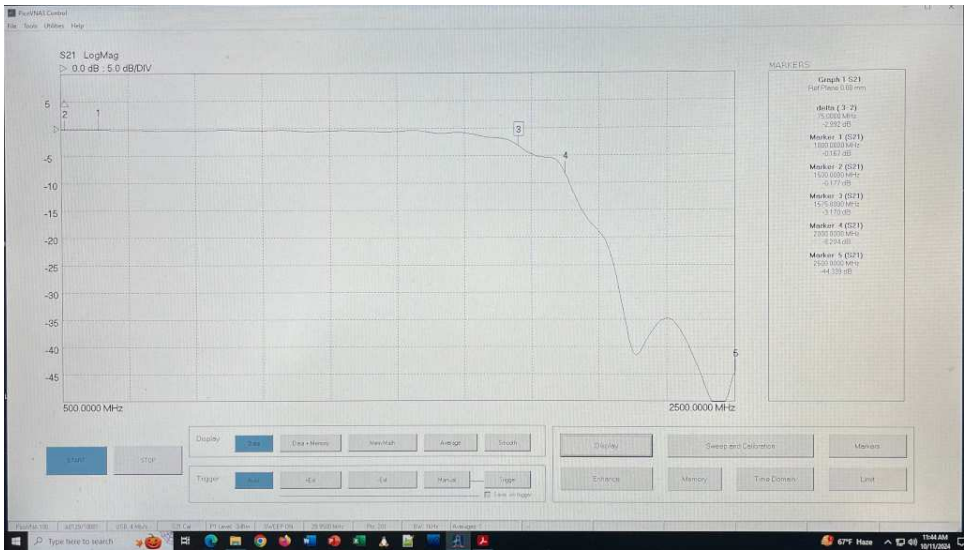
$$\sin \omega t + \frac{1}{3} \sin 3\omega t + \frac{1}{5} \sin 5\omega t + \frac{1}{7} \sin 7\omega t + \cdots + \frac{1}{n} \sin n\omega t$$

What the filter is responsible for is the attenuation of the output signal (channel B), and from the three pairs of peaks in this spectrum display we see the same amplitude *ratios* as shown in the previous table ($\frac{2.88}{3}$ = attenuation of 0.96 at 1 kHz, $\frac{0.75}{1}$ = attenuation of 0.75 at 3 kHz, $\frac{0.336}{0.6}$ = attenuation of 0.56 at 5 kHz).

[1] At the time of this writing (2021) decent-quality spectrum analysis in the audio-frequency range is actually less expensive than decent-quality sinusoidal function generators! Even the least expensive hobbyist-grade digital oscilloscopes come with powerful FFT capability, but similarly-graded function generators struggle to output clean sine waves.

A more sophisticated instrument for testing filters, especially at radio frequencies, is a device called a *Vector Network Analyzer* or *VNA*. This special instrument contains its own high-frequency signal generator (source) as well as precision high-frequency measurement circuitry which makes it ideal for testing the frequency response of any network such as a filter. Here we have a low-pass filter[2] connected between the output port 1 and input port 2 of the VNA through coaxial cable and SMA-style connectors:

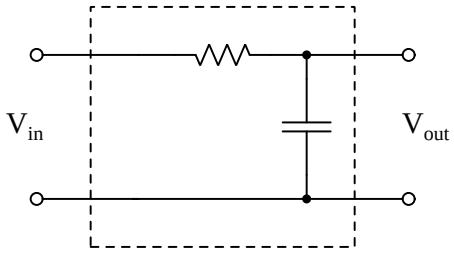
The resulting Bode plot generated by the VNA and displayed on a personal computer connected to the VNA through a USB cable shows a typical low-pass frequency response:



[2]The low-pass filter network is housed inside of the small brass cylinder threaded onto port 2 of the VNA with a cable threaded into its other end.

2.2 Example: RC filter design

Suppose we require a low-pass filter with a cutoff frequency of 5 kHz. On hand we have two different capacitors we might use, a 2.2 nanoFarad (2.2 nF) and a 0.01 microFarad (0.01 μF), as well as a wide range of resistors[3]. First, it is a good idea to sketch the general form of a resistor-capacitor (RC) low-pass filter:



Since we want this to be a *low-pass* filter, and we know that a capacitor's reactance *decreases* as frequency rises ($X_C = \frac{1}{2\pi fC}$), it makes sense to place the capacitor in parallel with the output terminals so that as its reactance approaches zero with increasing frequency it will "short out" the output signal and cause it to grow weaker.

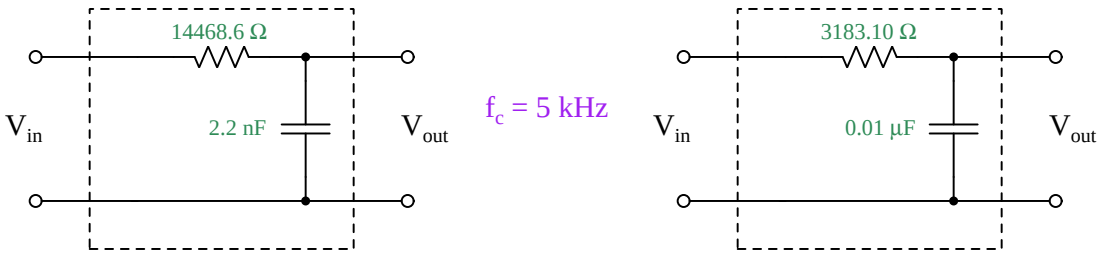
An ideal low-pass filter will fully pass any input signal through to its output terminals below the specified cutoff frequency while completely blocking any signal(s) above that cutoff frequency. However, ideal filter networks do not exist. Real filter networks *gradually* attenuate signals past a specified frequency value, and so "cutoff" must be defined in terms of *some* amount of signal passing through the filter. In the case of a simple RC filter such as this, cutoff frequency is given by the formula $f_c = \frac{1}{2\pi RC}$, but there is actually a more fundamental principle defining this point. For simple reactive filters (whether resistor-capacitor or resistor-inductor) the "cutoff" point is that frequency at which reactance equals resistance ($f = f_c$ when $X = R$). In fact, this is where the formula $f_c = \frac{1}{2\pi RC}$ comes from: if we set resistance (R) equal to capacitive reactance (X_C) and then solve for frequency f, we get $f_c = \frac{1}{2\pi RC}$. This exact same principle is true for simple inductor-resistor filter networks as well: setting $R = 2\pi fL$ and then solving for frequency yields a cutoff frequency of $f_c = \frac{R}{2\pi L}$.

Calculating capacitive reactance at 5 kHz yields 14468.6 Ohms for the 2.2 nF capacitor and 3183.10 Ohms for the 0.01 μF capacitor. This means we may make an RC low-pass filter either by connecting a 14468.6 Ohm resistance[4] to the 2.2 nF capacitor, or by connecting a 3183.10 Ohm resistance to the 0.01 μF capacitor.

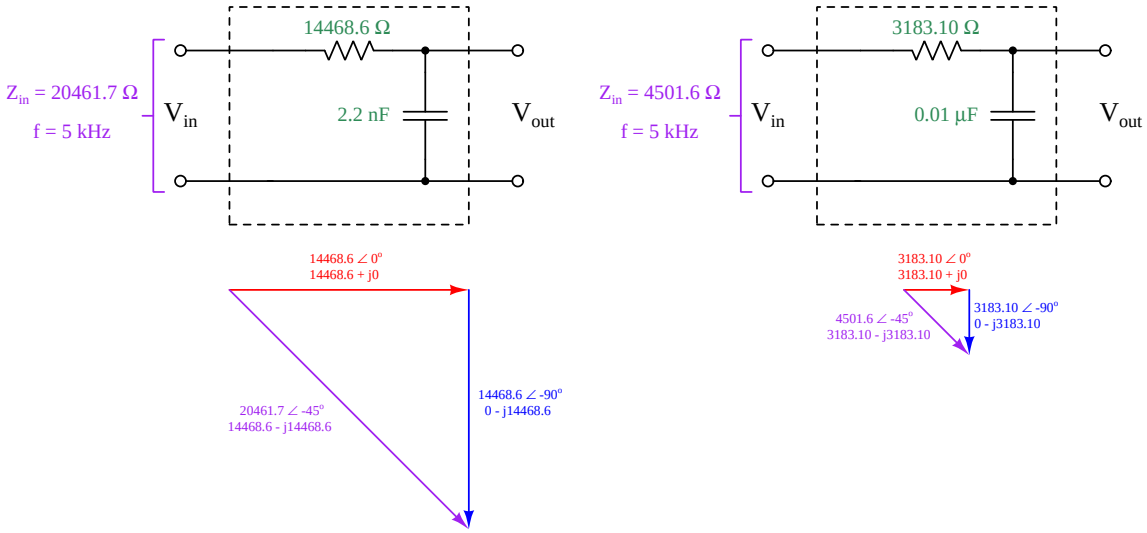
[3] This is a common state of affairs: resistors tend to be more commonly available in a wider range of values than capacitors, and so it makes more sense to choose resistance values to work with a given capacitance value than vice-versa.

[4] It's impossible to locate a single resistor having this particular resistance value, so our actual circuit will use either a series-parallel network of resistors to achieve 14468.6 Ohms and/or a potentiometer (rheostat) that could be precisely adjusted to this value. Alternatively, if our cutoff frequency specification didn't demand high accuracy, we could perhaps use the closest standard resistor value (in this case, 15 kiloOhms) instead.

Both solutions are shown by the following schematics:



Each of these filter networks is low-pass, and each of them has the exact same cutoff frequency value of 5 kHz. They differ in one important regard, though, and that is the amount of impedance they present to the signal source at their respective input terminals. If we consider each filter network on its own with no load connected, we may calculate input impedance as the simple series combination of R and X_C, summing their respective impedance vectors in right-triangle form. Recall that resistance is an impedance with a phase angle of zero, while capacitive reactance is an impedance with a phase angle of negative 90 degrees:



Input impedance matters to whatever electrical source is sending the AC signal to the filter network. If a filter network's impedance is too low for its source, it will cause the source's signal voltage to "sag" (weaken) and possibly even distort the waveform from its proper shape. Conversely, some signal sources require a certain amount of minimum impedance to present a proper amount of load, and so it may be detrimental for a filter network's impedance to be too high. In this example, though, we were not given any specified input impedance – as far as we know, the choice between these two filter designs is entirely arbitrary.

By contrast, had we been given an acceptable range of input impedance for our filter design such as *"Z_{in} must be within the range of 3 kΩ to 8 kΩ"*, we could tell that of these two designs only the one with the 0.01 μF capacitor with its input impedance value of 4501.6 Ω would work for this application.

Armed with an acceptable input impedance range, we may actually compute a *range* of possible R and C values for building a filter network with a 5 kHz cutoff frequency. Knowing that the $Z-R-X$ vector diagram will always form a 45^o angle at cutoff, we may use trigonometric functions to calculate the R and X_C values for minimum and maximum Z_{in}:

At $Z_{in} = 3$ kΩ:

$$R = X_C = (3000\,\Omega)\sin(45^o) = 2121.3\,\Omega$$

$$C = \frac{1}{2\pi(5000\text{ Hz})(2121.3\,\Omega)} = 15.005\text{ nF}$$

At $Z_{in} = 8$ kΩ:

$$R = X_C = (8000\,\Omega)\sin(45^o) = 5656.9\,\Omega$$

$$C = \frac{1}{2\pi(5000\text{ Hz})(5656.9\,\Omega)} = 5.6270\text{ nF}$$

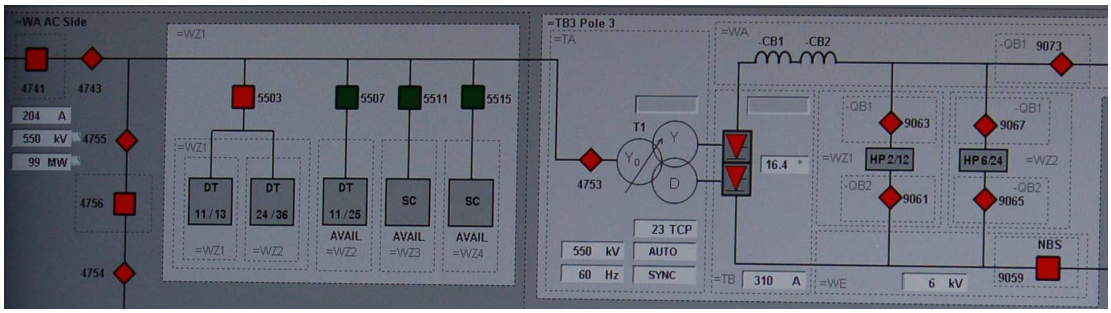
Finding any available capacitor between the values of 5.6270 nF and 15.005 nF, and then combining that capacitor with a resistor whose resistance is equal to that capacitor's reactance at the cutoff frequency of 5 kHz, will yield a filter network having the correct cutoff frequency and an acceptable input impedance for the application.

2.3 Example: HVDC harmonic filters

Some large-scale electric power grids benefit from transmission of power via DC rather than AC. Alternating current (AC) came to dominate power grid technology because AC induction motors and generators were simpler and much more reliable than DC brush-type motors and generators, and also because *transformers* allowed reliable and efficient transformation between different levels of voltage and current than possible with DC. However, modern semiconductor device technology now permits reliable and efficient conversion from AC to DC and also from DC to AC. Such technology, when applied to high-voltage DC transmission of electric power, is commonly referred to by the acronym *HVDC*, which stands for *High Voltage Direct Current*.

One challenge of semiconductor-based AC/DC conversion, though, is the creation of harmonic frequencies in power network voltages and currents. These harmonics, if not attenuated, create problems ranging from excessive component heating to electromagnetic interference with other systems, and so must be mitigated through the use of filter networks installed at HVDC substation facilities.

In the following single-line diagram we see the layout of a very large[5] AC-to-DC converter system where three-phase AC power at a fundamental frequency of 60 Hz gets rectified to DC using a twelve-pulse semiconductor rectifier network. "Twelve-pulse" simply means that for every one cycle of the 60 Hz AC there will be twelve distinct pulses applied to the DC bus. This unavoidably creates a strong 12th harmonic, or 720 Hz, on the DC side of the system. However, other harmonics result from semiconductor switching, including strong 11th and 13th harmonics on the AC side of an HVDC system, which is why we see multiple filters represented in this single-line diagram:



Each filter appears in this diagram as a grey-colored box with the harmonic numbers represented within. For example, the left-most grey box with "DT 11/13" written inside is a filter network designed to attenuate the 11th and 13th harmonics. Immediately to the right of that filter network is another ("DT 24/36") designed to attenuate the 24th and 36th harmonics. Another to the right of that one marked "DT 11/25" attenuates the 11th and 25th harmonics. These three filter networks exist on the AC side of the converter system operating at a nominal system voltage of 550 kiloVolts. On the DC (right-hand) side of this converter system we see another pair of filter networks: one filtering out the 2nd and the 12th harmonics, and another one filtering the 6th and 24th harmonics.

These harmonic numbers all relate to the AC side's fundamental frequency of 60 Hz, and the particular harmonics generated in this HVDC converter system are a function of the number of

[5] This particular HVDC converter facility has a full-power rating in excess of 3000 MegaWatts!

switching elements in the converter (in this case, twelve) and the pattern in which they are switched.

Harmonic filters designed for this purpose consist of resistor-inductor-capacitor networks where the L and C values are "tuned" to resonate together at such frequencies as to attenuate the undesired harmonics from the power circuit. In the photographs shown below we see two views of such a filter network belonging to the DC side of a HVDC converter system, each filter operating at a potential of 550 kV with respect to Earth ground:

In the left-hand photo we see the tall tower-like structure holding the filter's capacitors, and in the background we see the filter's cylindrical air-core inductors, all of them suspended above the Earth on porcelain insulator stacks. The capacitor assembly consists of multiple capacitors connected in series so as to divide the impressed voltage across them rather than have the full 550 kV across any single capacitor's dielectric layer. In the right-hand photo we see a sign on the security fence declaring this filter network to be tuned to the 6th and 24th harmonics (of 60 Hz), screening out those harmonics created by the twelve-pulse thyristor array from reaching the DC power transmission line.

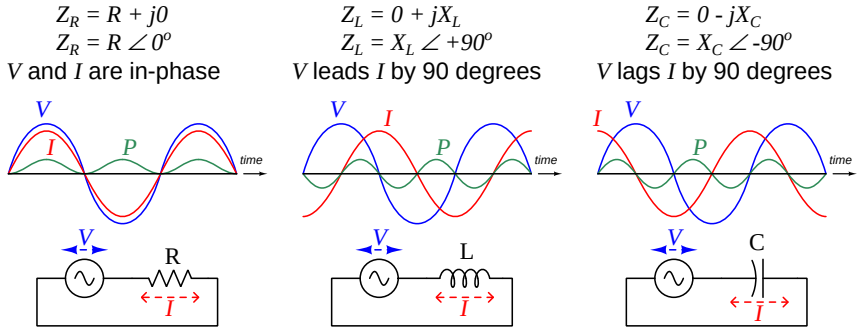
Chapter 3

Tutorial

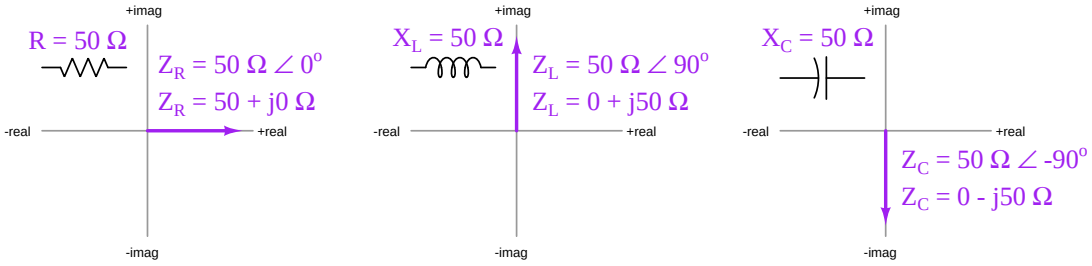
3.1 Phasor analysis review

When analyzing any AC circuit, it is not enough to simply quantify every voltage and current in terms of amplitude (e.g. how many Volts or Amperes) as we do in DC circuits. Instead, we must consider both the amplitude of each signal as well as the amount of phase shift separating them. A mathematically elegant way of accomplishing both is to use *complex numbers* which may be expressed either in rectangular form (e.g. $a + jb$) or polar form (e.g. $m\angle\theta$). When we use complex numbers to represent any AC circuit quantity, we call that value a *phasor*.

An illustrative example is how we characterize the *impedance* of passive components. Resistors function by dissipating energy in the form of heat, with voltage and current waveforms being perfectly in-phase. Inductors and capacitors, however, function by absorbing and releasing energy rather than dissipating, and as such we find voltage and current waveforms shifted by one-quarter of a cycle (i.e. 90^o) for each, voltage leading current for an inductor and voltage lagging current for a capacitor:



Graphical expressions of complex-number resistance and reactance values are called *phasor diagrams*. Three such diagrams illustrate the difference between a 50 Ω resistor versus an inductor having 50 Ω of reactance and a capacitor also having 50 Ω of reactance. Each has 50 Ohms of impedance, but each of these impedances has a different phase angle defined by the phase shift between the component's voltage and current:



The ratio of voltage to current in a DC network is resistance ($R = \frac{V}{I}$), and in an AC circuit where phase shifts exist is impedance ($Z = \frac{V}{I}$). Both are measured in unit of the Ohm (Ω). The ratio of current to voltage is the reciprocal of these quantities: for DC networks we call it *conductance* ($G = \frac{I}{V}$) and for AC it is known as *admittance* ($Y = \frac{I}{V}$). We measure both in the unit of *Siemens*[1] (S).

Some examples[2] of component impedances and admittances are shown here:

- A 570 Ω resistor at any frequency will have the following impedance and admittance values:

  $Z = 570\ \Omega \angle 0^o$ *(polar form)* $= 570 + \text{j}0\ \Omega$ *(rectangular form)*

  $Y = 0.0017544\ \text{S} \angle 0^o$ *(polar form)* $= 0.0017544 + \text{j}0\ \text{S}$ *(rectangular form)*

- A 3.5 H inductor at a frequency of 120 Hz will have the following impedance and admittance values:

  $Z = 2.639\ \text{k}\Omega \angle +90^o$ *(polar form)* $= 0 + \text{j}2.639\ \text{k}\Omega$ *(rectangular form)*

  $Y = 0.00037894\ \text{S} \angle -90^o$ *(polar form)* $= 0 - \text{j}0.00037894\ \text{S}$ *(rectangular form)*

- A 0.01 μF capacitor at a frequency of 3 kHz will have the following impedance and admittance values:

  $Z = 5.305\ \text{k}\Omega \angle -90^o$ *(polar form)* $= 0 - \text{j}5.305\ \text{k}\Omega$ *(rectangular form)*

  $Y = 0.00018850\ \text{S} \angle 90^o$ *(polar form)* $= 0 + \text{j}0.00018850\ \text{S}$ *(rectangular form)*

The utility of phasor representation in AC circuits is that with all signal and component values expressed in phasor form we find most of the foundational principles learned for DC circuit analysis still apply in AC circuits. Quantities that add in series DC networks (e.g. voltage V, resistance R) add as phasor quantities in AC networks (e.g. voltage V, impedance Z); additive quantities in parallel DC networks (e.g. current I, conductance G) add as phasor quantities in AC networks (e.g. current I, admittance Y). With phasor quantities, Ohm's Law, Kirchhoff's Voltage Law, and Kirchhoff's Current Law still hold true in AC networks just as they do for DC.

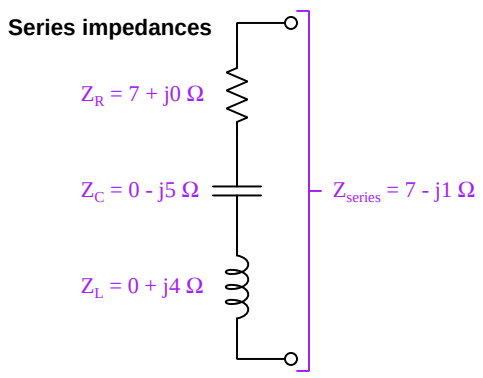
[1] Prior to the adoption of German engineer Werner von Siemens' surname as the unit of measurement for conductance and admittance, the unit of the *Mho* served quite well. This, of course, was a sort of pun on the spelling of *Ohm*, since "mho" is "ohm" spelled backwards, intended to represent the fact that the reciprocal of any Ohm value yields a value in Mhos.

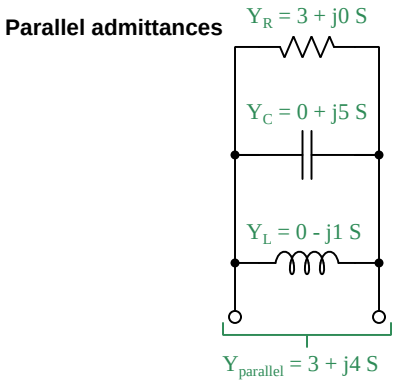
[2] Try calculating these impedance and admittance values from the given component values, to check your understanding. This is a good learning strategy to apply when reading any mathematical text: work through the presented examples on your own to see if you achieve the same results! Please note that when you apply either the $X_L = 2\pi fL$ formula or the $X_C = \frac{1}{2\pi fC}$ formula using your calculator to compute reactance, the result will *only* be a reactance value and not a (complex) impedance value. In order to attach the desired phase angle to your computed reactance value, you will have to perform the additional step of multiplying that reactance by a *unit phasor* which is nothing more than the quantity of 1 with the correct phase angle. For example, a capacitive reactance of 5.305 kΩ would be multiplied by $1 \angle -90^o$ to yield a capacitive *impedance* of 5.305 k$\Omega \angle -90^o$.

It is equally valid to express any phasor quantity in either polar or rectangular form. However, unless we have access to an electronic calculator capable of performing complex-number arithmetic, we find certain arithmetic operations much easier to perform with one notation more than the other. Specifically, addition and subtraction are simplest when phasors are in rectangular form, while multiplication and division are simplest when phasors are in polar form.

Addition of rectangular-form complex numbers consists simply of adding their real components together to find the real component of the sum, and doing the same with the imaginary components. Expressing this algebraically, $(a + jb) + (x + jy) = (a + x) + j(b + y)$. Subtraction follows much the same pattern: $(a + jb) - (x + jy) = (a - x) + j(b - y)$.

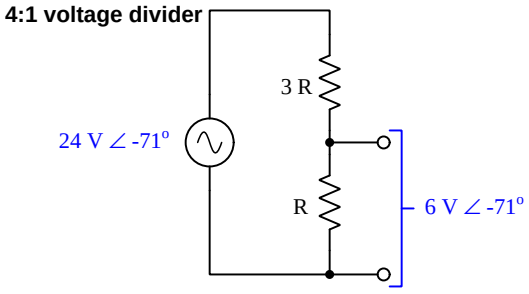
Here are some practical examples of rectangular-form phasor arithmetic where the calculations are simple enough to perform without a calculator:

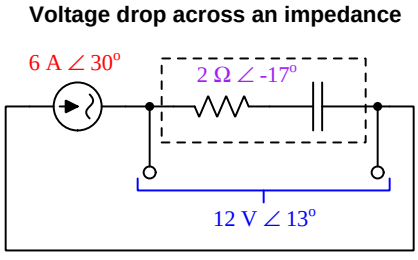




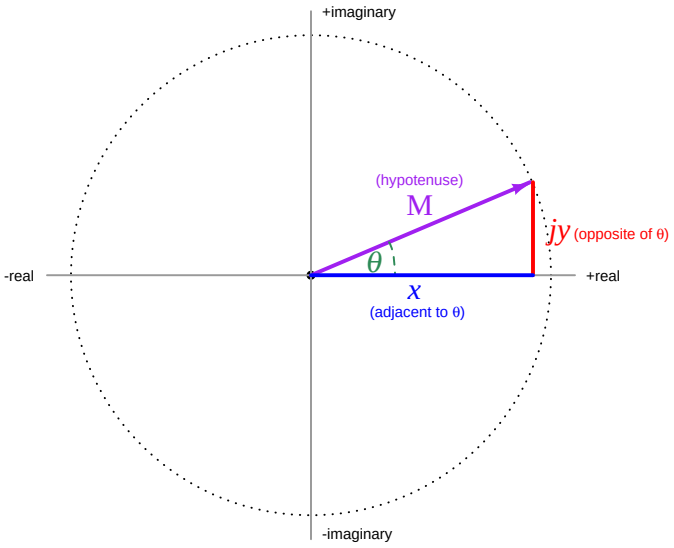
Multiplication of polar-form complex numbers consists simply of multiplying their magnitudes together to find the magnitude of the product, and adding the angles to find the angle of the product. Expressing this algebraically, $(a\angle b) \times (x\angle y) = (a \times x)\angle(b + y)$. Division follows a similar pattern: $(a\angle b) \div (x\angle y) = (a \div x)\angle(b - y)$.

Here are some practical examples of polar-form phasor arithmetic where the calculations are simple enough to perform without a calculator:





When analyzing AC circuits without the use of a complex-number calculator, we invariably must convert between rectangular and polar forms in order to prepare the phasor values for addition/subtraction or multiplication/division, respectively. Both conversions are most easily understood in terms of a right triangle, seeing the rectangular form's real and imaginary components as the adjacent and opposite sides, and the polar form's magnitude and angle as the hypotenuse:



Converting rectangular $(x + jy)$ into polar $(A\angle\theta)$:

$$A = \sqrt{x^2 + y^2} \qquad \theta = \arctan \frac{y}{x}$$

Converting polar $(A\angle\theta)$ into rectangular $(x + jy)$:

$$x = A \cos \theta \qquad y = A \sin \theta$$

Some cautionary notes are in order here. First, I highly recommend storing all computed values in your calculator's memory rather than re-entering them manually, because you will find even slight rounding errors tend to become exaggerated with trigonometric functions. Second, when computing the phase angle (θ) from real and imaginary quantities (x and jy) be careful to verify the angle against your qualitative expectations. For example, $5 + j5 = 7.071\angle 45^o$ and $-5 - j5 = 7.071\angle 225^o$, but you'll find arctan $\frac{-5}{-5}$ yields the same result (45^o) as arctan $\frac{5}{5}$ because $\frac{-5}{-5} = \frac{5}{5}$. To put it simply, the arc-tangent function does not "know" whether the phasor exists in the first or in the third quadrant of the complex plane.

Here are some rectangular and polar equivalents, useful for practice as you master these concepts:

$$20 - j11 = 22.83\angle -28.81^o \qquad 11.49 + j9.642 = 15\angle 40^o$$

$$-10 + j2 = 10.20\angle 168.7^o \qquad -11.82 - j2.084 = 12\angle -170^o$$

3.2 Signal separation

Separating thoroughly-mixed materials can be rather difficult. Liquid solutions such as saltwater or colloidal suspensions such as dairy milk, for example, require significant investments of energy to separate into their constituent compounds. One method for performing such separation is to pass the liquid solution through a *filter* with pores sized appropriately[3] to strain out some components from the rest.

Electrical signals of differing frequency, once mixed together, are generally much easier to separate than mixtures of matter. Any circuit designed to perform this task of separating one or more signals from the rest is called a *filter* as well, and for the same reason: a filter allows some components to pass through while blocking others.

Most people are quite familiar with a simple example of electrical filtering in music reproduction systems: the *treble* and *bass* controls on a typical audio amplifier serve to boost or attenuate tones of a certain frequency range from the complex mixture of frequencies that is music. "Treble" controls affect high-frequency components of the music (e.g. cymbals, violin, flute) while "bass" controls affect low-frequency components of the music (e.g. kick drum, tuba, bass guitar). Turning either of these controls to their minimum settings *filter out* those frequency ranges to make them less prominent in the final mix you hear. A more sophisticated device called an *equalizer* allows the listener to control the relative volumes of narrower frequency ranges in order to customize the range of frequencies heard.

[3] *Reverse osmosis* is an example of a type of filtration suitable for separating salt and other minerals from water.

3.3 Reactive filtering

A filter circuit must discriminate one signal from another on the sole basis of *frequency*. In order to build a circuit that performs this task, we must use electrical components whose characteristics vary with frequency. Fortunately, we already know of two classes of electrical component responding to frequency: *capacitors* and *inductors*. Capacitors store and release energy electrostatically as a function of voltage, while inductors store and release energy electromagnetically as a function of current. When subjected to AC, the alternate storage and release of energy in these components has the effect of impeding changes in voltage or current, and this manifests as a value in Ohms called *reactance* (X). The fundamental difference between resistance and reactance is that while resistance "spends" energy, reactance "borrows" and "returns" energy. Either way, though, a component impeding electricity may be used to attenuate an electrical signal, and *reactive* components impede electrical signals differently according to their frequency.

Capacitive reactance opposes changes in voltage; the greater the frequency, the less opposition. Inductors are just the opposite, opposing changes in current with greater frequency equating to more opposition. Two formulae expressing the number of Ohms of reactance for these components are shown here:

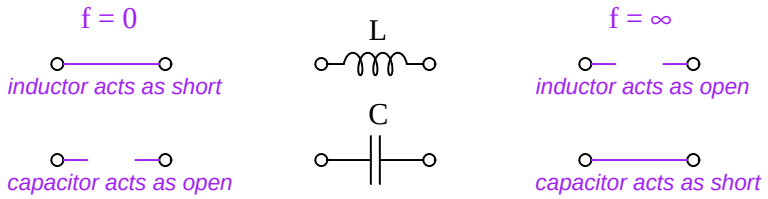
$$X_L = 2\pi f L \qquad\qquad X_C = \frac{1}{2\pi f C}$$

Where,
X_L = Reactance of the inductance (Ohms)
f = Frequency of the waveforms (Hertz, or cycles per second)
L = Inductance (Henrys)
X_C = Reactance of the capacitance (Ohms)
C = Capacitance (Farads)

AC circuit calculations in general tend to be math-intensive, and filter circuit analysis even more so. For this tutorial we will rely heavily on a problem-solving technique called *limiting cases*, where we simplify a quantitative problem into a qualitative problem by considering the effects of some parameter taken to *extreme* limits. The simplifying power of this technique will become rather obvious by example.

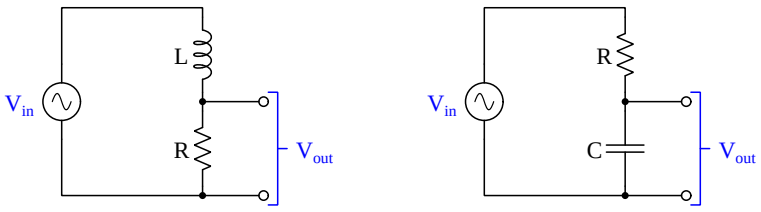
Consider the response of an inductor to limiting cases of frequency: zero frequency (DC) and infinitely[4] high frequency (AC). According to the inductive reactance formula $X_L = 2\pi f L$ an inductor will mimic a short-circuit when energized by DC (i.e. $X_L = 0$ when $f = 0$) but will mimic an open-circuit when energized by AC of infinitely high frequency (i.e. $X_L = \infty$ when $f = \infty$). Capacitors are just the opposite (i.e. $X_C = \infty$ when $f = 0$; $X_C = 0$ when $f = \infty$).

If we need to determine the response of a reactive circuit as frequency decreases, we may take the limiting case of a frequency decrease (all the way to zero) by replacing all inductors with shorts and all capacitors with opens. Likewise, to determine circuit response as frequency increases, we may take the limiting case of a frequency increase (all the way to infinity) by replacing all inductors with opens and all capacitors with shorts. Opening or shorting components in a circuit generally simplifies that circuit, hence the problem-solving value of the "limiting cases" technique when applied to the frequency response of filter circuits.



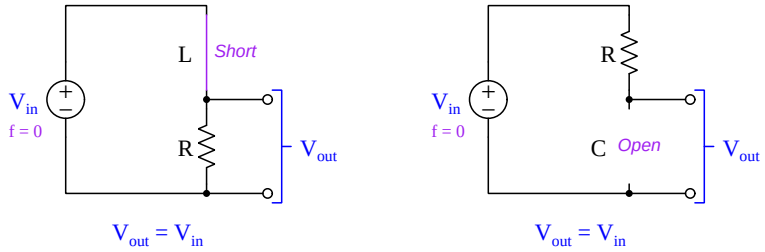
The following two circuits are both called *low-pass* filters for reasons which will soon become clear. You may think of them as *frequency-dependent voltage divider networks*:

Low-pass filter circuits

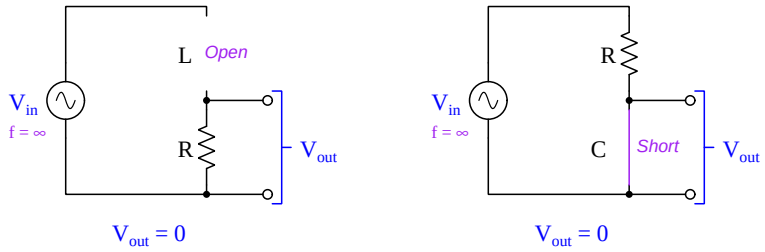


[4] Infinity simply refers to a quantity that is larger than anything imaginable. As I like to tell my students, *infinity is bigger than big, huger than huge.*

To see how these circuits respond as frequency decreases, we will analyze them in the limiting-case condition of zero frequency (DC):



Next, we will check their response as frequency increases by analyzing them in the limiting-case condition of infinite frequency:

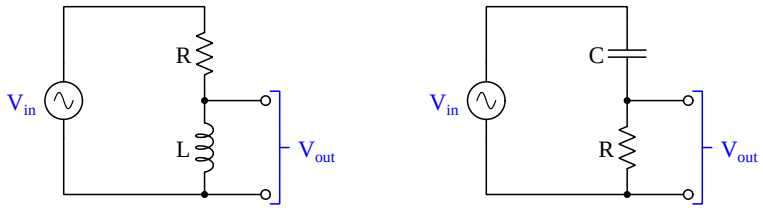


Both circuits exhibit the same fundamental behavior under these limiting-case conditions: they fully *pass* the input signal to the output terminals when frequency is zero, but fully *block* the input signal when frequency is infinitely high. This is why these circuits are called *low-pass filters*. Such filter circuits would be useful for passing DC power while blocking AC noise picked up along lengths of cable, or for accentuating bass tones over treble tones in an audio system.

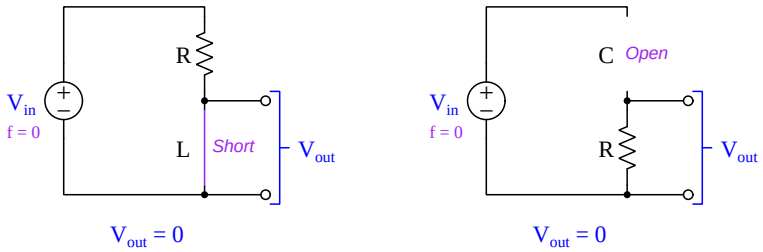
Remember that these are *limiting-case* analyses, and so these conclusions of full signal versus zero signal at the output terminals only apply to the extremes of zero and infinite frequency. For any non-zero, finite frequency value the output voltage will lie somewhere between zero and full input voltage.

The exact opposite type of filter circuit may be constructed simply by swapping the positions of the two components. Once again, we will analyze each circuit under the two limiting-case conditions of zero and infinite signal frequency:

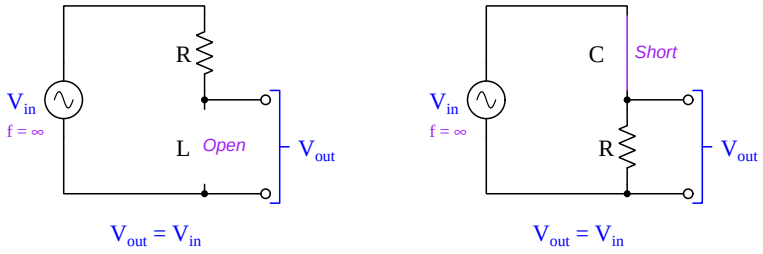
High-pass filter circuits



First, considering the limiting-case condition of zero frequency (DC):



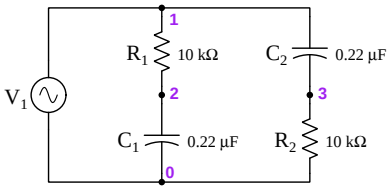
Next, considering the limiting-case condition of infinite frequency:



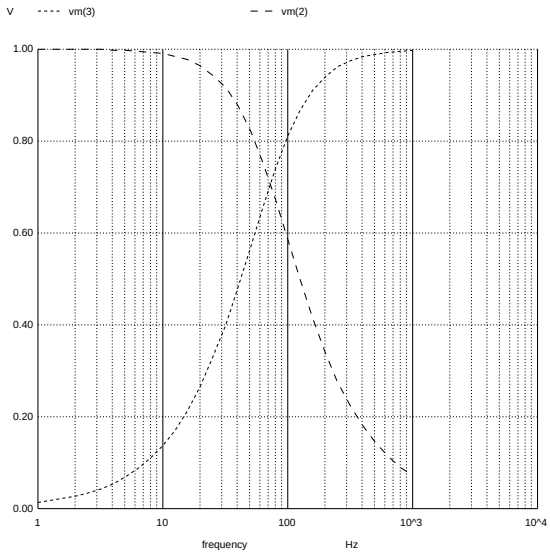
Once again we see how both circuits exhibit the same fundamental behavior under these limiting-case conditions: they fully *pass* the input signal to the output terminals when frequency is infinitely high, but fully *block* the input signal when frequency is zero. This is why these circuits are called *high-pass filters*. Such filter circuits would be useful for accentuating treble tones over bass tones in an audio system, or for separating AC signals intentionally superimposed on DC power wiring.

3.4 Bode plots

If we compute the output voltage of any filter circuit and plot that as a function of frequency, we obtain what is known as a *Bode plot*. The following plots, generated using NGSPICE circuit simulation software, show the frequency response of a low-pass filter and of a high-pass filter, each one comprised of a 10 kΩ resistor and a 0.22 μF capacitor powered by a 1 Volt AC source with a frequency sweeping from 1 Hz to 1000 Hz:




```
* Low-pass R1 and C1, out = node 2
* High-pass R2 and C2, out = node 3
v1 1 0 ac 1
r1 1 2 10000
c1 2 0 0.22e-6
r2 3 0 10000
c2 1 3 0.22e-6
.ac dec 10 1 1k
.plot ac vm(2) vm(3)
.end
```

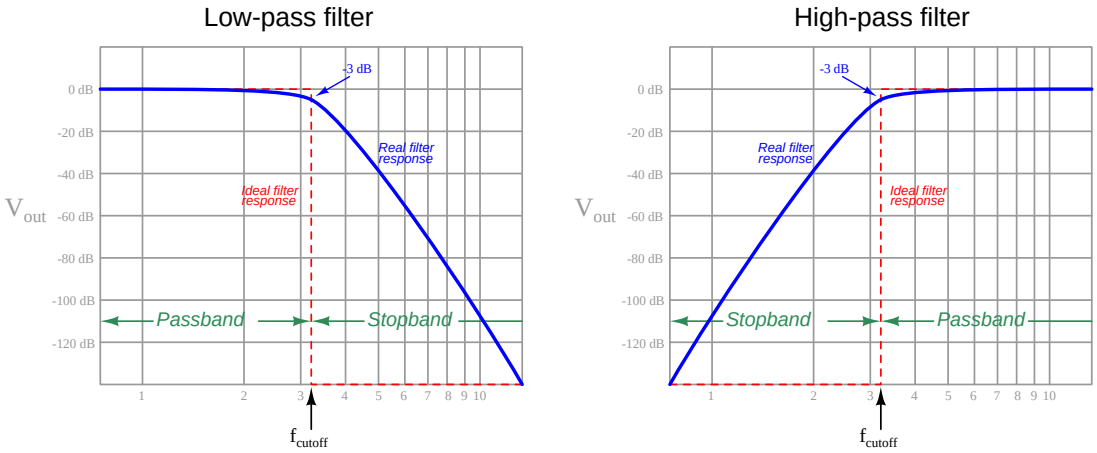


You will note that both of these demonstration filter networks use *capacitors* rather than inductors as the reactive element. The reason for this is that inductors tend to have worse parasitic properties such as wire resistance and turn-to-turn capacitance than capacitors: capacitors are simply *purer* reactive components than inductors and for this reason behave closer to ideal.

Note how the output signal for the low-pass filter (node 2 / node 0 voltage) reaches a peak of nearly 1 Volt and a low point of just less than 0.1 Volt, while the high-pass filter's output signal (node 3 / node 0 voltage) begins at a non-zero value and almost reaches 1 Volt at its high point. Neither filter circuit ever *perfectly* passes or blocks the signal.

Despite the imperfections of real filter circuits, it is useful to rate them as having a certain *cutoff frequency* so we will be able to practically apply them to real applications. Accepted convention for cutoff frequency is that frequency resulting in the output signal having *half the power* (i.e. -3 dB power attenuation) of the input signal, corresponding to an output voltage $\frac{\sqrt{2}}{2}$ of the input voltage. This happens to be the point at which the two simple filter circuits' output values cross when overlaid on the same Bode plot. Closely examining the Bode plot, we see this happens to be approximately 72 Hz. For simple RC and LR filters having only two components, cutoff frequency is that point at which $X = R$ in the circuit. For an RC circuit where $X_C = \frac{1}{2\pi fC}$, cutoff frequency may be calculated as $f_c = \frac{1}{2\pi RC}$, which yields 72.3 Hz for the filters shown above. For an LR circuit where $X_L = 2\pi fL$, cutoff frequency may be calculated as $f_c = \frac{R}{2\pi L}$.

The following Bode plots show the response of real and ideal filter networks, both low-pass (left) and high-pass (right). An ideal low-pass filter passes all signals below the cutoff frequency and completely blocks all signals above the cutoff frequency, as shown by the dashed red lines in the left-hand image. However, a real low-pass filter has a curved response as shown by the blue trace in the left-hand image. We see the same ideal/real contrast for high-pass filters in the right-hand Bode plot:

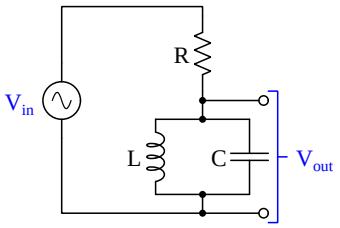


Cutoff frequency divides the spectrum into two frequency bands: the *passband* (representing all signal frequency values able to make it through the filter network), and the *stopband* (representing all frequencies blocked by the filter network). Ideal filter response is easy to understand, with the step-response Bode plot representing a "brick wall" through which no inappropriate signal may pass. In this case, cutoff frequency is a clear and unambiguous threshold marking the transition between passband and stopband. Real filters, however, cannot muster this idealized "brick wall" response, and so the transition between passband and stopband is necessarily gradual. This means we must define cutoff frequency at some arbitrary point between 100% signal passage and 0% signal passage. Historically this has been defined as the frequency at which the signal attenuates by -3 dB (i.e. output voltage is $\frac{\sqrt{2}}{2}$ of input voltage). As previously mentioned, this happens to be the frequency value where $X = R$ in a simple two-component reactive-resistive filter network.

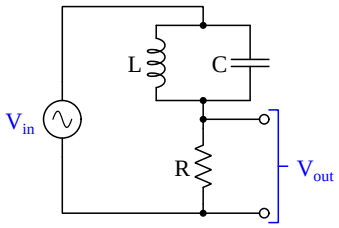
3.5 LC resonant filters

If we combine capacitors and inductors together in the same circuit to form *resonant* networks, we may create another class of filter circuit[5] capable of passing or blocking a specific range of frequencies not bound by zero or infinity:

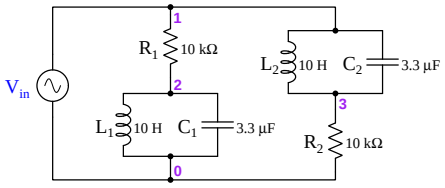
Band-pass filter circuit



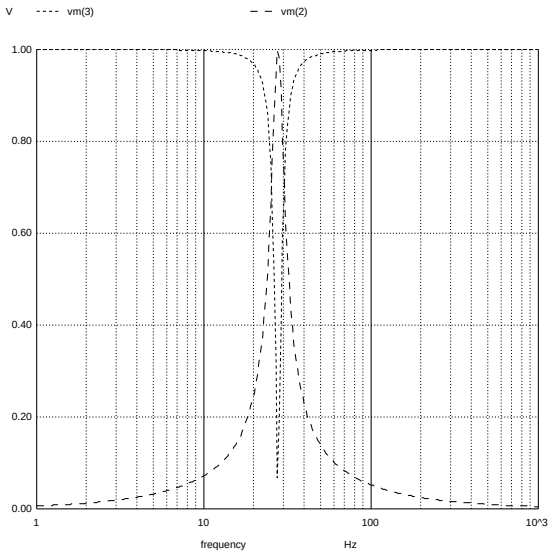
Band-stop filter circuit



The band-pass and band-stop filter circuits shown here exploit the phenomenon of *parallel resonance* between a capacitor and an inductor, where their combined impedance approaches infinity at their resonant frequency predicted by the formula $f_r = \frac{1}{2\pi\sqrt{LC}}$.




```
* Band-pass R1, C1, and L1 ; out = node 2
* Band-stop R2, C2, and L2 ; out = node 3
v1 1 0 ac 1
r1 1 2 10000
c1 2 0 3.3e-6
l1 2 0 10
r2 3 0 10000
c2 1 3 3.3e-6
l2 1 3 10
.ac dec 100 1 1k
.plot ac vm(2) vm(3)
.end
```

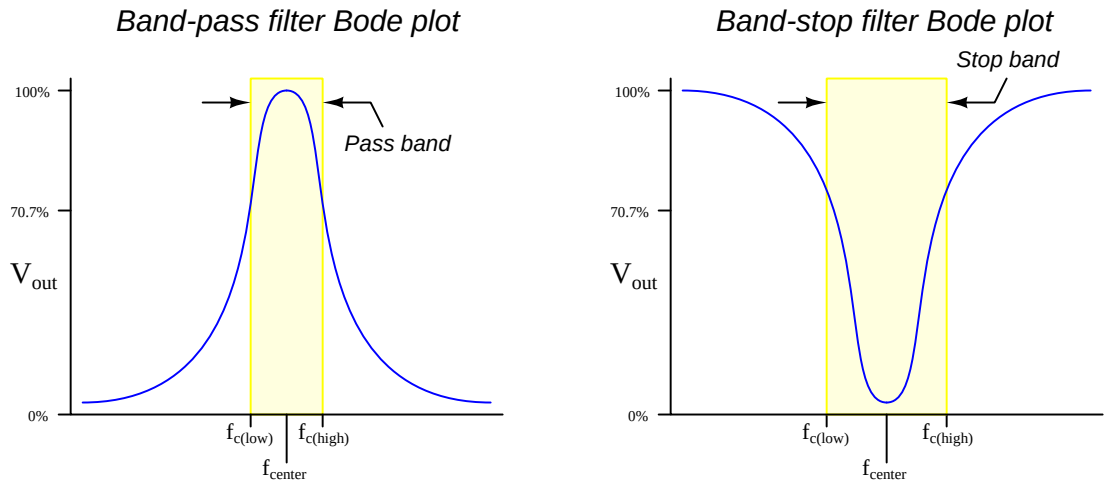


Note how each of these filter circuits "peaks" at the resonant frequency of 27.7 Hz predicted by the combination of a 10 Henry inductor and a 3.3 microFarad capacitor. The band-pass filter nearly outputs 100% of the signal at this frequency, and the band-stop filter nearly outputs 0% at this same frequency. Band-pass filters are useful for applications such as selecting one radio broadcast

[5] Parallel resonance is not the only means of creating a band-type filter. We may also use *series* resonance, as well as use combinations of non-resonant high- and low-pass filter networks to form either band-pass or band-stop filters.

signal out of a myriad of other signals in the same area. Band-stop filters (sometimes called *notch* filters) are useful for blocking noise of known frequency (e.g. 60 Hz AC power line interference) while passing all other signal frequencies.

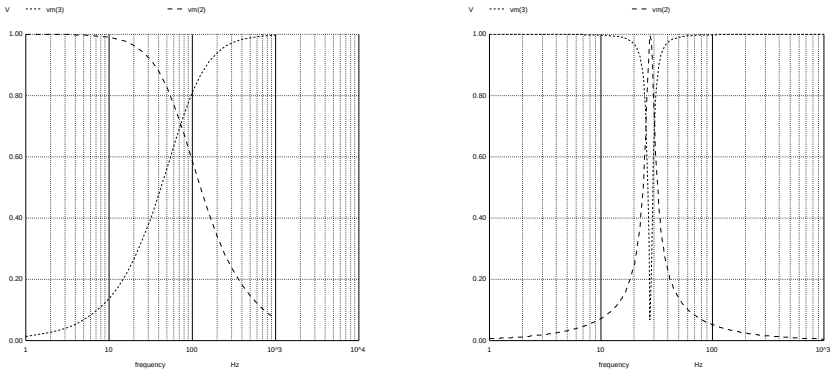
Note also how the output signals for these two filters cross at the same amplitude: approximately 70.7% of V_{in} (i.e. $\frac{\sqrt{2}}{2}V_{in}$) which is the conventionally-accepted definition of cutoff frequency. With *band* style filters, each circuit has two cutoff points: a low cutoff and a high cutoff. For the band-pass filter, any frequency between these two cutoff values is within the *passband* of the filter; for the band-stop filter, any frequency between the two cutoff values is within the *stopband*.



Not all band-pass and band-stop filters exhibit the same bandwidth (i.e. the difference between the high and low cutoff frequencies) – some filters provide a *narrower* bandwidth than others, which may be useful in applications where the filter must precisely discriminate a narrow range of frequencies over all the rest. The narrowness, or *selectivity* of a band-type filter may be quantified as a *quality factor* (Q), expressed as the ratio of center frequency to bandwidth as given by the formula $Q = \frac{f_{center}}{f_{c(high)} - f_{c(low)}}$. This quality factor is a function of the amount of resistance in the filter circuit: resonant filter circuits with little resistance have high quality factors and narrow bandwidths; filter circuits with much resistance have low quality factors and wide bandwidths.

3.6 Roll-off

Selectivity is closely related to another concept in filter circuits called *roll-off*. The "roll-off" for any filter circuit is the steepness of its Bode plot as the circuit transitions from the passband to the stopband of its frequency range. If you compare the SPICE-generated Bode plots for the RC high-pass and low-pass networks (left, below) versus the SPICE-generated Bode plots for the two LC resonant filters (right, below), you will notice how much steeper the LC filters' plots are compared to the RC filters' plots:



Despite the fact that the RC and LC filter networks are all performing different filtering functions, it is still fair to compare their roll-off rates as a measure of how sharply they "cut off" unwanted signal frequencies. When plotted on a graph with logarithmic axis scales (e.g. decibels[6] for signal strength, octaves or decades for frequency[7]) these slopes have a mostly-linear profile, and therefore roll-off is specified in units of *dB per octave* or *dB per decade*.

Steeper roll-off may be obtained from a filter network by adding filter stages. For example, a low-pass filter having a roll-off of 20 dB/decade staged with an identical filter having the same roll-off rate yields a low-pass filter with an aggregate roll-off of 40 dB/decade. It is also possible to steepen the roll-off of a filter by using complementary reactive elements, for example a low-pass filter design using a series L and parallel C instead of a series R and parallel C.

Generally speaking, more aggressive roll-off comes at the expense of a "flat" response in the filter's passband: that is to say, efforts to steepen roll-off usually result in "weird" behavior in the filter's passband, often with peaks and valleys rather than a flat response throughout. However, in some cases it is more important to have a filter that cuts off sharply than a filter with a uniform response to all signal frequencies within the desired band.

[6] A decibel (dB) is a logarithmic measure of a power ratio, defined as $\text{dB} = 10 \log \frac{P_1}{P_2}$. A decibel figure of 0 dB represents a power ratio of 1, while +10 dB represents a power ratio of 10 and −10 dB a power ratio of $\frac{1}{10}$ Furthermore, +20 dB represents a power ratio of 100 and −20 dB represents a power ratio of $\frac{1}{100}$.

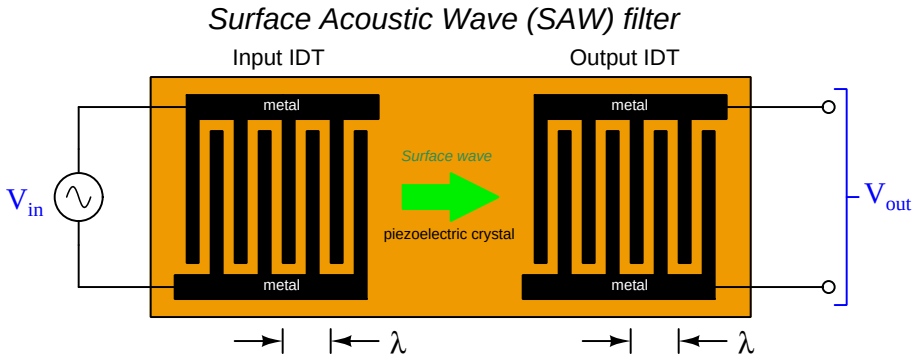
[7] An *octave* is a doubling of frequency, while a *decade* is a ten-fold change in frequency. For example, one octave away from 1 kHz would be 500 Hz (lower) and 2 kHz (higher), while one decade away from 1 kHz would be 100 Hz (lower) and 10 kHz (higher).

3.7 Mechanical-electrical filters

Resonance is a general principle useful for filtering, but it is not limited to combinations of electrical inductance and capacitance. *Mechanical* resonance can be exploited for filtering purposes, and one interesting example of this is an electronic component called a *quartz crystal*. Quartz is a compound exhibiting piezoelectricity, which means a quartz crystal generates voltage when physically stressed, and will experience physical strain (displacement) with an applied voltage. A suitably cut quartz crystal, like any physical object possessing both mass and elasticity, has a fundamental resonant frequency at which it will sustain vibrations. Owing to the piezoelectric nature of quartz crystals, this mechanical resonance translates into electrical resonance: by attaching metal electrodes to the surface of the crystal, it behaves much like an LC circuit with a fixed resonant frequency.

Quartz crystals exhibit high selectivity (e.g. high roll-off rates between passbands and stopbands) and their resonant frequencies are extremely stable over time, which makes them well suited for precision filters and oscillator circuits. By cutting the crystal small enough, the resonant frequency may be set in the megahertz (MHz) range, making them suitable for radio-frequency circuits and extremely precise clock circuits.

A similar technology called *Surface Acoustic Wave* (SAW) filtering works by inducing and detecting mechanical vibrations on the surface of a piezoelectric material by using a micro-machined network of metal electrodes called IDTs (Inter-Digital Transducers) attached to the material's surface at two locations. AC voltage applied between one set of electrodes induces surface waves in the material, which are received by another set of metal electrodes where those surface waves generate an AC voltage signal:

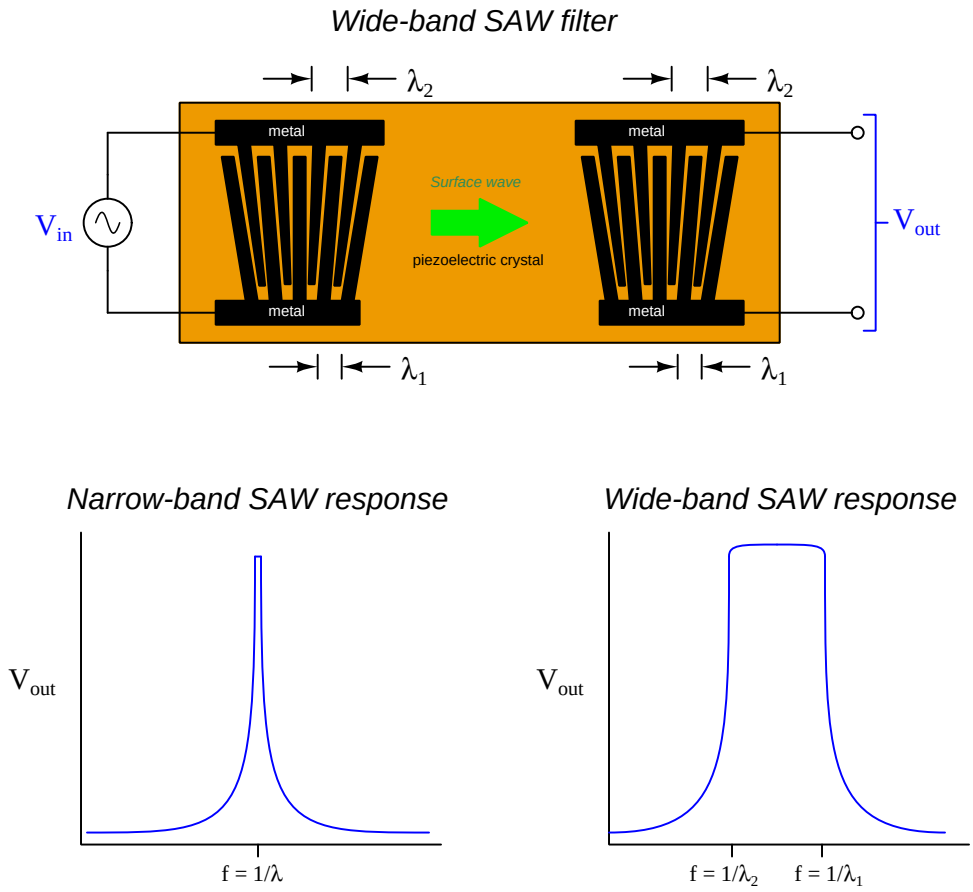


SAW filters naturally act as band-pass filters, where the ratio $\frac{V_{out}}{V_{in}}$ reaches its maximum value at one particular frequency. That frequency depends on the spacing between the metal "fingers" of each IDT, where that distance needs to equal the wavelength (λ) of the acoustic wave through the piezoelectric material[8]. SAW filters are highly selective, and for high frequencies they may be manufactured in very compact packages, which makes them well-suited for radio receiver circuit

[8] Since these vibrations are mechanical in nature, they travel through the crystal at that material's *speed of sound*. Therefore, the wavelength of these acoustic waves is much shorter than the wavelength of that same signal if it took the form of an electromagnetic wave traveling through space or along a cable at the *speed of light*. Typical speeds of sound through SAW substrate materials ranges from about 3,100 meters/second to about 4,800 meters/second; by contrast, the speed of light through vacuum or air is about 300,000,000 meters/second (nearly 10^5 times faster).

applications. These devices replaced LC filter networks used in television receivers as early as the 1980's.

The fact that a SAW filter's pass frequency is defined primarily by the inter-electrode spacing rather than by the bulk properties of the substrate material makes possible the construction of *wide-band* filter networks. One easy-to-understand method of achieving this goal is to construct each IDT in a "fan" shape so that a range of electrode spacings (wavelengths, λ) exist within the same device:



In the example shown here, the SAW filter has a passband extending from $\frac{1}{\lambda_2}$ to $\frac{1}{\lambda_1}$. Again, this capability makes SAW filters ideal for certain radio communication applications where the communications happen over a range of different signal frequencies.

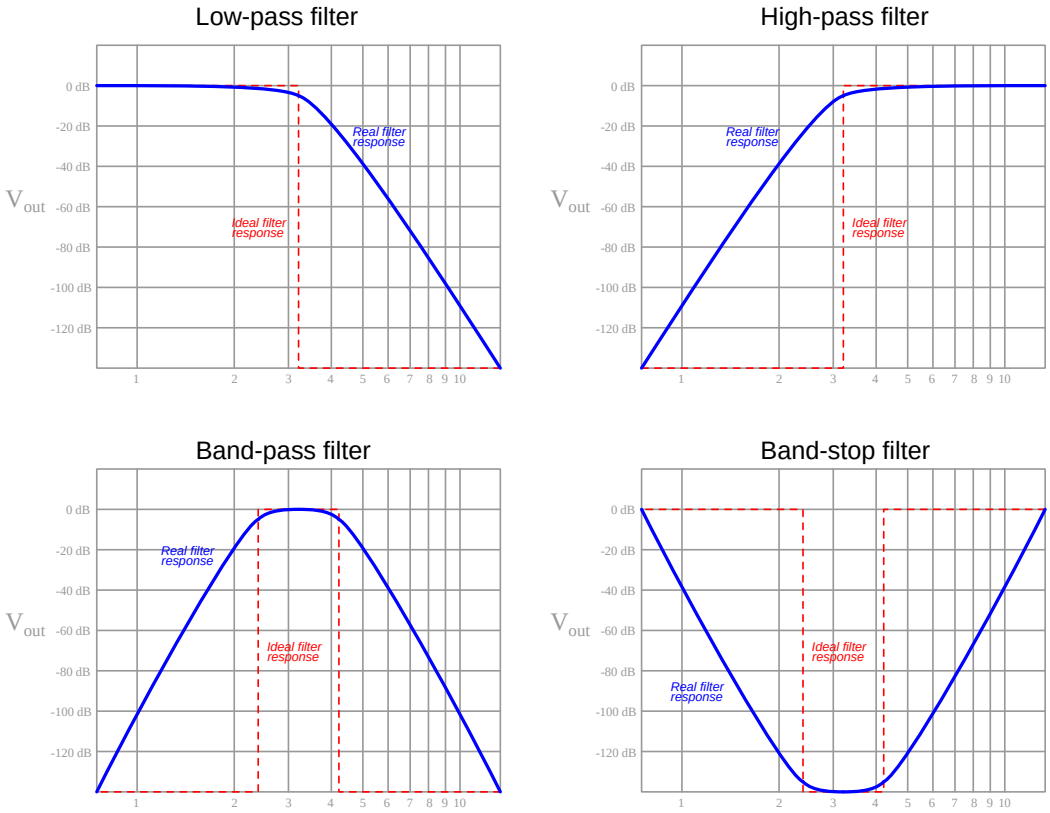
A limitation of mechanical-electrical filters is relatively low power capability compared to reactive (RC, LR, LC) filters. Since all filters rely at some point on the storage of energy within the filter network, the amount of electrical power a filter is able to pass through is a direct function of how much energy it is able to store over the period of a cycle, and filters exploiting microscopic motion of piezoelectric materials simply cannot store much energy at all. This fact limits quartz and

SAW filters to small-signal applications such as radio receiver circuitry, as opposed to high-power applications such as radio transmitters where reactive (usually LC) filters predominate.

Another interesting property of SAW networks is that the time delay between input and output signals is a function of the physical distance between the input and output IDTs. This makes them useful as *delay lines* in addition to being band-pass filters. A "delay line" does exactly what its name suggests: delays the arrival of a signal from input to output on the SAW device, acting in essence as a very long transmission line. However, unlike a transmission line where the signal travels as an electromagnetic wave at the speed of light, the wave in a SAW travels at the (much slower!) speed of sound through the piezoelectric material. This slower wave velocity means the distance between IDTs may be thousands of times shorter than the necessary length of a transmission line to implement the same delay time.

3.8 Summary of filter types

To summarize, any elementary filter circuit may be classified as one of four types, based on the range of frequencies it is designed to pass. In the following Bode plots we see the ideal "brick wall[9]" response of each filter type (dashed red) overlaid on more realistic filter response curves (bold blue):



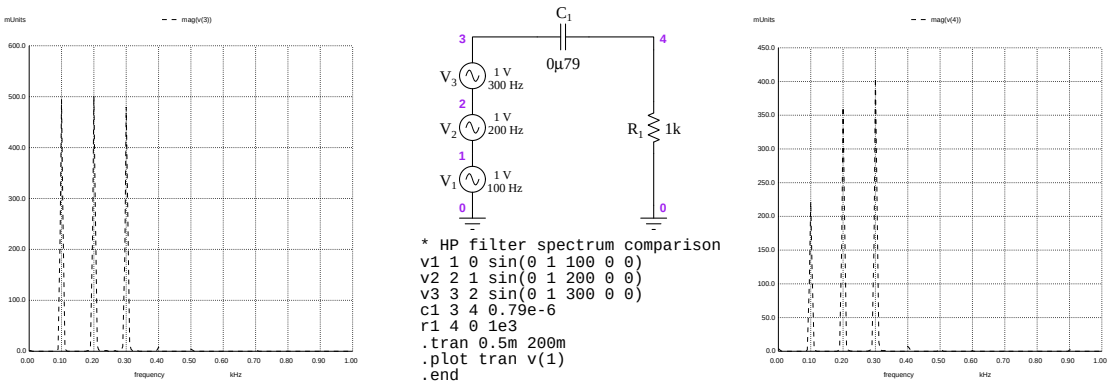
Cutoff frequency is defined as that frequency value at which the signal is reduced to $\frac{\sqrt{2}}{2}$ of its full strength, based on the output signal magnitude of a simple RC or LR filter network when $X = R$. Any frequency resulting in an output signal stronger than this is within the filter's *passband*, with all other frequencies lying in the filter's *stopband*. *Roll-off* is the rate at which a filter's cutoff changes with frequency, essentially the "steepness" of its Bode plot. Filters with high roll-off are more selective, but typically that increase in selectivity is realized only by unequal passage of signals at slightly different frequencies within the filter's "passband" (i.e. a passband response that is not flat). An ideal filter has a perfectly flat passband and infinite roll-off.

[9]This rather colorful description of ideal filter response evokes the image of an impenetrable wall completely stopping the passage of certain signal frequencies.

3.9 Filtering complex signals

As mentioned at the beginning of this tutorial, filters are typically used to selectively screen certain frequencies from complex AC signals possessing a mixture of different frequencies. Thus, it is useful to view filtered and unfiltered signals in the *frequency domain* to understand the effects of filter networks on complex signals.

In a sense Bode plots are already frequency-domain graphs, since their horizontal axes express frequency. However, a Bode plot shows the response of a filter *network* to all frequencies, rather than being a measurement of a *particular signal*. Below is a SPICE simulation[10] showing a simple high-pass RC filter network with its input signal (consisting of an equal blend of 100 Hz, 200 Hz, and 300 Hz sinusoidal voltages, shown on the left) and its output signal (consisting of a filtered blend of 100 Hz, 200 Hz, and 300 Hz peaks, shown on the right):



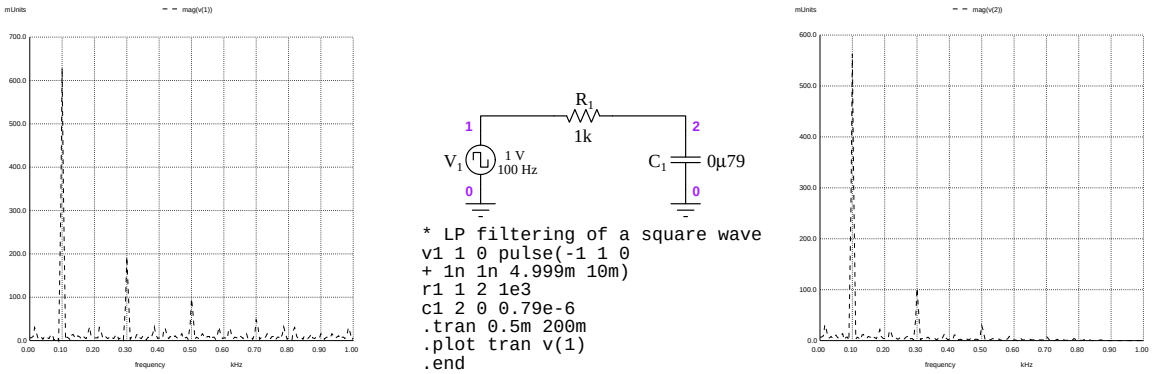
As you can see, the input signal spectrum measured between node 3 and ground consists of three equal-amplitude peaks, one at 100 Hz, one at 200 Hz, and another at 300 Hz. The filter circuit consists of one 0.79 μF capacitor and one 1 kΩ resistor, giving a cutoff frequency of 201.46 Hz. The output signal spectrum measured between node 4 and ground consists of three peaks as well, but of differing amplitude owing to the filter's high-pass characteristic: much more of the 300 Hz peak passes through the filter than either the 200 Hz or 100 Hz peaks.

Also evident from this simulation is the fact that "cutoff" for a filter circuit is a somewhat misleading term. One might expect a high-pass filter with $f_c = 201.46$ Hz to completely cut off all frequencies below 201.46 Hz, but as we can see from the output signal spectrum this is not the case. Remember that "cutoff" is defined for practical filter networks as the point at which a signal of that

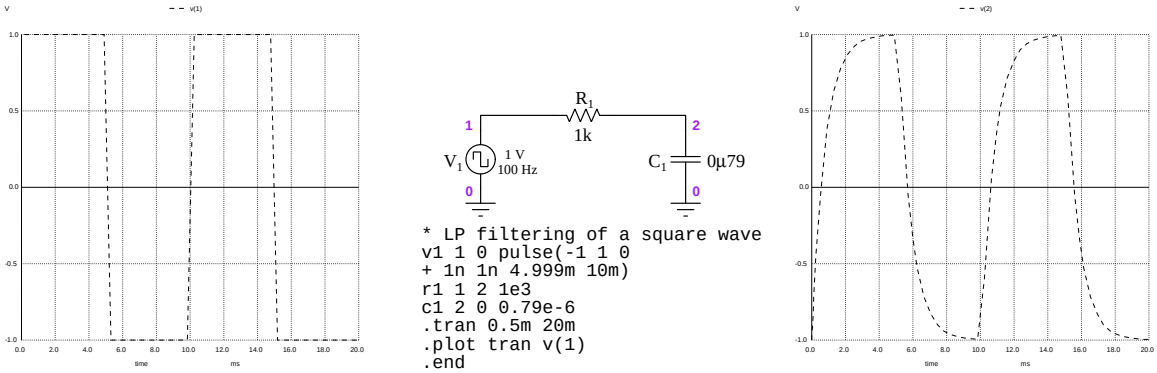
[10] The sequence of NGSPICE commands used to generate these spectra (after reading netlist file with the `source` command and "running" the simulation with the `run` command) are as follows: `linearize v(1) ; fft v(1) ; plot mag(v(1))`. Parameters in the transient analysis line ("`tran`") of the SPICE netlist define the resolution and domain of the FFT analysis: interval time (in this case, 0.5 milliseconds) adjusts the FFT domain, with smaller intervals resulting in a wider span of frequencies showing on the horizontal axis; and ending time (in this case, 200 milliseconds) determining the resolution of the FFT plot, with longer time resulting in finer resolution. In fact, when simulated with an ending time value of 1 second, the peaks were all so thin as to be barely viewable on the plots! I had to reduce the total transient analysis time to 200 milliseconds in order to make the peaks appear a bit wider at their bases and therefore easier to see on the plots. Remember that a signal consisting of three perfect sinusoids will have a spectrum consisting of three *vertical lines* as peaks.

frequency becomes attenuated to 70.7% of its original amplitude. Perfect filtration does not exist, either in fluid systems or electrical systems.

One of the fundamental principles of waves is that every wave, regardless of shape, is mathematically equivalent to a series of sinusoids. We call these sinusoids *harmonics*, and for periodic waves they are integer-multiples of the wave's fundamental frequency. Square waves are a textbook example of this principle: a perfect square wave is the sum of an infinite series of odd-harmonic sine waves[11]. If we pass a non-sinusoidal wave through a filter, some of its harmonics will be attenuated more than others, resulting in a *re-shaping* of the waveform. This concept is illustrated by the following SPICE simulation, showing the harmonic spectrum of a square wave before (left) and after (right) passing through a low-pass filter:



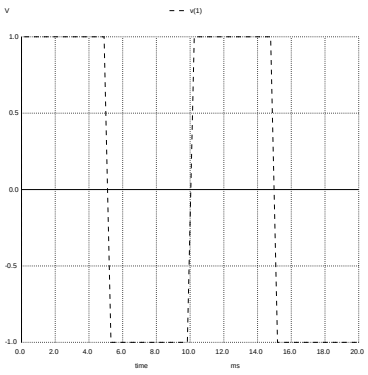
The wave's new shape is clearly evident when we compare the input and output signals in the *time domain*:

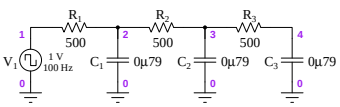


Note how the output waveform appears much more rounded than a square wave due to the higher-order harmonics being screened out by the filter network. This raises an interesting question: can we re-shape a square wave into a sine wave by filtering out *all* harmonics but the fundamental?

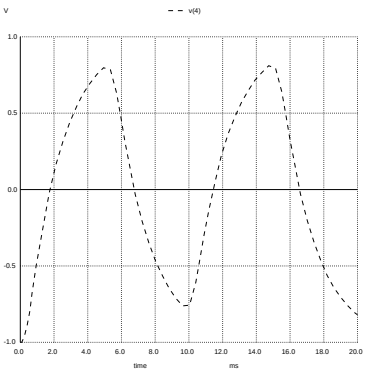
[11] Square wave $= \sin \omega t + \frac{1}{3} \sin 3\omega t + \frac{1}{5} \sin 5\omega t + \frac{1}{7} \sin 7\omega t + \cdots + \frac{1}{n} \sin n\omega t$, where ω is the fundamental frequency of the square wave in radians per second, assuming sine values calculated from radians rather than degrees.

Of course, a perfect low-pass filter would be required to completely eliminate all but the first harmonic from a square wave, and as we know perfect filters do not exist. However, we may *approach* perfection by using a filter circuit with more elements in it[12], to achieve a steeper roll-off:






```
* LP filtering of a square wave
v1 1 0 pulse(-1 1 0
+ 1n 1n 4.999m 10m)
r1 1 2 500
c1 2 0 0.79e-6
r2 2 3 500
c2 3 0 0.79e-6
r3 3 4 500
c3 4 0 0.79e-6
.tran 0.5m 20m
.plot tran v(4)
.end
```



As we can see in the right-hand oscillograph, the wave-shape is beginning to appear more like a sine wave with the added filter stages.

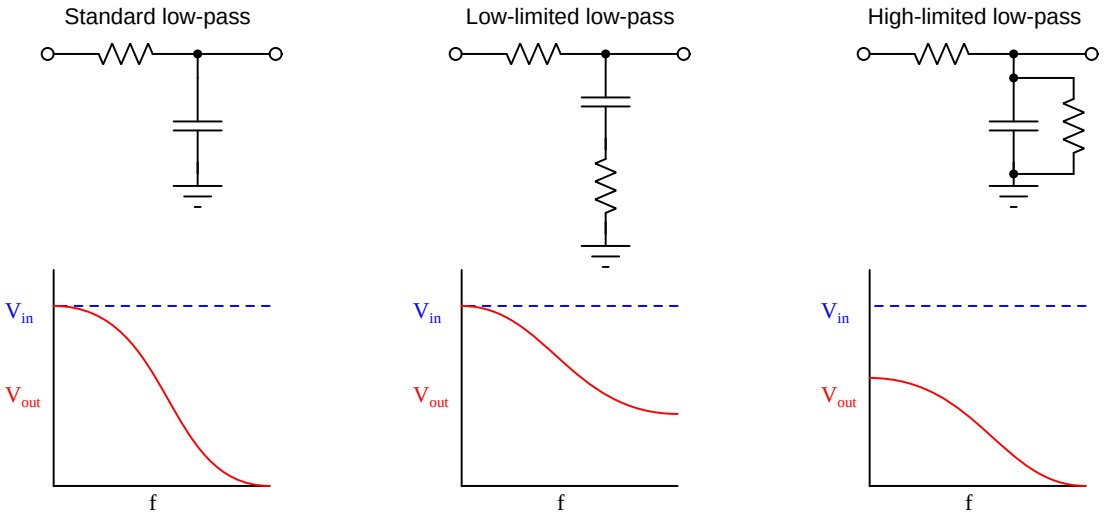
In summary, we have seen how combinations of resistance, capacitance, and/or inductance function as frequency-dependent voltage dividers, and in so doing have the ability to selectively pass or attenuate signals of differing frequency. We have seen how simple filter circuits may be characterized as having a *cutoff frequency* defined as the frequency at which an incident signal becomes attenuated to 70.7% of its original amplitude. Furthermore, we have explored the use of *Bode plots* to describe the behavior of a filter over a range of frequencies, and discussed the concept of *roll-off* which describes how "sharply" a filter cuts off unwanted frequencies.

Practical applications for filters include signal separation and wave-shaping. The former refers to the separation of signals we do not wish to belong together, for example when filtering out "noise" that is combined with some signal of interest. The latter refers to the altering of a non-sinusoidal signal's harmonic composition, for example to alter the signal's wave-shape as viewed in the time domain.

[12] Multi-stage filter design is an extremely complicated topic, and this particular filter network represents a fairly crude way to achieve steeper roll-off. Usually one would design a low-pass filter network employing both capacitive and inductive elements rather than simply add more RC stages. The astute reader will note different resistor values used in this filter than before, to help account for the "loading" each successive stage places on the previous stage. Suffice to say, this multi-stage filter is a *proof-of-concept* circuit rather than a practical example of how to achieve "sharper" low-pass filtering.

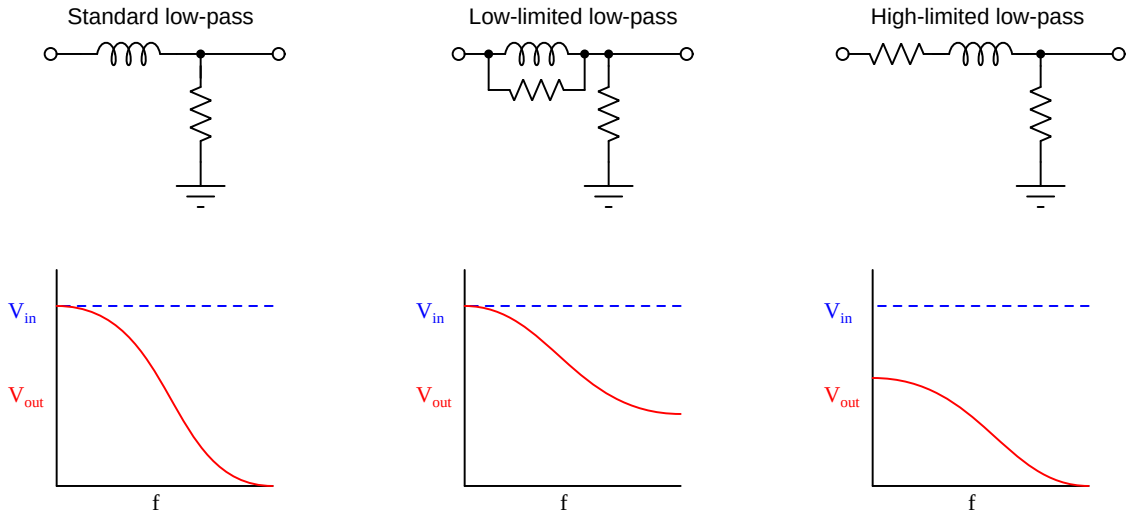
3.10 Output-limited filter networks

An interesting variation on the theme of simple resistance-reactance filter networks is one where we intentionally install an extra resistor in order to limit the output signal either to some maximum value less than source's voltage or to some minimum value greater than zero:



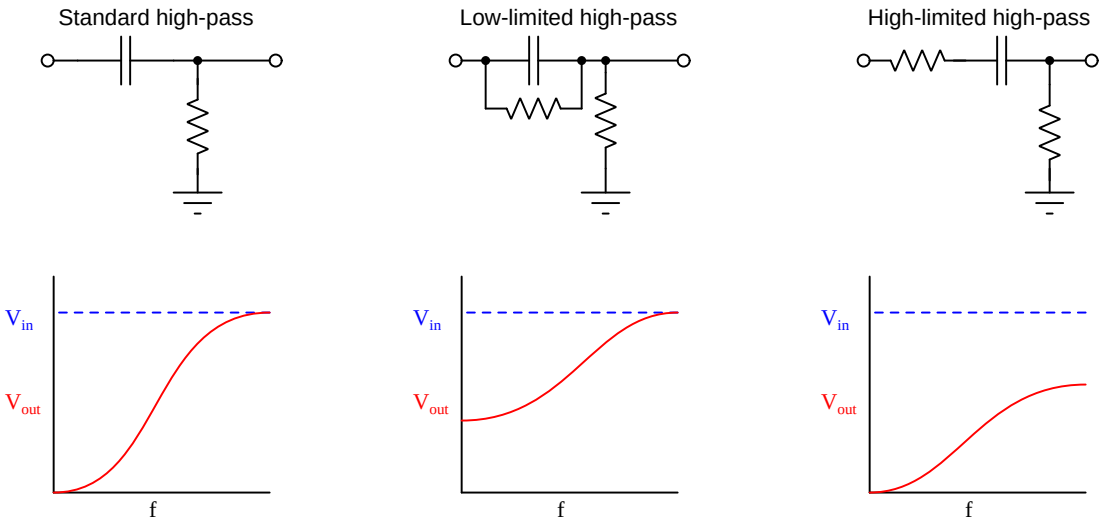
Our familiar problem-solving technique of *limiting cases* works just as well to explore the behavior of these limited filter networks as they do on the standard low-pass network: at very low frequencies the capacitor will function as an *open*, and at very high frequencies the capacitor will act like a *short*. For the low-limited low-pass filter this means full output at DC and a resistively voltage-divided output at infinite frequency; for the high-limited low-pass filter it means a resistively voltage-divided output at DC and zero output at infinite frequency.

Of course, the same general principle extends to inductor-based low-pass filters as well:

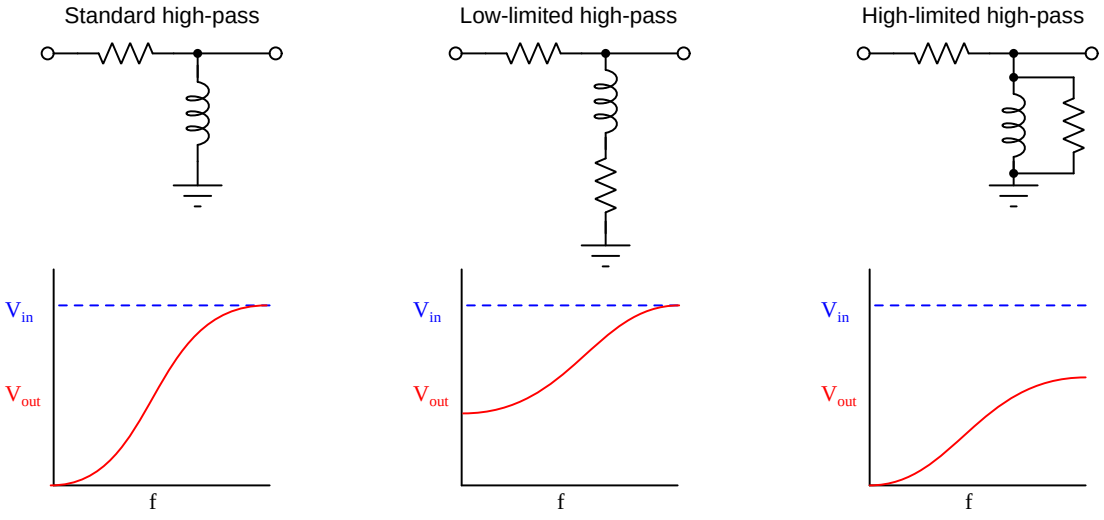


At very low frequencies the inductor will act as a short, and at very high frequencies as an open. For the low-limited low-pass filter this means full output at DC and a resistively voltage-divided output at infinite frequency. For the high-limited low-pass filter this means a resistively voltage-divided output at DC and zero output at infinite frequency.

High-pass filter networks may also be constructed in similar manner by strategically adding a second resistor:



And, of course, inductor-based versions of these same output-limited high-pass circuits exist as well:

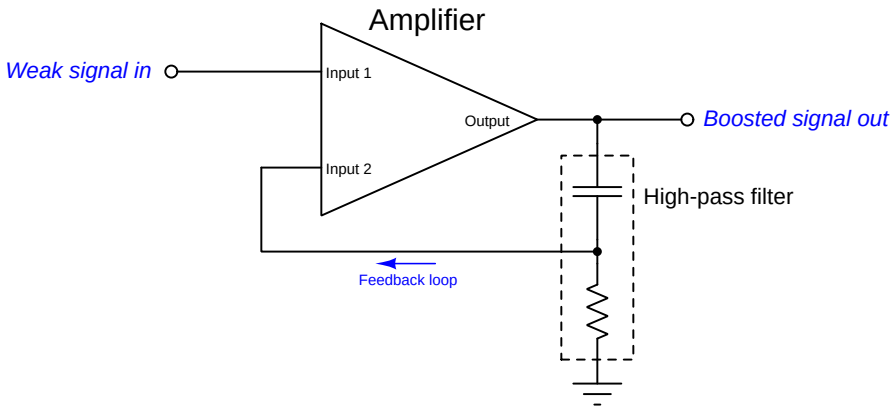


In every one of these output-limited filter networks, the output limiting is accomplished by the added resistor limiting the extremes to which the reactive component's effective impedance may reach. That additional resistor either prevents the reactive component from completely shorting or completely opening two nodes in the network, and in doing so either limits the output voltage from

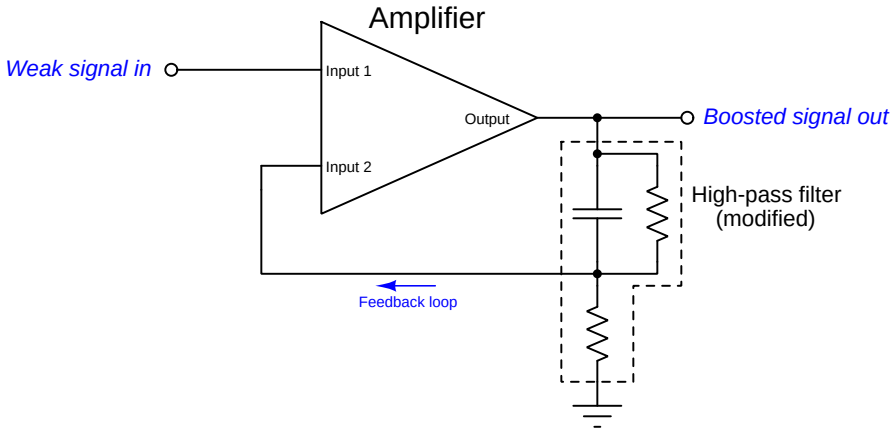
ever attaining full source voltage (high-limited) or limits the output voltage from ever decreasing to zero (low-limited).

Output-limited filter networks are frequently used in electronic systems employing *negative feedback*, where a signal-boosting system (called an *amplifier*) re-directs part of the boosted output signal back around to the amplifier's input as a means of improving certain performance criteria such as stability. Often a filter network is placed within this feedback "loop" to favor some frequencies over others, but if some of those un-favored frequencies are *completely* cut off by the filter the system may become unstable for those excluded frequencies.

Here is a visual example of an amplifier system using negative feedback with standard high-pass filtering in the feedback loop:



Unfortunately, this feedback system will cause trouble for the amplifier because it completely blocks all DC (zero-frequency) signals from being fed back to Input 2. This will mean that the amplifier cannot naturally stabilize itself for DC signals using negative feedback as it can for AC signals. The solution is to modify the filter network to have a low-limited characteristic so that at least some of the DC signals are able to reach Input 2:



Chapter 4

Historical References

This chapter is where you will find references to historical texts and technologies related to the module's topic.

Readers may wonder why historical references might be included in any modern lesson on a subject. Why dwell on old ideas and obsolete technologies? One answer to this question is that the initial discoveries and early applications of scientific principles typically present those principles in forms that are unusually easy to grasp. Anyone who first discovers a new principle must necessarily do so from a perspective of ignorance (i.e. if you truly *discover* something yourself, it means you must have come to that discovery with no prior knowledge of it and no hints from others knowledgeable in it), and in so doing the discoverer lacks any hindsight or advantage that might have otherwise come from a more advanced perspective. Thus, discoverers are forced to think and express themselves in less-advanced terms, and this often makes their explanations more readily accessible to others who, like the discoverer, comes to this idea with no prior knowledge. Furthermore, early discoverers often faced the daunting challenge of explaining their new and complex ideas to a naturally skeptical scientific community, and this pressure incentivized clear and compelling communication. As James Clerk Maxwell eloquently stated in the Preface to his book *A Treatise on Electricity and Magnetism* written in 1873,

> It is of great advantage to the student of any subject to read the original memoirs on that subject, for science is always most completely assimilated when it is in its nascent state . . . `[page xi]`

Furthermore, grasping the historical context of technological discoveries is important for understanding how science intersects with culture and civilization, which is ever important because new discoveries and new applications of existing discoveries will always continue to impact our lives. One will often find themselves impressed by the ingenuity of previous generations, and by the high degree of refinement to which now-obsolete technologies were once raised. There is much to learn and much inspiration to be drawn from the technological past, and to the inquisitive mind these historical references are treasures waiting to be (re)-discovered.

4.1 Wave screens

Charles Proteus Steinmetz was an electrical engineer employed for many years by the General Electric Company in New York. He was widely recognized as a genius in this field, and did much to elevate the mathematical rigor of electrical engineering. In his book *Theory and Calculation of Electric Circuits* first published in 1917 he describes the use of capacitance and inductance to form filtering circuits which he referred to as *wave screens* useful for separating alternating and direct current components of any pulsating electrical signal:

> **78.** By "wave screens" the separation of pulsating currents into their alternating and their continuous component, or the separation of complex alternating currents – and thus voltages – into their constituent harmonics can be accomplished, and inversely, the combination of alternating and continuous currents or voltages into resultant complex alternating or pulsating currents.
>
> The simplest arrangement of such a wave screen for separating, or combining, alternating and continuous currents into pulsating ones, is the combination, in shunt with each other, of a capacity, C, and an inductance, L, as shown in Fig. 75. If, then, a pulsating voltage, e, is impressed upon the system, the pulsating current, i, produced by it divides, as the continuous component can not pass through the condenser, C, and the alternating component is barred by the inductance, L, the more completely, the higher this inductance. Thus the current, i_1, in the apparatus, A, is a true alternating current, while the current, i_0, in the apparatus, C, is a slightly pulsating direct current. [page 156]

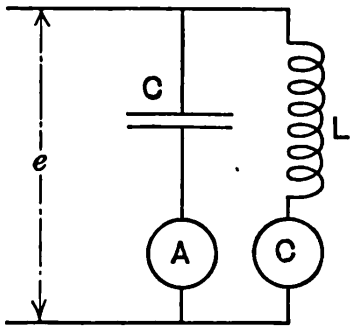


FIG. 75.

In this illustration A and C each represent electrical ammeters registering current through their respective branches of the parallel ("shunt") network.

On the next page, Steinmetz describes the use of series LC resonance to form band-pass filters useful for separating various harmonic[1] frequencies from a complex AC signal:

> Wave screens based on resonance for a definite frequency by series connection of capacity and inductance, can be used to separate the current of this frequency from a complex current or voltage wave, such as those given in Figs. 56 to 63, and thus can be used for the separation of complex waves into their components, by "harmonic analysis."
>
> Thus in Fig. 76, if the successive capacities and inductances are chosen such that

$$2\pi f L_1 = \frac{1}{2\pi f C_1} \ ,$$

$$6\pi f L_3 = \frac{1}{6\pi f C_3} \ ,$$

$$10\pi f L_5 = \frac{1}{10\pi f C_5} \ ,$$

$$2n\pi f L_n = \frac{1}{2\pi f n C_n}$$

> where f = frequency of the fundamental wave. [page 180]

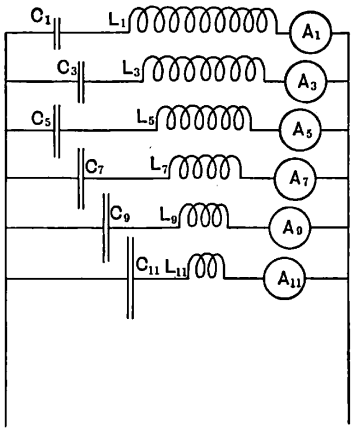


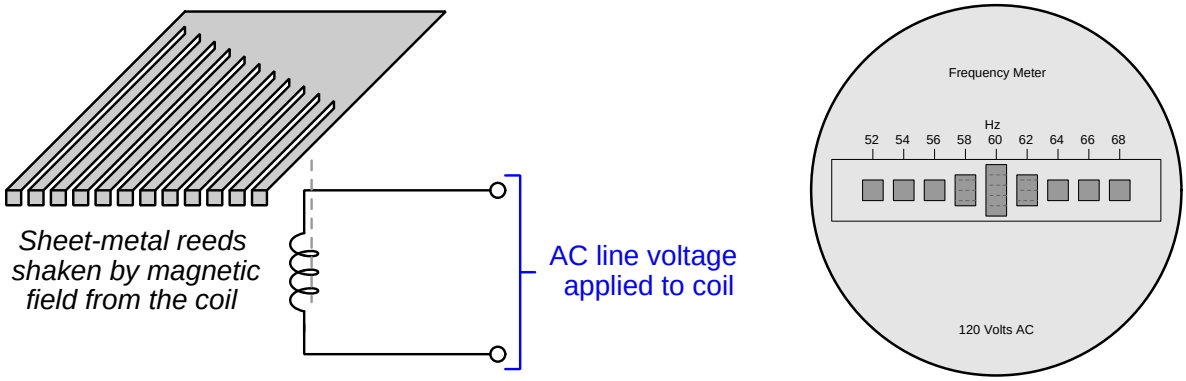
Fig. 76.

Steinmetz's conception of multiple band-pass filter networks tuned to resonate with respective harmonics of a known fundamental frequency, each one connected to its own ammeter to register the strength of each harmonic, is analogous to obsolete vibrating-reed frequency meters with their multiple reeds tuned to different frequencies.

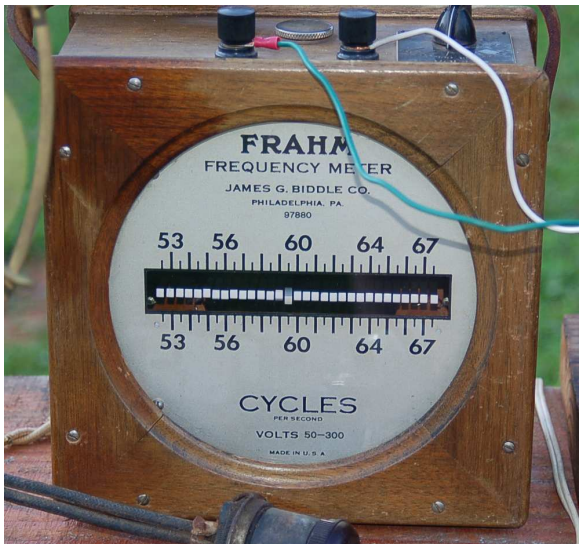
[1] As mathematically proven by Fourier, any periodic wave of any shape whatsoever is mathematically equivalent to the sum of a set of sinusoidal waves having frequency values equal to whole-numbered multiples of the fundamental frequency of the complex wave. For example, a complex-shaped waveform having a frequency of 45 Hz may consist of a 45 Hz "fundamental" sinusoid (the first harmonic) plus other sinusoidal waves of specific amplitudes having frequencies of 90 Hz (the second harmonic), 135 Hz (the third harmonic), 180 Hz (the fourth harmonic), etc.

4.2 Vibrating-reed meters as spectrum analyzers

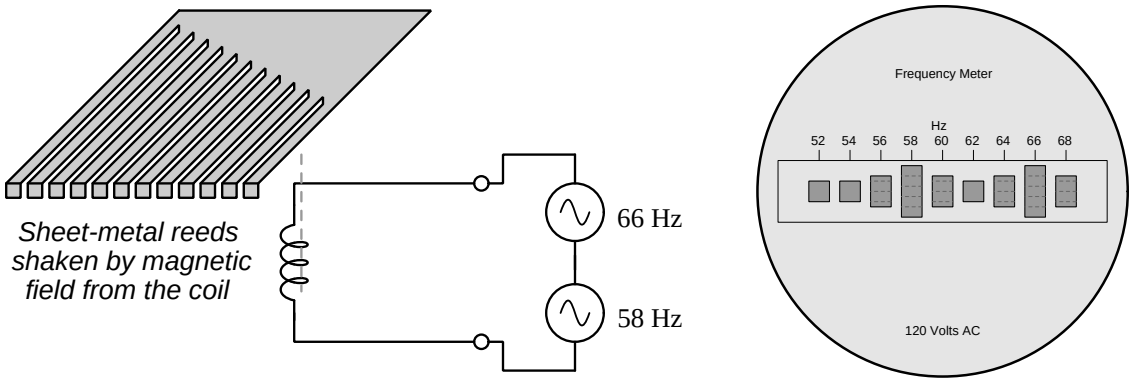
An obsolete technology for measuring the frequency of AC electric power is the *vibrating-reed* frequency meter. This meter consisted of a set of metal reeds made of spring-steel, each reed cut to a different length, and all of them excited by the magnetic field from an electromagnet coil energized by the AC line voltage to be measured. The illustration on the left shows the internal construction of the meter with its metal reed array and coil, while the illustration on the right shows what the meter looks like when energized by 60 Hz AC:



If the AC line frequency is 60 Hz, the reed tuned to resonate at that frequency will vibrate at the greatest amplitude, making the end of that reed appear "taller" as it rapidly shakes up and down. All the other reeds vibrate as well, but none as strongly as the reed resonating with the line frequency. A photograph of a real vibrating-reed frequency meter appears here, connected to an AC generator outputting approximately 59.5 Hz:



As crude a measuring instrument as a vibrating-reed frequency meter is, it actually functions as a sort of spectrum analyzer. A pure sinusoidal AC voltage has but a single frequency in its spectrum, and the moving reed ends reveal the outline of this spectrum: a single peak at the line frequency. If we were to connect a pair of AC generators in series and run them at different speeds to form a superposition of two sinusoidal voltages, we would see *two* reeds vibrate strongly on the face of such a frequency meter: two peaks in the spectrum, one for each generator's output:



For readers familiar with *filter* networks, a good way to model each of the metal reeds in such a meter is as a *mechanical band-pass filter*. Each reed has its own resonant frequency dictated by its mass, length, and elasticity. Adjusting the length of each reed is the simplest way to "tune" each of the reeds to the desired resonant frequency, which is why each reed in a meter such as this has a different length.

Chapter 5

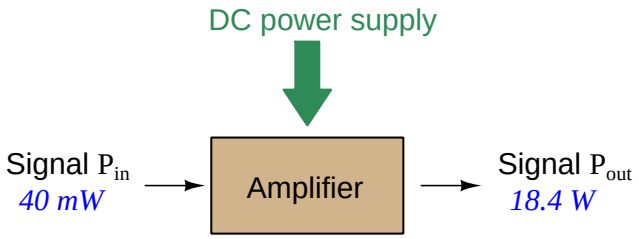
Derivations and Technical References

This chapter is where you will find mathematical derivations too detailed to include in the tutorial, and/or tables and other technical reference material.

5.1 Decibels

One of the mathematical tools popularly used to gauge increases and decreases of electrical power is the *common logarithm*, expressed as a measurement unit called the *decibel*. The basic idea of decibels is to express a ratio of two electrical power quantities in logarithmic terms. Every time you see the unit of "decibel" you can think: *this is an expression of how much greater (or how much smaller) one power is to another*. The only question is which two powers are being compared.

Electronic amplifiers are a type of electrical system where comparisons of power are useful. Students of electronics learn to compare the output power of an amplifier against the input power as a unitless ratio, called a *gain*. Take for example an electronic amplifier with a signal input of 40 milliWatts and a signal output of 18.4 Watts:



$$\text{Gain} = \frac{P_{out}}{P_{in}} = \frac{18.4\ W}{40\ mW} = 460$$

An alternative way to express the gain of this amplifier is to do so using the unit of the *Bel*, defined as the common logarithm of the gain ratio:

$$\log\left(\frac{P_{out}}{P_{in}}\right) = \log\left(\frac{18.4\ \text{W}}{40\ \text{mW}}\right) = 2.66276\ \text{B}$$

When you see an amplifier gain expressed in the unit of "Bel", it's really just a way of saying "The output signal coming from this amplifier is x powers of ten greater than the input signal." An amplifier exhibiting a gain of 1 Bel outputs 10 times as much power as the input signal. An amplifier with a gain of 2 Bels boosts the input signal by a factor of 100. The amplifier shown above, with a gain of 2.66276 Bels, boosts the input signal 460-fold.

At some point in technological history it was decided that the "Bel" (B) was too large and cumbersome, and so it became common to express powers in fractions of a Bel instead: the *deci*Bel (1 dB = $\frac{1}{10}$ of a Bel). Therefore, this is the form of formula you will commonly see for expressing electrical signal power gains or losses:

$$\text{dB} = 10 \log\left(\frac{P_{out}}{P_{in}}\right)$$

The gain of our hypothetical electronic amplifier, therefore, would be more commonly expressed as 26.6276 dB rather than 2.66276 B, although either expression is technically valid[1].

[1] It is interesting to note that although the "Bel" is a metric unit, it is seldom if ever used without the metric prefix

An operation students often struggle with is converting a decibel figure back into a ratio, since the concept of logarithms seems to be universally perplexing. Here I will demonstrate how to algebraically manipulate the decibel formula to solve for the power ratio given a dB figure.

First, we will begin with the decibel formula as given, solving for a value in decibels given a power ratio:

$$\text{dB} = 10 \log(\text{Ratio})$$

If we wish to solve for the ratio, we must "undo" all the mathematical operations surrounding that variable. One way to determine how to do this is to reverse the order of operations we would follow if we knew the ratio and were solving for the dB value. After calculating the ratio, we would then take the logarithm of that value, and then multiply that logarithm by 10: start with the ratio, then take the logarithm, then multiply last. To un-do these operations and solve for the ratio, we must un-do each of these operations in reverse order. First, we must un-do the multiplication (by dividing by 10):

$$\frac{\text{dB}}{10} = \frac{10 \log(\text{Ratio})}{10}$$

$$\frac{\text{dB}}{10} = \log(\text{Ratio})$$

Next, we must un-do the logarithm function by applying its mathematical inverse to both sides of the formula – making each expression a power of 10:

$$10^{\frac{\text{dB}}{10}} = 10^{\log(\text{Ratio})}$$

$$10^{\frac{\text{dB}}{10}} = \text{Ratio}$$

To test our algebra, we can take the previous decibel value for our hypothetical amplifier and see if this new formula yields the original gain ratio:

$$\text{Ratio} = 10^{\frac{26.6276 \text{ dB}}{10}}$$

$$\text{Ratio} = 10^{2.66276 \text{ B}}$$

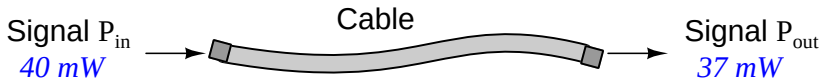
$$\text{Ratio} = 460$$

Sure enough, we arrive at the correct gain ratio of 460, starting with the decibel gain figure of 26.6276 dB.

"deci" ($\frac{1}{10}$). One could express powers in microbels, megabels, or any other metric prefix desired, but it is never done in industry: only the decibel is used.

We may also use decibels to express power *losses* in addition to power *gains*. There are many practical applications of this in signaling systems, both electronic and optical. One such application is *filtering*, where a "filter" circuit screens out certain components of the signal while letting others pass through (e.g. the bass or treble control for an audio system). Another application is *attenuation*, where the entirety of a signal is reduced in magnitude (e.g. the volume control for an audio system).

We will explore yet another application of signal power reduction as a case study for decibels: *cable loss*. Cables designed to convey signals over long distances are not perfect conduits of energy, as some of the signal's energy is inevitably lost along the way. This is true for different types of signals, electrical and optical being two popular examples. In the following illustration we see a signal cable losing power along its length[2], such that the power out is less than the power in:



$$\text{Gain} = \frac{P_{out}}{P_{in}} = \frac{37\ mW}{40\ mW} = 0.925$$

$$10 \log \left(\frac{P_{out}}{P_{in}} \right) = 10 \log \left(\frac{37 \text{ mW}}{40 \text{ mW}} \right) = -0.3386 \text{ dB}$$

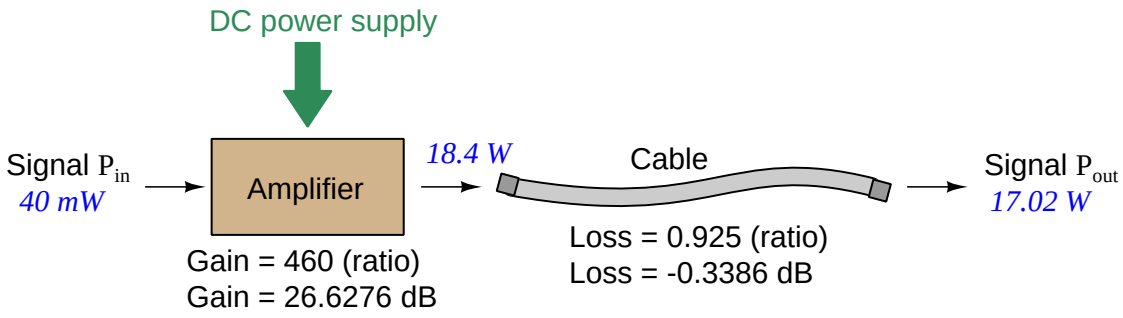
Contrasting this result against the previous result (with the amplifier) we see a very important property of decibel figures: any power *gain* is expressed as a *positive* decibel value, while any power *loss* is expressed as a *negative* decibel value. Any component outputting the exact same power as it takes in will exhibit a "gain" value of 0 dB (equivalent to a gain *ratio* of 1).

Remember that Bels and decibels are nothing more than logarithmic expressions of "greater than" and "less than". Positive values represent powers that are *greater* while negative values represent powers that are *lesser*. Zero Bel or decibel values represent *no change* (neither gain nor loss) in power.

A couple of simple decibel values are useful to remember for approximations, where you need to quickly estimate decibel values from power ratios (or vice-versa). Each addition or subtraction of 10 dB exactly represents a 10-fold multiplication or division of power ratio: e.g. +20 dB represents a power ratio gain of $10 \times 10 = 100$, whereas −30 dB represents a power ratio reduction of $\frac{1}{10} \times \frac{1}{10} \times \frac{1}{10} = \frac{1}{1000}$. Each addition or subtraction of 3 dB approximately represents a 2-fold multiplication or division or power ratio: e.g. +6 dB is approximately equal to a power ratio gain of $2 \times 2 = 4$, whereas −12 dB is approximately equal to a power ratio reduction of $\frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} = \frac{1}{16}$. We may combine ± 10 dB and ± 3 dB increments to come up with ratios that are products of 10 and 2: e.g. +26 dB is approximately equal to a power ratio gain of $10 \times 10 \times 2 \times 2 = 400$.

[2] For high-frequency signals such as those used in radio communications, the dominant mode of energy dissipation is *dielectric heating*, where the AC electric field between the cable conductors excites the molecules of the conductor insulation. This energy loss manifests as heat, which explains why there is less signal energy present at the load end of the cable than is input at the source end of the cable. For DC and low-frequency AC circuits the dominant mode of energy dissipation is cable conductor resistance, which is typically very small.

Observe what happens if we combine a "gain" component with a "loss" component and calculate the overall power out versus power in:



The overall gain of this amplifier and cable system expressed as a ratio is equal to the *product* of the individual component gain/loss ratios. That is, the gain ratio of the amplifier *multiplied* by the loss ratio of the cable yields the overall power ratio for the system:

$$\text{Overall gain} = \frac{17.02 \text{ W}}{40 \text{ mW}} = (460)(0.925) = 425.5$$

The overall gain may be alternatively expressed as a decibel figure, in which case it is equal to the *sum* of the individual component decibel values. That is, the decibel gain of the amplifier *added* to the decibel loss of the cable yields the overall decibel figure for the system:

$$\text{Overall gain} = 10 \log \left(\frac{17.02 \text{ W}}{40 \text{ mW}} \right) = 26.6276 \text{ dB} + (-0.3386 \text{ dB}) = 26.2890 \text{ dB}$$

It is often useful to be able to estimate decibel values from power ratios and vice-versa. If we take the gain ratio of this amplifier and cable system (425.5) and round it down to 400, we may easily express this gain ratio as an expanded product of 10 and 2:

$$425.5 \approx 400 = (10) \times (10) \times (2) \times (2)$$

Knowing that every 10-fold multiplication of power ratio is an addition of +10 dB, and that every 2-fold multiplication of power is an addition of +3 dB, we may express the expanded product as a sum of decibel values:

$$(10) \times (10) \times (2) \times (2) = (10 \text{ dB}) + (10 \text{ dB}) + (3 \text{ dB}) + (3 \text{ dB}) = 26 \text{ dB}$$

Therefore, our power ratio of 425.5 is approximately equal to +26 decibels.

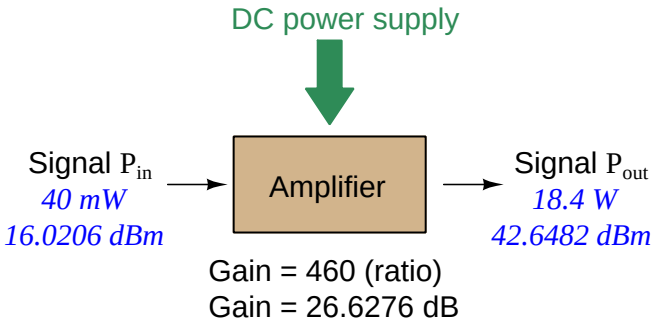
Decibels always represent comparisons of power, but that comparison need not always be P_{out}/P_{in} for a system component. We may also use decibels to express an amount of power compared to some standard reference. If, for example, we wished to express the input power to our hypothetical amplifier (40 milliWatts) using decibels, we could do so by comparing 40 mW against a standard "reference" power of exactly 1 milliWatt. The resulting decibel figure would be written as "dBm" in honor of the 1 *m*illiWatt reference:

$$P_{in} = 10 \log \left(\frac{40 \text{ mW}}{1 \text{ mW}} \right) = 16.0206 \text{ dBm}$$

The unit of "dBm" literally means the amount of dB "greater than" 1 milliWatt. In this case, our input signal of 40 milliWatts is 16.0206 dB greater than a standard reference power of exactly 1 milliWatt. The output power of that amplifier (18.4 Watts) may be expressed in dBm as well:

$$P_{out} = 10 \log \left(\frac{18.4 \text{ W}}{1 \text{ mW}} \right) = 42.6482 \text{ dBm}$$

A signal power of 18.4 Watts is 42.6482 dB greater than a standard reference power of exactly 1 milliWatt, and so it has a decibel value of 42.6482 dBm.



Notice how the output and input powers expressed in dBm relate to the power gain of the amplifier. Taking the input power and simply *adding* the amplifier's gain factor yields the amplifier's output power in dBm:

$$P_{in}(\text{dB}) + P_{gain}(\text{dB}) = P_{out}(\text{dB})$$

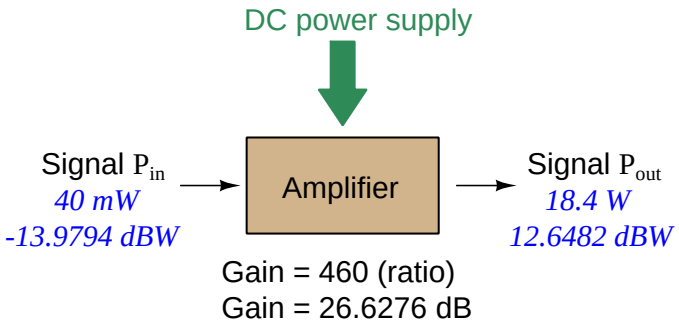
$$16.0206 \text{ dBm} + 26.6276 \text{ dB} = 42.6482 \text{ dBm}$$

An electronic signal that begins 16.0206 dB greater than 1 milliWatt, when boosted by an amplifier gain of 26.6276 dB, will become 42.6482 dB greater than the original reference power of 1 milliWatt.

We may alternatively express all powers in this hypothetical amplifier in reference to a 1-Watt standard power, with the resulting power expressed in units of "dBW" (decibels greater than 1 Watt):

$$P_{in} = 10 \log \left(\frac{40 \text{ mW}}{1 \text{ W}} \right) = -13.9794 \text{ dBW}$$

$$P_{out} = 10 \log \left(\frac{18.4 \text{ W}}{1 \text{ W}} \right) = 12.6482 \text{ dBW}$$



Note how the input power of 40 milliWatts equates to a negative dBW figure because 40 milliWatts is *less* than the 1 Watt reference, and how the output power of 18.4 Watts equates to a positive dBW figure because 18.4 Watts is *more* than the 1 Watt reference. A positive dB figure means "more than" while a negative dB figure means "less than."

Note also how the output and input powers expressed in dBW still relate to the power gain of the amplifier by simple addition, just as they did when previously expressed in units of dBm. Taking the input power in units of dBW and simply *adding* the amplifier's gain factor yields the amplifier's output power in dBW:

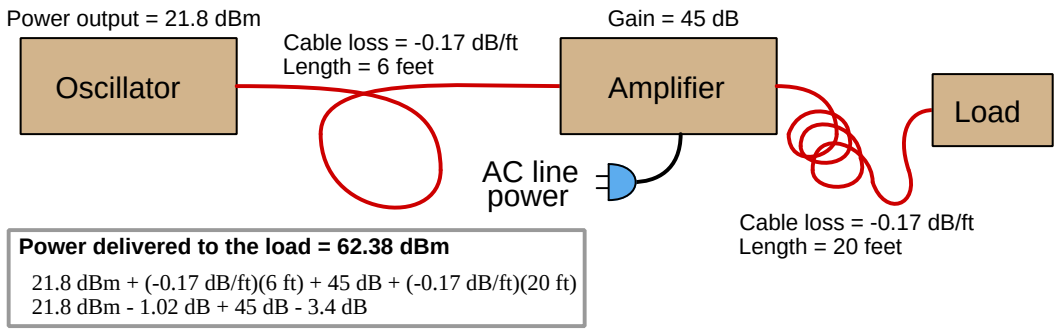
$$P_{in}(\text{dB}) + P_{gain}(\text{dB}) = P_{out}(\text{dB})$$

$$-13.9794 \text{ dBW} + 26.6276 \text{ dB} = 12.6482 \text{ dBW}$$

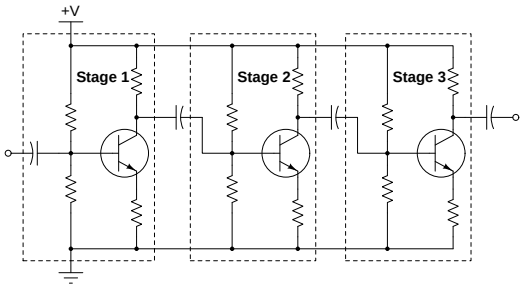
An electronic signal that begins 13.9794 dB less than 1 Watt, when boosted by an amplifier gain of 26.6276 dB, will become 12.6482 dB greater than the original reference power of 1 Watt.

This is one of the major benefits of using decibels to express powers: we may very easily calculate power gains and losses by summing a string of dB figures, each dB figure representing the power gain or power loss of a different system component. Normally, any compounding of *ratios* involves multiplication and/or division of those ratios, but with decibels we may simply add and subtract. One of the interesting mathematical properties of logarithms is that they "transform[3]" one type of problem into a simpler type: in this case, a problem of multiplying ratios into a (simpler) problem of adding decibel figures.

For example, we may express the power dissipated along a cable in terms of decibels per foot; the longer the cable, of course, the more power will be lost this way, all other factors being equal. For example, a radio-frequency signal cable having a loss figure of −0.15 decibels per foot at a signal frequency of 2.4 GHz will suffer −15 dB over 100 feet, and −150 dB over 1000 feet. To illustrate how decibels may be used to calculate power delivered to a load in such a system, accounting for various gains and losses along the way using decibel figures:



A similar application of decibels is found in multi-stage amplifier circuits, where one stage amplifies a signal to be fed into a successive stage to be amplified more. The power gains of these stages, each expressed as a ratio, *multiply* to make the over-all amplifier's power gain (ratio). The power gains of those same stages, each expressed as a decibel figure, *add* to make the over-all amplifier's power gain (dB):



[3]In fact, logarithms are one of the simplest examples of a *transform function*, converting one type of mathematical problem into another type. Other examples of mathematical transform functions used in engineering include the *Fourier transform* (converting a time-domain function into a frequency-domain function) and the *Laplace transform* (converting a differential equation into an algebraic equation).

Another common application of decibels is to express ratios of voltage and/or current rather than power. However, since the unit of the Bel has been defined as an expression of a *power* ratio, we cannot simply substitute V or I for P in any of the formulae we've seen so far.

Suppose an amplifier has a voltage gain of 2 (i.e. V_{out} is twice as large as V_{in}), and we would like to express this gain in decibels. Since decibels are intended to express power gain and not voltage gain, we must figure out how much power gain is equivalent to a voltage gain of two. Obviously, voltage and power are fundamentally different quantities, but if we imagine ourselves connecting a fixed load resistance to the input signal, and then to the output signal, we will realize that load's power dissipation will be more than double when energized by a voltage twice as large. Joule's Law is helpful to determine the exact ratio of power dissipation:

$$P = \frac{V^2}{R}$$

Doubling voltage for any given load resistance results in power *quadrupling* because power is proportional to the square of the voltage applied to a fixed resistance. Using this as the basis for applying decibels to a voltage ratio. Knowing that Joule's Law also declares power is proportional to the square of the current applied to a fixed resistance ($P = I^2R$) means this same mathematical relationship will apply to current gains and reductions as well as voltage gains and reductions:

$$\text{dB} = 10 \log \left(\frac{P_{out}}{P_{in}} \right) = 10 \log \left(\frac{V_{out}}{V_{in}} \right)^2 = 10 \log \left(\frac{I_{out}}{I_{in}} \right)^2$$

An algebraic identity of logarithms is that the logarithm of any quantity raised to a power is equal to that power multiplied by the logarithm of the quantity. Expressed in general terms:

$$\log x^y = y \log x$$

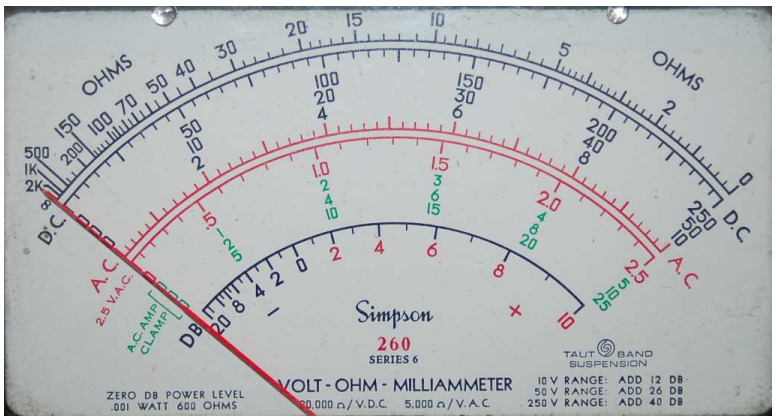
Therefore, we may simplify the decibel formula for voltage gain by removing the "2" power and making it a multiplier:

$$10 \log \left(\frac{V_{out}}{V_{in}} \right)^2 = (2)(10) \log \left(\frac{V_{out}}{V_{in}} \right) = 20 \log \left(\frac{V_{out}}{V_{in}} \right)$$

$$10 \log \left(\frac{I_{out}}{I_{in}} \right)^2 = (2)(10) \log \left(\frac{I_{out}}{I_{in}} \right) = 20 \log \left(\frac{I_{out}}{I_{in}} \right)$$

Thus, we may use decibels to express voltage or current ratios if we simply substitute 20 instead of 10 as the multiplier.

We can see the practicality of using decibels to represent something other than electrical *power* by examining this analog meter face, belonging to a Simpson model 260 VOM (Volt-Ohm-Milliammeter). Note the bottom scale on this meter's face, calibrated in decibels (DB):



Pay attention to the note on decibels written in the lower-left corner of the meter face, where 0 dB is defined as 0.001 Watt dissipated by 600 Ohms. The fact that 0 dB is defined as 1 milliWatt means it should (properly) be labeled dBm rather than dB[4]. A load resistance value is necessary as part of this definition for dB because this meter cannot measure power directly but must infer signal power from measurements of AC *voltage*. Without a specific load resistance, there is no clear relation between voltage and power. 600 Ohms is an old telecommunications standard for audio-frequency AC signals, and continues to be used today for voltage-based decibel measurements of audio-frequency AC signals.

The meter as shown is connected to nothing at all, and so registers 0 Volts AC. This, of course, corresponds to zero power, and it has no corresponding decibel value because the logarithm of zero is mathematically *undefined*[5]. Practically, it means $-\infty$ dB, which is why the needle at the 0 Volt position "falls off" the left-hand end of the dB scale.

Close inspection of the dB scale on this meter face reveals another interesting property of decibels, and that is the *nonlinear* nature of the dB scale. This contrasts starkly against all the voltage and current scales on this meter face which are linear. This nonlinearity is a fundamental property of decibels because it is based on the *logarithm* function which is nonlinear.

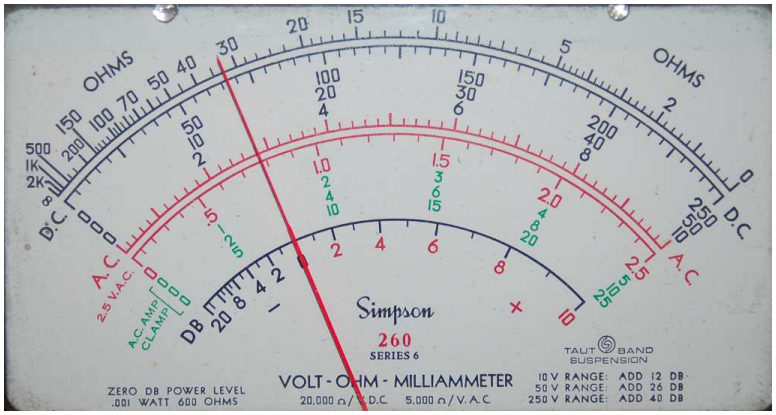
[4] Such mis-labeling is not that uncommon in the profession, the expectation being that the technician or engineer working with the instrument ought to be familiar enough with the concept of decibels to know when dB really means dBm, or dBW, etc.

[5] Your electronic calculator will complain if you attempt to take the logarithm of zero!

Now, we will explore what is necessary to make this meter register 0 dBm (i.e. 1 milliWatt) with an applied AC voltage. 1 milliWatt of power dissipated by 600 Ohms is equivalent to:

$$V = \sqrt{PR} = \sqrt{(0.001)(600)} = 0.7746 \text{ Volts}$$

Setting the VOM to the 2.5 VAC range and applying just enough AC voltage to bring the needle to the 0 dB mark allows us to verify that this is indeed equivalent to just under 0.8 Volts (read on the 2.5 VAC scale):



In the lower-right corner of the meter face we see some notes regarding correction values for decibel measurements when using different AC voltage ranges. The dB scale is read directly when the meter is set on the 2.5 VAC range. When set on the 10 VAC range (i.e. a range *four times* as great), the meter's needle will experience a deflection one-fourth as much as when set to the 2.5 VAC range, and therefore it will point to a lesser (or even negative) value on the dB scale. Converting a voltage ratio of 0.25 into a decibel figure shows us how much less the needle will register on the dB scale when the voltage range is quadrupled:

$$20 \log \left(\frac{2.5}{10} \right) = -12.04 \text{ dB}$$

Therefore, when using the 10 VAC range instead of the 2.5 VAC range, one must add 12 dB to the reading. Likewise, we may prove each of the printed correction factors for the alternative voltage-measurement ranges listed (50 Volt AC range and 250 Volt AC range):

$$20 \log \left(\frac{2.5}{50} \right) = -26.02 \text{ dB}$$

$$20 \log \left(\frac{2.5}{250} \right) = -40.0 \text{ dB}$$

5.2 IEC standard component values

Components such as resistors, inductors, and capacitors are manufactured in several *standard values*, described by IEC standard 60063. Rather than having a single series of standard values, the IEC publishes lists called *E series* based on the number of unique values spanning a single *decade* (i.e. a 10:1 range).

The shortest of these series, called *E3* contains just three values: 10, 22, and 47. The next series is called *E6* with six unique values: 10, 15, 22, 33, 47, and 68. These values represent *significant values* for components, meaning the decimal point may be freely moved to create values spanning multiple decades. For example, "33" simply means one can expect to find components manufactured in values of 33, 3.3, 0.33, and 0.033 as well as 330, 3.3 k, 33 k, etc.

Although this may seem like a strange standard for component manufacturers to follow, there is a compelling logic to it. The terms of each series are closer-spaced at the low end than at the high end, and this allows for *series* and/or *parallel* combinations of components to achieve most any desired value. For example, in the E6 series we only have values with the significant figures 10, 15, 22, 33, 47, and 68, but this doesn't mean we are limited to *total* values with these significant figures. For example, if we needed 80 Ohms of resistance we could connect a 33 Ohm and 47 Ohm resistor together in series. 50 Ohms could be made from two 68 Ohm resistors in parallel (making 34 Ohms) plus a 15 Ohm and 1 Ohm resistor in series.

On the next page is a table showing the four most common E-series specified by IEC standard 60063.

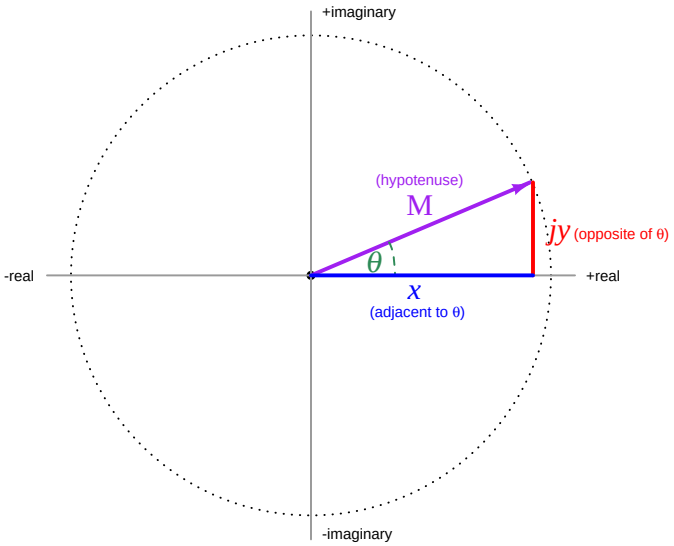
E3	**E6**	**E12**	**E24**
10	10	10	10
			11
		12	12
			13
	15	15	15
			16
		18	18
			20
22	22	22	22
			24
		27	27
			30
	33	33	33
			36
		39	39
			43
47	47	47	47
			51
		56	56
			62
	68	68	68
			75
		82	82
			91

E48, *E96*, and *E192* series are also found in the IEC 60063 standard, used for components with tighter tolerance ratings than typical.

5.3 Complex-number arithmetic

Complex numbers are very useful in AC circuit analysis because each one has the ability to represent both a magnitude and a phase shift between that quantity and some other reference quantity. Despite the existence of electronic calculators and computer software capable of performing arithmetic on complex-number quantities, there are still times when we must perform some calculation on these quantities "by hand". This technical reference reviews the basic arithmetic operations on complex numbers, complete with examples.

Recall that complex numbers may be represented in either *rectangular* or *polar* form, rectangular being a quantity with both a "real" and an "imaginary" component, and polar being a quantity with a magnitude and an angle. Graphically, these two forms relate to the sides of a right triangle:



Rectangular form: $x + jy$ (where $j = \sqrt{-1}$)

Polar form: $M\angle\theta$

To convert from rectangular form to polar form, $M = \sqrt{x^2 + y^2}$ and $\theta = \arctan \frac{y}{x}$

To convert from polar form to rectangular form, $x = M \cos\theta$ and $y = M \sin\theta$

As we will see, addition and subtraction is easiest to do with rectangular-form notation while multiplication and division is easiest to do with polar-form notation. Thus, circuit analysis doing "long-hand" complex-number arithmetic often involves conversions back and forth between rectangular and polar forms in order to set up the quantities before applying Ohm's Law, Kirchhoff's Laws, etc. This can be tedious, and it is also prone to rounding errors. The reader is advised to store all intermediate results in their calculator's memory and recall when needed, rather than re-type quantities and thereby incur rounding errors due to truncation.

5.3.1 Negating complex numbers

The sign of a complex number may be reversed just as easily in rectangular form as in polar form. Rectangular-form negation consists of multiplying -1 through to both the real and imaginary terms. Polar-form negation consists solely of adding 180 degrees to the angle, or alternatively, by reversing the sign of the magnitude and leaving the angle alone.

Example: reverse the sign of $5 - j4$

$$-(5 - j4)$$

$$-5 + j4$$

Example: reverse the sign of $6\angle 30^o$

$$-(6\angle 30^o)$$

$$6\angle 210^o = 6\angle -150^o = -6\angle 30^o$$

5.3.2 Adding complex numbers

Complex numbers are most easily added in *rectangular form*: simply add the real portions and then add the imaginary portions.

Example: add $5 - j4$ to $-1 - j3$

$$(5 - j4) + (-1 - j3)$$

$$(5 + (-1)) + (-j4 + (-j3))$$

$$4 - j7$$

5.3.3 Subtracting complex numbers

Complex numbers are most easily subtracted in *rectangular form*: simply subtract the real portions and then subtract the imaginary portions.

Example: subtract $5 - j4$ from $-1 - j3$

$$(-1 - j3) - (5 - j4)$$

$$(-1 - (5)) + (-j3 - (-j4))$$

$$-6 + j1$$

5.3.4 Multiplying complex numbers

Complex numbers are most easily multiplied in *polar form*: simply multiply the magnitudes and add the angles.

Example: multiply $6\angle 30^o$ by $2\angle -10^o$

$$(6\angle 30^o) \times (2\angle -10^o)$$

$$(6 \times 2)\angle(30^o + (-10^o))$$

$$12\angle 20^o$$

Multiplication of rectangular-form complex numbers less straight-forward then with polar-form numbers, and resembles multiplication of algebraic polynomials:

Example: multiply $5 - j4$ by $-1 - j3$

$$(5 - j4) \times (-1 - j3)$$

$$(5 \times (-1)) + (5 \times (-j3)) + (-j4 \times (-1)) + (-j4 \times (-j3))$$

$$(-5) + (-j15) + (j4) + (j^2 12)$$

$$(-5) + (-j15) + (j4) + ((-1)12)$$

$$(-5) + (-j15) + (j4) + (-12)$$

$$-17 - j11$$

5.3.5 Dividing complex numbers

Complex numbers are most easily divided in *polar form*: simply divide the magnitudes and subtract the angles.

Example: divide $6\angle 30^o$ by $2\angle -10^o$

$$\frac{6\angle 30^o}{2\angle -10^o}$$

$$\frac{6}{2}\angle(30^o - (-10^o))$$

$$3\angle 40^o$$

5.3.6 Reciprocating complex numbers

Reciprocation is division into one, and so complex numbers are reciprocated most easily in *polar form* just as division is best performed in polar form: simply reciprocate the magnitude and negate the angle.

Example: reciprocate $2\angle -10^o$

$$\frac{1}{2\angle -10^o}$$

$$\frac{1}{2}\angle -(-10^o)$$

$$0.5\angle 10^o$$

5.3.7 Calculator tips

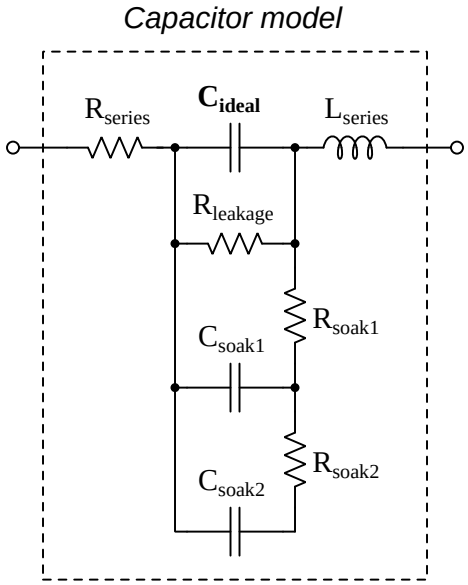
Here is some advice when using calculators to do complex-number arithmetic:

- When manually entering a complex-number value, enclose that value in parentheses. Some calculators struggle to properly perform order-of-operations with complex numbers. For example, some calculators will interpret $45\angle 30^o \times 5$ as $45\angle(30^o \times 5)$ to give $45\angle 150^o$ when what was really intended was $(45\angle 30^o) \times 5 = 225\angle 30^o$. Also, note that the practice of highlighting previous results in a multi-line display and then "pasting" those results into a new calculation may suffer similar problems.
- *Never* re-enter a non-round computed result, but instead save that to a memory location and then recall from memory when needed for further calculations. You will find that rounding errors compound *aggressively* in complex-number arithmetic, and so the general good habit of using memory locations becomes a near-necessity with these calculations. Another important benefit to using memory locations is the avoidance of the order-of-operations problem mentioned previously: when recalling a complex-number value from memory and then placing that variable name (e.g. x) into subsequent calculations, the calculator treats the memory variable as a complete number rather than incorrectly operating on only one of its parts.

5.4 Capacitor parasitics

5.4.1 Model of a real capacitor

An ideal capacitor exhibits only capacitance, with no inductance, resistance, or other characteristics to interfere. Real capacitors exhibit all these phenomena to varying degrees, and we collectively refer to these undesirable traits as *parasitic effects*. The following diagram models some of the parasitic effects observed in real capacitors:



In addition to the capacitance the capacitor is supposed to exhibit (C_{ideal}), the capacitor also has parasitic resistance (R_{series}, also known as *Equivalent Series Resistance*, or *ESR*), parasitic inductance (L_{series}), and additional energy storage in the form of *soakage* (also known as *dielectric absorption*) whereby the dielectric substance itself absorbs and releases energy after relatively long periods of time compared to the main (ideal) capacitance.

Some of these parasitic effects – such as leakage resistance and soakage – affect the capacitor's performance in DC applications. Most of the other parasitic effects cause problems in AC and pulsed applications. For example, the effective series capacitor-inductor combination formed by C_{ideal} and L_{series} will cause *resonance* to occur at a particular AC frequency, resulting in much less reactance at that frequency than what would be predicted by the capacitive reactance formula $X_C = \frac{1}{2\pi fC}$.

Next we will explore common mechanisms for each of these effects.

5.4.2 Parasitic resistance in capacitors

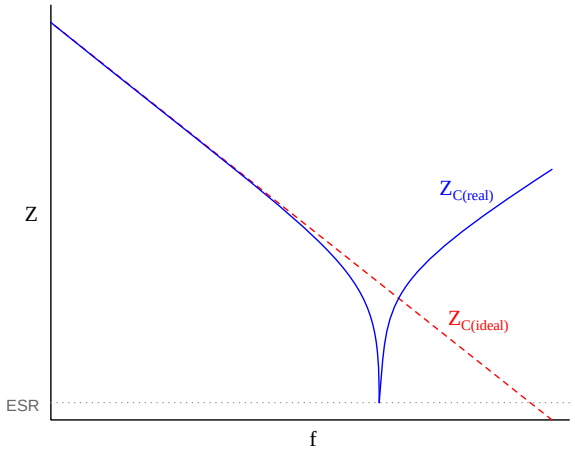
Wire resistance, of course, plays a part in this parasitic effect, but this is generally very small due to the short conductor lengths we typically see with capacitors. More significant is *dielectric losses* – energy dissipation caused by the stressing and relaxation of dipoles within the dielectric material – which act like resistance because energy ends up leaving the component (in the form of heat) and not returning to the circuit. Electrolytic capacitors have an additional source of parasitic resistance, in the form of the electrolytic gel substance used to make electrical contact from the metal-foil "plate" to the surface of the dielectric layer.

Another form of parasitic resistance within a capacitor behaves like a resistor connected in parallel with the ideal capacitance ($R_{leakage}$), resulting from the dielectric not being a perfectly insulating medium. This parasitic characteristic results in a small current passing through the capacitor even when the voltage across the capacitor is steady (i.e. $\frac{dV}{dt}$ is zero).

5.4.3 Parasitic inductance in capacitors

Any time a magnetic field forms around a current-carrying conductor, energy is stored in that magnetic field. We call this magnetic-based energy-storing capability *inductance*, and of course all capacitors must have some inductance due to the wire leads serving as connection points to the capacitor's metal plates. Much of a capacitor's parasitic inductance may be minimized by maintaining short lead lengths as it attaches to a printed-circuit board.

Parasitic inductance is a problem for capacitors in AC applications because inductive reactance (X_L) tends to cancel out capacitive reactance (X_C). If we plot the impedance of a capacitor as a function of frequency, we would expect an ideal capacitor to manifest a straight-line descent on a logarithmic plot. However, what we see is that at a certain frequency the series parasitic inductance resonates with the capacitance leaving only parasitic resistance (ESR), and then past that frequency the inductive effects overshadow the capacitance:



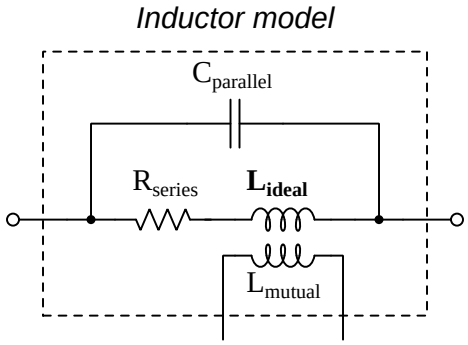
5.4.4 Other parasitic effects in capacitors

Soakage is an interesting effect resulting from dipole relaxation within the dielectric material itself, and may be modeled (as shown) by a series of resistor-capacitor subnetworks. This effect is especially prominent in aluminum electrolytic capacitors, and may be easily demonstrated by discharging a capacitor (by briefly connecting a shorting wire across the capacitor's terminals) and then monitoring the capacitor's DC voltage slowly "recover" with no connection to an external source.

5.5 Inductor parasitics

5.5.1 Model of a real inductor

An ideal inductor exhibits only inductance, with no capacitance, resistance, or other characteristics to interfere. Real inductors exhibit all these phenomena to varying degrees, and we collectively refer to these undesirable traits as *parasitic effects*. The following diagram models some of the parasitic effects observed in real inductors:



In addition to the inductance the inductor is supposed to exhibit (L_{ideal}), the inductor also has parasitic resistance (R_{series}, also known as *Equivalent Series Resistance*, or *ESR*), parasitic capacitance ($C_{parallel}$), and mutual inductance (L_{mutual}) with nearby wires and components.

Some of these parasitic effects – such as equivalent series resistance – affect the inductor's performance in DC applications. Most of the other parasitic effects cause problems in AC and pulsed applications. For example, the effective inductor-capacitor "tank circuit" formed by L_{ideal} and $C_{parallel}$ will cause *resonance* to occur at a particular AC frequency, resulting in much more reactance at that frequency than what would be predicted by the inductive reactance formula $X_L = 2\pi fL$.

Next we will explore common mechanisms for each of these effects.

5.5.2 Parasitic resistance in inductors

Wire resistance plays a dominant role in this parasitic effect due to the typically long lengths of wire necessary to wind the coil that forms most inductors. Wire resistance is not the only dissipative mechanism at work, though. Other losses include magnetic hysteresis of the iron core material as well as *eddy currents* induced in the iron core. An "eddy" current is a circulating electric current induced within the iron core of an inductor, made possible by the fact that iron is an electrically-conductive material as well as being ferromagnetic. These circulating currents do no useful work, and dissipate energy in the form of heating the iron. They may be minimized by forming the iron core from pieces of iron that are electrically insulated from one another, e.g. forming the iron core from *laminated sheets* or *powdered particles* of iron where each sheet or particle is electrically insulated from the next by a layer of non-conductive material on its outer surface.

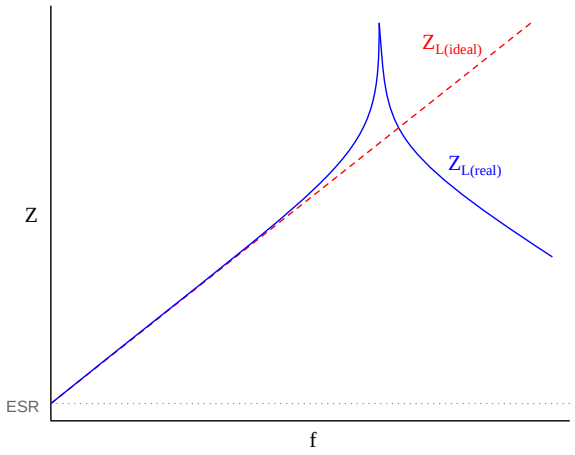
The series resistance of an inductor is always frequency-dependent. In DC conditions (i.e. frequency of zero Hertz) there will be the basic wire resistance of the coil at play. As frequency increases from zero, however, both the magnetic core losses from hysteresis and eddy currents also increase which add to the DC resistance to form a larger ESR. At extremely high frequencies the *skin effect*[6] further adds to the inductor's ESR.

[6] At high frequencies, electric current travels more toward the outer surface of a conductor rather than through the conductor's entire cross-section, effectively decreasing the conductor's cross-sectional area (gauge) as frequency rises.

5.5.3 Parasitic capacitance in inductors

Any time an electric field forms between two conductors, energy is stored in that electric field. We call this electric-based energy-storing capability *capacitance*, and of course all inductors must have some capacitance due to the insulating media between wire leads as well as between adjacent turns of wire within the coil (and between the wire turns and the iron core).

Parasitic capacitance is a problem for inductors in AC applications because capacitive reactance (X_C) tends to cancel out inductive reactance (X_L). If we plot the impedance of an inductor as a function of frequency, we would expect an ideal inductor to manifest a straight-line ascent on a logarithmic plot. However, what we see is that at a certain frequency the parallel parasitic capacitance resonates with the inductance to create a nearly-infinite impedance, and then past that frequency the capacitive effects overshadow the inductance:



Precious little may be done to eliminate parasitic capacitance within any inductor, whereas parasitic inductance is fairly easy to minimize within a capacitor. This explains why when faced with an equivalent choice between a circuit design using capacitors and a circuit design using inductors, capacitors nearly always win. Simply put, it is easier to make a nearly-ideal capacitor than it is to make a nearly-ideal inductor.

This also explains why the self-resonant frequency of most inductors is much lower than the self-resonant frequency of most capacitors: all other factors being equal, an inductor will have more parasitic capacitance in it than an equivalent capacitor will have parasitic inductance within it, making the LC product greater for the inductor than for the capacitor.

5.5.4 Other parasitic effects in inductors

Mutual inductance occurs whenever adjacent conductors' magnetic fields link with one another, which is difficult to avoid especially in physically dense circuit layouts. This parasitic effect may be minimized by proper placement of inductive components (e.g. keeping them spaced as far apart from each other as possible, orienting their axes perpendicular to each other rather than parallel) as well as by core designs with strong magnetic field containment (e.g. toroidal cores contain their magnetic fields better than rectangular cores).

Chapter 6

Programming References

A powerful tool for mathematical modeling is text-based *computer programming*. This is where you type coded commands in text form which the computer is able to interpret. Many different text-based languages exist for this purpose, but we will focus here on just two of them, *C++* and *Python*.

6.1 Programming in C++

One of the more popular text-based computer programming languages is called *C++*. This is a *compiled* language, which means you must create a plain-text file containing C++ code using a program called a *text editor*, then execute a software application called a *compiler* to translate your "source code" into instructions directly understandable to the computer. Here is an example of "source code" for a very simple C++ program intended to perform some basic arithmetic operations and print the results to the computer's console:

```
#include <iostream>
using namespace std;

int main (void)
{
  float x, y;

  x = 200;
  y = -560.5;

  cout << "This simple program performs basic arithmetic on" << endl;
  cout << "the two numbers " << x << " and " << y << " and then" << endl;
  cout << "displays the results on the computer's console." << endl;

  cout << endl;

  cout << "Sum = " << x + y << endl;
  cout << "Difference = " << x - y << endl;
  cout << "Product = " << x * y << endl;
  cout << "Quotient of " << x / y << endl;

  return 0;
}
```

Computer languages such as C++ are designed to make sense when read by human programmers. The general order of execution is left-to-right, top-to-bottom just the same as reading any text document written in English. Blank lines, indentation, and other "whitespace" is largely irrelevant in C++ code, and is included only to make the code more pleasing[1] to view.

[1] Although not included in this example, *comments* preceded by double-forward slash characters (`//`) may be added to source code as well to provide explanations of what the code is supposed to do, for the benefit of anyone reading it. The compiler application will ignore all comments.

Let's examine the C++ source code to explain what it means:

- `#include <iostream>` and `using namespace std;` are set-up instructions to the compiler giving it some context in which to interpret your code. The code specific to your task is located between the brace symbols ({ and }, often referred to as "curly-braces").

- `int main (void)` labels the "Main" function for the computer: the instructions within this function (lying between the { and } symbols) it will be commanded to execute. Every complete C++ program contains a `main` function at minimum, and often additional functions as well, but the `main` function is where execution always begins. The `int` declares this function will return an *integer* number value when complete, which helps to explain the purpose of the `return 0;` statement at the end of the `main` function: providing a numerical value of zero at the program's completion as promised by `int`. This returned value is rather incidental to our purpose here, but it is fairly standard practice in C++ programming.

- Grouping symbols such as (parentheses) and {braces} abound in C, C++, and other languages (e.g. Java). Parentheses typically group data to be processed by a function, called *arguments* to that function. Braces surround lines of executable code belonging to a particular function.

- The `float` declaration reserves places in the computer's memory for two *floating-point* variables, in this case the variables' names being `x` and `y`. In most text-based programming languages, variables may be named by single letters or by combinations of letters (e.g. `xyz` would be a single variable).

- The next two lines assign numerical values to the two variables. Note how each line terminates with a semicolon character (`;`) and how this pattern holds true for most of the lines in this program. In C++ semicolons are analogous to periods at the ends of English sentences. This demarcation of each line's end is necessary because C++ ignores whitespace on the page and doesn't "know" otherwise where one line ends and another begins.

- All the other instructions take the form of a `cout` command which prints characters to the "standard output" stream of the computer, which in this case will be text displayed on the console. The double-less-than symbols (`<<`) show data being sent *toward* the `cout` command. Note how verbatim text is enclosed in quotation marks, while variables such as `x` or mathematical expressions such as `x - y` are not enclosed in quotations because we want the computer to display the numerical values represented, not the literal text.

- Standard arithmetic operations (add, subtract, multiply, divide) are represented as `+`, `-`, `*`, and `/`, respectively.

- The `endl` found at the end of every `cout` statement marks the end of a line of text printed to the computer's console display. If not for these `endl` inclusions, the displayed text would resemble a run-on sentence rather than a paragraph. Note the `cout << endl;` line, which does nothing but create a blank line on the screen, for no reason other than esthetics.

After saving this *source code* text to a file with its own name (e.g. `myprogram.cpp`), you would then *compile* the source code into an *executable* file which the computer may then run. If you are using a console-based compiler such as *GCC* (very popular within variants of the Unix operating system[2], such as Linux and Apple's OS X), you would type the following command and press the Enter key:

```
g++ -o myprogram.exe myprogram.cpp
```

This command instructs the `GCC` compiler to take your source code (`myprogram.cpp`) and create with it an executable file named `myprogram.exe`. Simply typing `./myprogram.exe` at the command-line will then execute your program:

```
./myprogram.exe
```

If you are using a graphic-based C++ development system such as Microsoft Visual Studio[3], you may simply create a new console application "project" using this software, then paste or type your code into the example template appearing in the editor window, and finally run your application to test its output.

As this program runs, it displays the following text to the console:

```
This simple program performs basic arithmetic on
the two numbers 200 and -560.5 and then
displays the results on the computer's console.

Sum = -360.5
Difference = 760.5
Product = -112100
Quotient of -0.356824
```

As crude as this example program is, it serves the purpose of showing how easy it is to write and execute simple programs in a computer using the C++ language. As you encounter C++ example programs (shown as source code) in any of these modules, feel free to directly copy-and-paste the source code text into a text editor's screen, then follow the rest of the instructions given here (i.e. save to a file, compile, and finally run your program). You will find that it is generally easier to

[2] A very functional option for users of Microsoft Windows is called *Cygwin*, which provides a Unix-like console environment complete with all the customary utility applications such as GCC!

[3] Using Microsoft Visual Studio community version 2017 at the time of this writing to test this example, here are the steps I needed to follow in order to successfully compile and run a simple program such as this: (1) Start up Visual Studio and select the option to create a New Project; (2) Select the Windows Console Application template, as this will perform necessary set-up steps to generate a console-based program which will save you time and effort as well as avoid simple errors of omission; (3) When the editing screen appears, type or paste the C++ code within the `main()` function provided in the template, deleting the "Hello World" `cout` line that came with the template; (4) Type or paste any preprocessor directives (e.g. `#include` statements, `namespace` statements) necessary for your code that did not come with the template; (5) Lastly, under the Debug drop-down menu choose either Start Debugging (F5 hot-key) or Start Without Debugging (Ctrl-F5 hotkeys) to compile ("Build") and run your new program. Upon execution a console window will appear showing the output of your program.

learn computer programming by closely examining others' example programs and modifying them than it is to write your own programs starting from a blank screen.

6.2 Programming in Python

Another text-based computer programming language called *Python* allows you to type instructions at a terminal prompt and receive immediate results without having to compile that code. This is because Python is an *interpreted* language: a software application called an *interpreter* reads your source code, translates it into computer-understandable instructions, and then executes those instructions in one step.

The following shows what happens on my personal computer when I start up the Python interpreter on my personal computer, by typing `python3`[4] and pressing the Enter key:

```
Python 3.7.2 (default, Feb 19 2019, 18:15:18)
[GCC 4.1.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The `>>>` symbols represent the prompt within the Python interpreter "shell", signifying readiness to accept Python commands entered by the user.

Shown here is an example of the same arithmetic operations performed on the same quantities, using a Python interpreter. All lines shown preceded by the `>>>` prompt are entries typed by the human programmer, and all lines shown without the `>>>` prompt are responses from the Python interpreter software:

```
>>> x = 200
>>> y = -560.5
>>> x + y
-360.5
>>> x - y
760.5
>>> x * y
-112100.0
>>> x / y
-0.35682426404995538
>>> quit()
```

[4] Using version 3 of Python, which is the latest at the time of this writing.

More advanced mathematical functions are accessible in Python by first entering the line `from math import *` which "imports" these functions from Python's math *library* (with functions identical to those available for the C programming language, and included on any computer with Python installed). Some examples show some of these functions in use, demonstrating how the Python interpreter may be used as a scientific calculator:

```
>>> from math import *
>>> sin(30.0)
-0.98803162409286183
>>> sin(radians(30.0))
0.49999999999999994
>>> pow(2.0, 5.0)
32.0
>>> log10(10000.0)
4.0
>>> e
2.7182818284590451
>>> pi
3.1415926535897931
>>> log(pow(e,6.0))
6.0
>>> asin(0.7071068)
0.78539819000368838
>>> degrees(asin(0.7071068))
45.000001524425265
>>> quit()
```

Note how trigonometric functions assume angles expressed in *radians* rather than *degrees*, and how Python provides convenient functions for translating between the two. Logarithms assume a base of e unless otherwise stated (e.g. the `log10` function for common logarithms).

The interpreted (versus compiled) nature of Python, as well as its relatively simple syntax, makes it a good choice as a person's first programming language. For complex applications, interpreted languages such as Python execute slower than compiled languages such as C++, but for the very simple examples used in these learning modules speed is not a concern.

Another Python math library is `cmath`, giving Python the ability to perform arithmetic on complex numbers. This is very useful for AC circuit analysis using *phasors*[5] as shown in the following example. Here we see Python's interpreter used as a scientific calculator to show series and parallel impedances of a resistor, capacitor, and inductor in a 60 Hz AC circuit:

```
>>> from math import *
>>> from cmath import *
>>> r = complex(400,0)
>>> f = 60.0
>>> xc = 1/(2 * pi * f * 4.7e-6)
>>> zc = complex(0,-xc)
>>> xl = 2 * pi * f * 1.0
>>> zl = complex(0,xl)
>>> r + zc + zl
(400-187.38811239154882j)
>>> 1/(1/r + 1/zc + 1/zl)
(355.837695813625+125.35793777619385j)
>>> polar(r + zc + zl)
(441.717448903332, -0.4381072059213295)
>>> abs(r + zc + zl)
441.717448903332
>>> phase(r + zc + zl)
-0.4381072059213295
>>> degrees(phase(r + zc + zl))
-25.10169387356105
```

When entering a value in rectangular form, we use the `complex()` function where the arguments are the real and imaginary quantities, respectively. If we had opted to enter the impedance values in polar form, we would have used the `rect()` function where the first argument is the magnitude and the second argument is the angle in radians. For example, we could have set the capacitor's impedance (`zc`) as $X_C \angle -90^o$ with the command `zc = rect(xc,radians(-90))` rather than with the command `zc = complex(0,-xc)` and it would have worked the same.

Note how Python defaults to rectangular form for complex quantities. Here we defined a 400 Ohm resistance as a complex value in rectangular form (400 +j0 Ω), then computed capacitive and inductive reactances at 60 Hz and defined each of those as complex (phasor) values ($0 - jX_c$ Ω and $0 + jX_l$ Ω, respectively). After that we computed total impedance in series, then total impedance in parallel. Polar-form representation was then shown for the series impedance (441.717 Ω $\angle -25.102^o$). Note the use of different functions to show the polar-form series impedance value: `polar()` takes the complex quantity and returns its polar magnitude and phase angle in *radians*; `abs()` returns just the polar magnitude; `phase()` returns just the polar angle, once again in radians. To find the polar phase angle in degrees, we nest the `degrees()` and `phase()` functions together.

The utility of Python's interpreter environment as a scientific calculator should be clear from these examples. Not only does it offer a powerful array of mathematical functions, but also unlimited

[5] A "phasor" is a voltage, current, or impedance represented as a complex number, either in rectangular or polar form.

assignment of variables as well as a convenient text record[6] of all calculations performed which may be easily copied and pasted into a text document for archival.

It is also possible to save a set of Python commands to a text file using a text editor application, and then instruct the Python interpreter to execute it at once rather than having to type it line-by-line in the interpreter's shell. For example, consider the following Python program, saved under the filename `myprogram.py`:

```
x = 200
y = -560.5

print("Sum")
print(x + y)

print("Difference")
print(x - y)

print("Product")
print(x * y)

print("Quotient")
print(x / y)
```

As with C++, the interpreter will read this source code from left-to-right, top-to-bottom, just the same as you or I would read a document written in English. Interestingly, whitespace *is* significant in the Python language (unlike C++), but this simple example program makes no use of that.

To execute this Python program, I would need to type `python myprogram.py` and then press the Enter key at my computer console's prompt, at which point it would display the following result:

```
Sum
-360.5
Difference
760.5
Product
-112100.0
Quotient
-0.35682426405
```

As you can see, syntax within the Python programming language is simpler than C++, which is one reason why it is often a preferred language for beginning programmers.

[6] Like many command-line computing environments, Python's interpreter supports "up-arrow" recall of previous entries. This allows quick recall of previously typed commands for editing and re-evaluation.

If you are interested in learning more about computer programming in *any* language, you will find a wide variety of books and free tutorials available on those subjects. Otherwise, feel free to learn by the examples presented in these modules.

6.3 Modeling low-pass filters using C++

Modeling an RC low-pass filter circuit at a single frequency is very simple in the C++ language, as the following source code reveals:

```
#include <iostream>
#include <cmath>
using namespace std;

int main (void)
{
  float f, Vsrc, R1, C1, XC, Ztotal, I;

  Vsrc = 10.0;
  R1 = 1.5e3;
  C1 = 0.33e-6;
  f = 400.0;

  cout << "+---Vsrc----R1--+---C1---+ " << endl;
  cout << "|               |        | " << endl;
  cout << "|               * Out *--+ " << endl;
  cout << "|                        | " << endl;
  cout << "+------------------------+ " << endl;

  XC = 1 / (2 * M_PI * f * C1);
  Ztotal = sqrt(pow(R1,2) + pow(XC,2));
  I = Vsrc/Ztotal;

  cout << "V_out = " << I * XC << " Volts" << endl;

  return 0;
}
```

This particular program simplifies the mathematics as much as possible, using only real numbers (rather than complex numbers) for all its calculations. Capacitive reactance is computed using the formula $X_C = \frac{1}{2\pi f C_1}$, and then total series impedance is computed using the Pythagorean theorem $Z_{total} = \sqrt{R_1^2 + X_C^2}$. After those calculations, the program computes circuit current using Ohm's Law ($I = \frac{V_{src}}{Z_{total}}$) and displays output voltage as the product of current and capacitive reactance again using Ohm's Law ($V_{out} = IX_C$).

When compiled and run, the program produces the following output:

```
+---Vsrc----R1--+---C1---+
|               |        |
|               * Out *--+
|                        |
+------------------------+
V_out = 6.26505 Volts
```

Analyzing the source code, we will explore how the program accomplishes the task of modeling a simple RC low-pass filter. Along the way we will discuss the following programming principles:

- Order of execution
- Preprocessor directives, namespaces
- The `main` function: return values, arguments
- Delimiter characters (e.g. { } ;)
- Variable types (`float`), names, and declarations
- Variable assignment/initialization (`=`)
- Printing text output (`cout`, `<<`, `endl`)
- Basic arithmetic (`+`, `-`, `*`, `/`)
- Arithmetic functions (`pow`, `sqrt`) and arguments
- Accepting user input (`cin`, `>>`)
- Loops (`while`, `for`)
- Comparison (`<=`)
- Redirection of console output to a file

As with most text-based computer programming languages, C and C++ alike are evaluated from left to right, top to bottom, much the same as reading an English-language document.

The first three lines of code instruct the C++ compiler software how to interpret the rest of the program. The two `#include` lines tell it to read the `iostream` and `cmath` header files (installed on the computer as part of the compiler software), inside of which reside all the detailed definitions of C++ instructions such as `cout` and `sqrt`. The `namespace` line instructs the compiler to use the standard (`std`) definitions of instruction names. None of these lines of code are particularly important to our understanding of the filter circuit simulation proper, but they address low-level details necessary for our program to compile and function.

All of our filter simulation code is indented (for esthetics only), appearing after the opening brace symbol ({) and before the closing brace symbol (}). These "curly-brace" symbols define the

boundaries of the `main` function which marks the starting point of execution for any C or C++ program. The `int main (void)` line labels the function as being `main`, as accepting no input variables (`void`), and declares it will return an integer number (`int`) upon completion. Again, these details are not particularly important to our immediate task of modeling a filter circuit, but they are necessary for any properly-formed C or C++ program.

Arriving at our filter-specific code, we first encounter the `float` line which declares several *floating-point* variables. These are places in the computer's memory which will hold decimal-point numerical values (i.e. non-integers). As you can see, valid variable names in C++ include single letters (e.g. `f` and `I`) as well as letter-number combinations (e.g. `R1` and `C1`), and "words" (e.g. `Vsrc` and `Ztotal`). No spaces are allowed in variable names, as C and C++ interpret space characters as delimiters separating different entities.

Independent lines of executable code must be terminated by a semicolon character in C and C++ alike. This is analogous to a period at the end of an English sentence, signifying the end of a sentence. Languages such as C and C++ ignore "whitespace" in the source code file, and so must be explicitly told when one line of code ends and another line begins[7].

The next four lines of code *assign* numerical values to some of the variables. The "equals" symbol (`=`) has a slightly different meaning in C and C++ than in mathematics. In general math notation, this symbol expresses equality between the entities on either side of it. In C/C++, a single equals symbol denotes an *action* for the computer to take, meaning the variable on the left will be *set equal to* the value on the right. In general mathematics, $f = 400.0$ means the same thing as $400.0 = f$, but in C/C++ the latter statement would be nonsense, for the computer cannot *assign* a value to the constant 400.

Note how C++ accepts power-of-ten notation (e.g. `0.33e-6` means 0.33×10^{-6}).

Next, we see several `cout` lines, the purpose of which is to print text to the computer's display console. Anything enclosed within quotation marks will be printed verbatim by `cout`. At the end of each `cout` line we see another character included in the print action, `endl`: this forces `cout` to print an *end-of-line* character so that our displayed text does not appear as one run-on sentence.

It should be noted that these `cout` lines are not strictly necessary to simulate a filter circuit. All they do for the user is print a crude rendering of the filter circuit to give some context to what the rest of the program is doing.

After this we see three lines of code where the mathematical computations take place. Basic arithmetic operations are simple enough to represent in C/C++: addition (`+`), subtraction (`-`), multiplication (`*`), and division (`/`), and parentheses work as grouping symbols just as they do in conventional math notation. What might appear strange here to anyone familiar with mathematical notation is the use of two types of C++ math *functions*: *square root* (`sqrt`) and *powers* (`pow`). In a mathematical formula we would see these two operations denoted using symbology such as $\sqrt{\ }$ and x^2, but these symbols do not exist in plain-text typesetting and so C++ requires a different

[7] It is valid in C and C++ to "stack" separate lines of code in one actual line within the text file, so long as the semicolons remain intact. For example, the four lines initializing `Vsrc`, `R1`, `C1`, and `f` with values could be written in the text editor as a single line: `Vsrc = 10.0; R1 = 1.5e3; C1 = 0.33e-6; f = 400.0;`. It is typical to find C/C++ source code written not just for function, but for *readability* so that human beings analyzing the code will have an easier time understanding what it is supposed to do.

method of description. The solution is to use *functions* as we typically see in higher mathematics (e.g. $f(x)$).

For example, to raise `R1` and `XC` to the second power (i.e. *square*), we use the `pow` function that is part of the basic C/C++ math library. So, R_1^2 is represented as `pow(R1,2)` and X_C^2 is represented as `pow(XC,2)`. The terms contained within each functions' parentheses are called *arguments* to the function. In the case of the `pow` function, the first argument is the variable to be raised to a power and the second argument is the power we are raising that variable to.

For the square root operation, we use the `sqrt` function which expects a single argument. For example, if we wished to take the square root of 9 we would write it as $\sqrt{9}$ using standard mathematical notation but we would code it as `sqrt(9)` in C or C++.

The last `cout` line in this program prints some explanatory text to the console, sandwiching a numerical display of output voltage. Note how the text to be printed verbatim is enclosed in quotation marks, but the mathematical expression `I * XC` is not because we want that expression's *value* to be printed and not "`I * XC`" literally. As usual, an `endl` is printed at the end to terminate the line on the console display.

Finally, we have a `return` statement providing a numerical value (in this case, zero) as promised at the conclusion of the *main* function. Again, this is not strictly necessary for our filter simulation to function, but it is a good programming practice. When we begin writing our own functions in C++ this concept will become very important!

This filter-modeling program is very simple, and could be made much more useful than it presently is. One practical change would be to make it so the user can enter their own filter circuit parameters at run-time, instead of having those parameters "hard-coded" into the source code where only the programmer has access to alter them.

C++ provides an instruction called `cin` for this purpose, and it works in similar fashion to `cout`. Note how these are applied in the following edit to the program:

```
#include <iostream>
#include <math.h>
using namespace std;

int main (void)
{
  float f, Vsrc, R1, C1, XC, Ztotal, I;

  cout << "Source voltage (Volts) = ";
  cin >> Vsrc;
  cout << "Resistor value (Ohms) = ";
  cin >> R1;
  cout << "Capacitor value (Farads) = ";
  cin >> C1;
  cout << "Frequency (Hz) = ";
  cin >> f;

  cout << "+---Vsrc----R1--+---C1---+ " << endl;
  cout << "|               |        | " << endl;
  cout << "|               * Out *--+ " << endl;
  cout << "|                        | " << endl;
  cout << "+------------------------+ " << endl;

  XC = 1 / (2 * M_PI * f * C1);
  Ztotal = sqrt(pow(R1,2) + pow(XC,2));
  I = Vsrc/Ztotal;

  cout << "V_out = " << I * XC << " Volts" << endl;

  return 0;
}
```

When we run this program, it prompts us for source voltage value, resistor value, capacitor value, and frequency before displaying the diagram and calculating output voltage. In this case, I entered 45.2 Volts, 12 kΩ (entered as `12000`), 0.01 μF (entered as `0.01e-6`), and 2.5 kHz (entered as `2.5e3`):

```
Source voltage (Volts) = 45.2
Resistor value (Ohms) = 12000
Capacitor value (Farads) = 0.01e-6
Frequency (Hz) = 2.5e3
+---Vsrc----R1--+---C1---+
|               |        |
|               * Out *--+
|                        |
+------------------------+
V_out = 21.183 Volts
```

Re-running the program gives us fresh opportunity to enter parameters. One practical use of this feature is to test the same filter circuit at different frequency values. Here I have re-run the program, changing only the frequency from 2.5 kHz to 3 kHz:

```
Source voltage (Volts) = 45.2
Resistor value (Ohms) = 12000
Capacitor value (Farads) = 0.01e-6
Frequency (Hz) = 3e3
+---Vsrc----R1--+---C1---+
|               |        |
|               * Out *--+
|                        |
+------------------------+
V_out = 18.2764 Volts
```

As expected for a low-pass filter, the output voltage decreases as frequency increases. This prompts another idea for a useful feature. Why not re-code the program to prompt just *once* for the circuit component values (source voltage, resistance, capacitance) and then have it repeatedly prompt us for different frequency values? That way we could analyze how any one filter would respond to a multitude of frequency values without having to re-enter *everything* each time.

In C and C++, the *while* instruction causes portions of a program to "loop" (i.e. repeatedly execute). The basic structure of a `while` loop looks like this:

```
while (1)
{
  // Code to be executed
  // repeatedly . . .
}
```

The `while` loop's argument is evaluated as either "true" (1) or "false" (0) – if true, the code enclosed within the `while` loop's brace symbols repeatedly executes; if not the computer skips past the `while` loop entirely and executes whatever code comes *after* it.

Examine the following code to see how the `while` instruction provides this "looping" feature:

```
#include <iostream>
#include <math.h>
using namespace std;

int main (void)
{
  float f, Vsrc, R1, C1, XC, Ztotal, I;

  cout << "Source voltage (Volts) = ";
  cin >> Vsrc;
  cout << "Resistor value (Ohms) = ";
  cin >> R1;
  cout << "Capacitor value (Farads) = ";
  cin >> C1;

  cout << "+---Vsrc----R1--+---C1---+ " << endl;
  cout << "|               |        | " << endl;
  cout << "|               * Out *--+ " << endl;
  cout << "|                        | " << endl;
  cout << "+------------------------+ " << endl;

  while(1)
  {
    cout << "Frequency (Hz) = ";
    cin >> f;

    XC = 1 / (2 * M_PI * f * C1);
    Ztotal = sqrt(pow(R1,2) + pow(XC,2));
    I = Vsrc/Ztotal;

    cout << "V_out = " << I * XC << " Volts" << endl;
  }

  return 0;
}
```

Note how the `cout` and `cin` lines related to the user prompts for frequency have been moved *after* (i.e. below) the `cout` lines printing the circuit diagram, and how the last lines of code (all

except the `return` statement) have been placed inside a `while` loop. Those "looped" code lines are all indented purely for ease of reading, as C++ does not care about whitespace.

The following screen-capture shows an example of this program simulating the same filter circuit as last time, but for six different frequency values from 2.5 kHz to 5 kHz:

```
Source voltage (Volts) = 45.2
Resistor value (Ohms) = 12000
Capacitor value (Farads) = 0.01e-6
+---Vsrc----R1--+---C1---+
|               |        |
|               * Out *--+
|                        |
+------------------------+
Frequency (Hz) = 2.5e3
V_out = 21.183 Volts
Frequency (Hz) = 3e3
V_out = 18.2764 Volts
Frequency (Hz) = 3.5e3
V_out = 16.0167 Volts
Frequency (Hz) = 4e3
V_out = 14.2255 Volts
Frequency (Hz) = 4.5e3
V_out = 12.7784 Volts
Frequency (Hz) = 5e3
V_out = 11.5889 Volts
```

Since this `while` loop's condition is always true, the program must be externally interrupted. When running programs in console mode, the conventional way to do this is to simultaneously press the "Control" and "C" buttons on the keyboard. The <Ctrl-C> sequence issues an *interrupt signal* to the program in order to halt execution.

Yet another improvement we could make to this program is to have it calculate a sequence of test frequencies given a user-entered *range*. For example, instead of requiring the user to enter frequency value after frequency value, we could simply prompt the user for a "starting" frequency, an "ending" frequency, and a frequency interval value, and have the computer do the rest.

A different type of programming *loop* we could use for this purpose is the `for` loop, as an alternative to the `while` loop. The basic form of this instruction is shown here:

```
for (start ; during ; change)
{
  // Code to be executed
  // repeatedly . . .
}
```

Examine the following code to see how this `for` loop functions:

```
#include <iostream>
#include <math.h>
using namespace std;

int main (void)
{
  float f, f_start, f_end, f_step, Vsrc, R1, C1, XC, Ztotal, I;

  cout << "Source voltage (Volts) = ";
  cin >> Vsrc;
  cout << "Resistor value (Ohms) = ";
  cin >> R1;
  cout << "Capacitor value (Farads) = ";
  cin >> C1;

  cout << "+---Vsrc----R1--+---C1---+ " << endl;
  cout << "|               |        | " << endl;
  cout << "|               * Out *--+ " << endl;
  cout << "|                        | " << endl;
  cout << "+------------------------+ " << endl;

  cout << "Starting frequency (Hz) = ";
  cin >> f_start;
  cout << "Ending frequency (Hz) = ";
  cin >> f_end;
  cout << "Frequency step interval (Hz) = ";
  cin >> f_step;

  cout << "Frequency , V_out" << endl;

  for (f = f_start ; f <= f_end ; f = f + f_step)
  {
    XC = 1 / (2 * M_PI * f * C1);
    Ztotal = sqrt(pow(R1,2) + pow(XC,2));
    I = Vsrc/Ztotal;

    cout << f << " , " << I * XC << endl;
  }

  return 0;
}
```

Running this program with the same component values and a user-entered frequency range of 1 kHz to 5 kHz in 500 Hz steps:

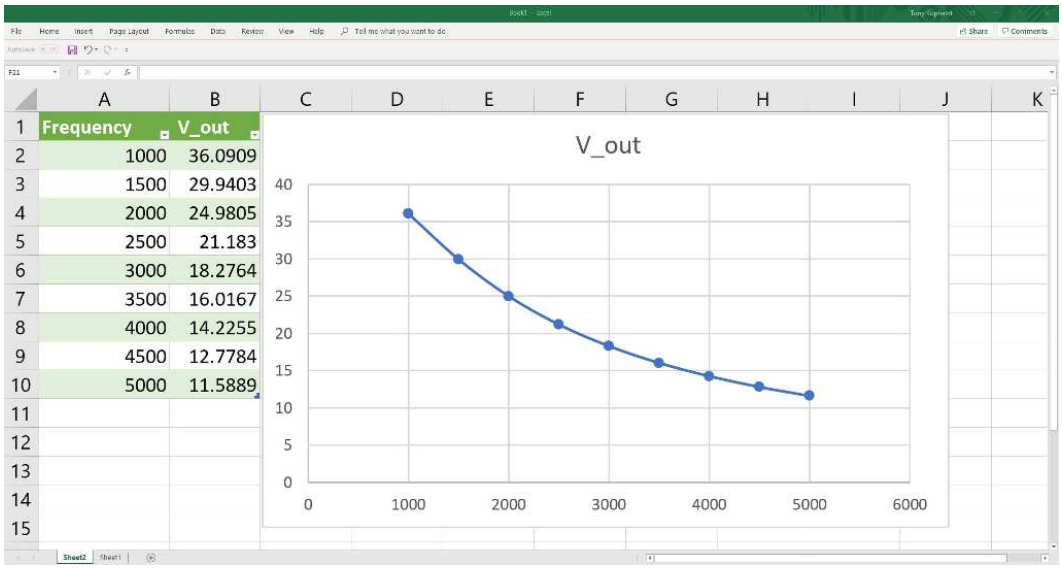
```
Source voltage (Volts) = 45.2
Resistor value (Ohms) = 12000
Capacitor value (Farads) = 0.01e-6
+---Vsrc----R1--+---C1---+
|               |        |
|               * Out *--+
|                        |
+------------------------+
Starting frequency (Hz) = 1000
Ending frequency (Hz) = 5000
Frequency step interval (Hz) = 500
Frequency , V_out
1000 , 36.0909
1500 , 29.9403
2000 , 24.9805
2500 , 21.183
3000 , 18.2764
3500 , 16.0167
4000 , 14.2255
4500 , 12.7784
5000 , 11.5889
```

Examining this program's source code, we see three new floating-point variables added to the declaration near the top of the code listing: `f_start, f_end, and f_step`. Recall that spaces are not allowed in C/C++ variable names, and in order to make these variable names more readable to human beings we use underscore characters to separate what would ordinarily be single-letter variable names and subscripts (e.g. f_{end} is written as `f_end` in C++).

The `for` statement deserves some explanation as well. Within its parentheses are three arguments, telling `for` what the value of frequency (`f`) should start out at, what condition of `f` must be met for the loop to repeat, and how much `f` should increment with every iteration of the loop. In our case, `f` needs to begin at the user-entered value of `f_start`; the loop should repeat so long as `f` is less than or equal to the user-entered value of `f_end`; and with each pass through the loop we need to increment `f` by the user-entered value of `f_step`.

With the addition of the `for` loop we have also modified the formatting of the last `cout` instruction. Now, that final `cout` instruction prints the frequency and voltage values separated by a comma. A newly-added `cout` instruction just prior to the `for` loop prints a text "header" with `Frequency` and `Vout` also separated by a comma. These changes serve a new purpose, that being to make the program's printed output compatible with the standard comma-separated-value format (`.csv`) of modern spreadsheet software applications. After running this program, the user will be able to copy-and-paste these columns of numbers into a spreadsheet for easy graphing.

The following screenshot shows Microsoft Excel graphing this same filter data output[8] by our C++ program:

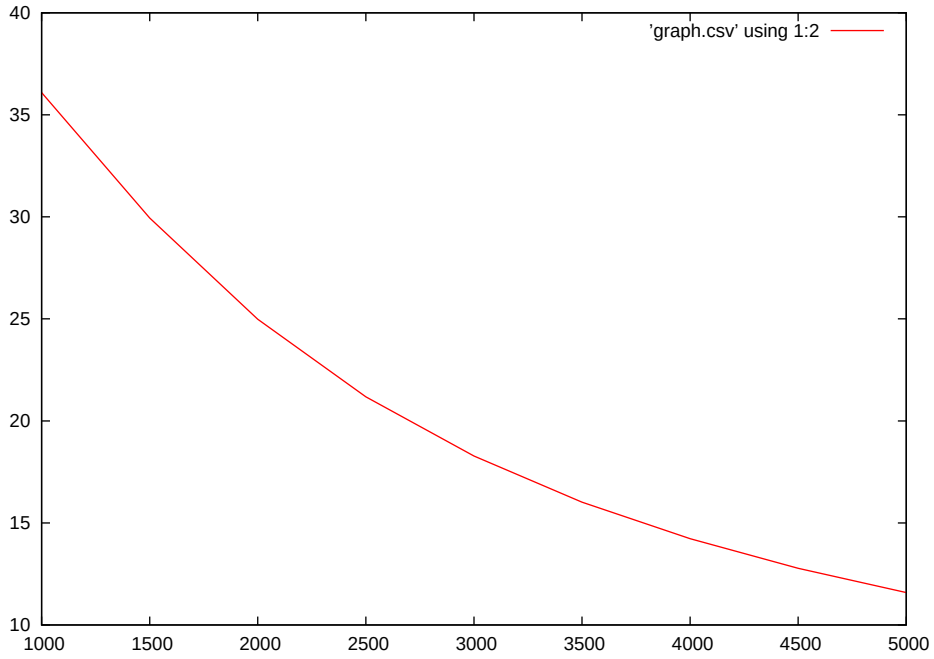


[8] A relatively easy way to do this is to run the C++ program from a console, using the *redirection* symbol (`>`). For example, if we saved our source code file under the name `filter.cpp` and then entered `g++ -o filter.exe filter.cpp` at the command-line interface to compile it, the resulting executable file would be named `filter.exe`. If we simply type `./filter.exe` and press Enter, the program will run as usual. If, however we type `./filter.exe > graph.csv` and press Enter, the program will run "silently" with all of its printed text output redirected into a file named `graph.csv` instead of to the console for us to see. Alternatively, we can use the *pipe* (`|`) and `tee` commands to send the program's text output *both* to the console (for viewing) *and* to a file of our own choosing for spreadsheet import: `./filter.exe | tee graph.csv`. Then, starting Microsoft Excel, we can tell it to read `graph.csv` as a comma-separated-value file. Once read by Excel, those two columns of numbers may be selected (using your mouse) and linked to a "scatter plot" style of chart for graphical presentation.

Spreadsheets aren't the only software applications capable of plotting numerical data from a text file. Taking the same comma-delimited text from the output of our C++ program and feeding that data in to an open-source software application called `gnuplot` using the following commands[9]:

`gnuplot` instruction code:

```
set datafile separator ","
set style line 1 lw 2 lc rgb "red"
plot 'graph.csv' using 1:2 with lines ls 1
```



[9]These commands may be entered interactively at the `gnuplot` prompt or saved to a text file (e.g. `format.txt`) and invoked at the operating system command line (e.g. `gnuplot -p format.txt`).

6.4 Discrete Fourier Transform algorithm in C++

The following page of C++ code is the `main()` function for a Discrete Fourier Transform algorithm. As written, this C++ program simulates a square wave and computes the DC average value as well as the first nine harmonics of this wave, although the `f(x)` function code could be re-written to generate any test waveform desired.

A DFT algorithm requires no calculus, only simple trigonometric functions (sine and cosine) and basic arithmetic (multiplication and addition, squares and square roots). The basic idea of it is simple enough: multiply the instantaneous values of the test waveform by the corresponding values of a sinusoid at some harmonic of the test frequency, and sum all of those values over one period of the test waveform. If the sum adds up to zero (or nearly) zero, then that harmonic does not exist in the test waveform. The magnitude of this sum indicates how strong the harmonic is in the test waveform.

Even the mathematical foundation of the DFT is simple, and requires no calculus. It is based on trigonometric identities, specifically those involving the product (multiplication) of sine and/or cosine terms. When two sinusoids of differing frequency are multiplied together, the result is two completely different sinusoids: one having a frequency equal to the sum of the two original frequencies, and the other having a frequency equal to the difference of the two original frequencies. The basic trigonometric identity is shown here:

$$\cos x \cos y = \frac{\cos(x - y) + \cos(x + y)}{2}$$

Next, is the version of this using ω_x and ω_y to represent the two waves' frequencies:

$$\cos(\omega_x t) \cos(\omega_y t) = \frac{\cos(\omega_x t - \omega_y t) + \cos(\omega_x t + \omega_y t)}{2}$$

If the sinusoids being multiplied happen to have the same frequency and be in-phase with each other, the result is a second harmonic and a DC (constant) value (i.e. one sinusoid having a frequency of 2ω and the other having a frequency of zero). So, in order to test a waveform for the presence of a particular harmonic, we multiply it by that other harmonic and see if the resulting product contains DC. How do we test a wave for DC? We sum up all its instantaneous values and see if the result is anything other than zero!

Any practical DFT needs to be just a bit more sophisticated, though, because we must account for phase. We obtain a DC-containing product only if the frequencies *and* phases match. If we happen to multiply a wave by another that's exactly 90^o out of phase, we don't get any DC. To account for phase shift, then, what we do is compute two products and two sums: one based on a sine wave and the other based on a cosine wave (i.e. 90^o apart from each other, so at least one of these two sums will show a match) and then tally their respective sums by the Pythagorean theorem: $\sqrt{x^2 + y^2}$. The rationale for using sine and cosine waves is the same as representing an AC phasor quantity in rectangular form: the sum based on cosines represents the real component of the phasor while the sum based on sines represents the imaginary component. $\sqrt{x^2 + y^2}$ simply computes the polar-form magnitude of these sinusoids' sums.

```
#include <iostream>
#include <math.h>
using namespace std;

float f(int x);

int main(void)
{
  int sample, harmonic;
  float sinsum, cossum, polarsum[10];

  for (harmonic = 1; harmonic < 10; ++harmonic)
    {
      sinsum = 0;
      cossum = 0;

      for (sample = 0; sample < 128; ++sample)
        {
          sinsum = sinsum + (f(sample) * (sin(sample*harmonic*2*M_PI/128)));
          cossum = cossum + (f(sample) * (cos(sample*harmonic*2*M_PI/128)));
        }

      polarsum[harmonic] = sqrt(pow(cossum, 2) + pow(sinsum, 2));

      cout << "Harmonic = " << harmonic << " -- Normalized weight = "
      << fixed << polarsum[harmonic] / polarsum[1] << endl;
    }

  return 0;
}

float f(int x)
{
  if (x < 64)
    return 1.0;

  else
    return -1.0;
}
```

What follows is an explanation of how this DFT algorithm's code works.

- The `include` and `namespace` directives instruct the compiler to be prepared for functions of text printing (`iostream`) and for mathematics (`math.h`).
- The next line (`float f(int x);`) is a *function prototype* for a C++ function named `f`. This function purposely resembles the standard mathematical function form $f(x)$ because it is where the code will reside for the waveform to be analyzed. The input to this function will be an integer number, and the output will be a floating-point number (i.e. capable of fractional values, unlike an integer). The domain of our function happens to be 0 to 127, in whole-numbered steps. The range of our function can be anything representable by a floating-point number. The actual code for this function may appear later in the file (as is the case in this example), or it may even reside in its own source file to be linked to the main program at compilation time.
- Inside the `main()` function we first declare several variables, both integer and floating-point. All mathematical functions are computed over 128 samples, numbered 0 through 127. During each of these samples, we compute the value of our test waveform (`f(x)`) and multiply it by the corresponding value of a sine wave and of a cosine wave, each at some harmonic frequency of the test waveform. The inner `for()` loop computes these products, and also a running total of each (`sinsum` and `cossum`). After each completion of the inner `for()` loop, we use the Pythagorean Theorem to combine the sine- and cosine-sums so that we get a complete summation (`polarsum`) for that harmonic, saving each one in an array `polarsum[]`, with `polarsum[1]` being the basis for normalizing the values of all others. We then print that summed value. The outer `for()` loop repeats this process for harmonics 1 through 9.
- Our test waveform is generated within its own subroutine, called a *function* in C and C++ alike. Here is where we insert code to generate whatever waveform we wish to analyze. In this particular example, it is a square wave with a peak value of 1. The algorithm for creating this square wave is extremely simple: for x values from 0 to 63 the wave is at $+1$, and for x values from 64 to 127 the wave is at -1. Since the domain of x happens to be 0 to 127 (as called by the `main()` program) this produces one symmetrical cycle of a square wave.

Locating the `f(x)` function within its own section of C++ code allows for easy modification of that function in the future, without modifying the `main()` program. This is generally a good programming practice: to make your code modular so that individual sections of it may be separately edited (and even reside in separate source files!). Doing this makes it easier for teams of programmers to develop projects together, and also makes it easier for code to be re-used in other projects.

6.4.1 DFT of a square wave

When the example code previously shown is compiled and run, the result is the following text output:

```
Harmonic = 1 -- Normalized weight = 1.000000
Harmonic = 2 -- Normalized weight = 0.000000
Harmonic = 3 -- Normalized weight = 0.333601
Harmonic = 4 -- Normalized weight = 0.000000
Harmonic = 5 -- Normalized weight = 0.200483
Harmonic = 6 -- Normalized weight = 0.000000
Harmonic = 7 -- Normalized weight = 0.143548
Harmonic = 8 -- Normalized weight = 0.000000
Harmonic = 9 -- Normalized weight = 0.112009
```

This program assumes the first harmonic's amplitude is the "norm" by which all other harmonics are scaled. Therefore, the first harmonic always shows up as having a normalized weight of 1, with all other harmonic values shown proportionate to that norm.

Fourier theory predicts that a square wave with a 50% duty cycle will only contain odd harmonics (in agreement with our **symmetry rule**), the relative amplitudes of those harmonics diminishing by a factor of $\frac{1}{n}$ where n is the harmonic number. Therefore, if the first harmonic is normalized to an amplitude of 1, then the third harmonic will have an amplitude of $\frac{1}{3}$, the fifth harmonic an amplitude of $\frac{1}{5}$, etc.:

$$v_{square} = \frac{4}{\pi} V_m \left(\sin \omega t + \frac{1}{3} \sin 3\omega t + \frac{1}{5} \sin 5\omega t + \frac{1}{7} \sin 7\omega t + \cdots + \frac{1}{n} \sin n\omega t \right)$$

- 1st harmonic = $\frac{1}{1} = 1$
- 3rd harmonic = $\frac{1}{3} \approx 0.3333$
- 5th harmonic = $\frac{1}{5} = 0.2000$
- 7th harmonic = $\frac{1}{7} \approx 0.1429$
- 9th harmonic = $\frac{1}{9} \approx 0.1111$

As you can see, the output of our simple DFT algorithm closely approximates these theoretical results.

By modifying just the code within the `f(x)` function we may compute the harmonic content of different wave-shapes. The next several examples will show the modified `f(x)` function code and the resulting output of this DFT algorithm.

6.4.2 DFT of a sine wave

First, we will re-code `f(x)` to generate a simple sine wave. The argument x passed to this function is an integer number starting at zero and incrementing to 127, representing a sequence of samples spanning one period of the fundamental frequency, and so some scaling arithmetic is necessary to convert this domain into a value in radians from 0 to 2π suitable for the `sin()` function:

```
float f(int x)    // Sine wave function
{
  return sin(2 * M_PI * x / 128.0);
}
```

```
Harmonic = 1 -- Normalized weight = 1.000000
Harmonic = 2 -- Normalized weight = 0.000000
Harmonic = 3 -- Normalized weight = 0.000000
Harmonic = 4 -- Normalized weight = 0.000000
Harmonic = 5 -- Normalized weight = 0.000000
Harmonic = 6 -- Normalized weight = 0.000000
Harmonic = 7 -- Normalized weight = 0.000000
Harmonic = 8 -- Normalized weight = 0.000000
Harmonic = 9 -- Normalized weight = 0.000000
```

Not surprisingly, the result is a strong first harmonic and no other harmonics. Also, we get the same results if we replace the sine function with a cosine function in `f(x)`: in either case, a plain sinusoid only has one harmonic component, and that is the first harmonic.

6.4.3 DFT of a delta function

As another test of our DFT algorithm, we will re-code `f(x)` to output a *delta function*, which is nothing more than the briefest of impulses. A delta function consists of a "spike"[10] at time zero followed (and preceded) by values of zero:

```
float f(int x)  // Delta impulse function
{
  if (x == 0)
    return 1.0;

  else
    return 0.0;
}
```

```
Harmonic = 1 -- Normalized weight = 1.000000
Harmonic = 2 -- Normalized weight = 1.000000
Harmonic = 3 -- Normalized weight = 1.000000
Harmonic = 4 -- Normalized weight = 1.000000
Harmonic = 5 -- Normalized weight = 1.000000
Harmonic = 6 -- Normalized weight = 1.000000
Harmonic = 7 -- Normalized weight = 1.000000
Harmonic = 8 -- Normalized weight = 1.000000
Harmonic = 9 -- Normalized weight = 1.000000
```

The result is *all* harmonics at equal strength, which is what the Fourier transform predicts for a delta function: a constant-valued function in the frequency domain. In other words, an infinitesimally brief impulse is equivalent to a superposition of *all* frequencies.

This is a good example of our **steepness rule** in action: a delta function consists of nothing but steepness, being a "spike" up and down over the briefest possible time interval. As such, it contains all frequencies, which of course includes the nine harmonic frequencies shown.

If we consider carefully how the DFT algorithm works, it becomes evident why this must be so, and precisely how every frequency's value must have the same normalized value. The very first sample (`sample = 0`) is the only one where the delta function is not zero, and therefore this will be the only sample where any of the sums tallied in the program accumulate any value. Furthermore, the only sums accumulating value during this sample must be the *cosine* sums because the sine function is zero at an angle of zero, while cosine is one at an angle of zero. Therefore, every cosine function multiplied by the delta impulse function will increment its sum by one. This must include every cosine *of every conceivable frequency* and not just the select harmonics tested by our DFT algorithm. Therefore, based on the criteria of the DFT algorithm, a delta function must contain *all* cosine terms, of *every* frequency.

[10] A true *Dirac delta function* actually consists of an infinite-magnitude spike with zero width, but having an enclosed area equal to unity. We cannot emulate that in procedural code, but we may approximate it!

In practice there is no such thing as a real delta impulse function. A function consisting of a pulse of infinitesimal width defies physical implementation, but nevertheless is useful as a theoretical tool, and serves as a limit for very brief (real) pulses. The practical lesson to learn here is that the spectra of real pulse signals approaches uniformity as the width of the pulse approaches zero – i.e. the briefer the pulse duration, the wider the spread of constituent frequencies. This means any circuitry tasked with amplifying, attenuating, or otherwise processing this pulse signal must contend with a broad span of frequencies, and failure to properly process *all* of the frequencies within that pulse signal invariably corrupts the pulse in some way.

6.4.4 DFT of two sine waves

Next, we will try modifying `f(x)` to generate a superposition of two sine waves, one at 5× our assumed fundamental frequency, and another at 8× the fundamental:

```
float f(int x)  // Dual sine waves
{
  return sin(5 * 2 * M_PI * x / 128.0)
  + sin(8 * 2 * M_PI * x / 128.0);
}
```

From this we would expect a harmonic spectrum consisting of a 5th harmonic and 8th harmonic, and nothing else. What we obtain looks strange at first, though:

```
Harmonic = 1 -- Normalized weight = 1.000000
Harmonic = 2 -- Normalized weight = 3.847159
Harmonic = 3 -- Normalized weight = 4.564931
Harmonic = 4 -- Normalized weight = 5.773269
Harmonic = 5 -- Normalized weight = 159252960.000000
Harmonic = 6 -- Normalized weight = 4.397245
Harmonic = 7 -- Normalized weight = 2.756398
Harmonic = 8 -- Normalized weight = 159252960.000000
Harmonic = 9 -- Normalized weight = 1.914945
```

The amplitudes of the 5th and 8th harmonics are enormous, while the others are meager by comparison. Remember, though, that our DFT algorithm *normalizes* all harmonic amplitudes to that of the first harmonic, which in this particular case should be virtually nonexistent. Therefore, the first harmonic registers with a weight of 1, the 5th and 8th with very large weights, and the others about as small as the first harmonic (in comparison with the 5th and 8th). So, even with the crude nature of this algorithm, we get a spectral response that makes sense for the test waveform.

This is a good example of our **superposition rule**, where the spectrum of two superimposed waves is the superposition of those waves' spectra. The 5th harmonic wave consisted of a single peak in its "spectrum" as did the 8th harmonic wave. When these two waves were added in their time domains, the result is a spectrum consisting of those two frequency peaks, no more and no less.

6.4.5 DFT of an amplitude-modulated sine wave

Next, we will re-code `f(x)` to generate an *amplitude-modulated* waveform: the product of a sine wave at 2× the assumed fundamental and another sine wave at 5× the fundamental.

```
float f(int x)  // Mixed sine waves (AM)
{
  return sin(2 * 2 * M_PI * x / 128.0)
  * sin(5 * 2 * M_PI * x / 128.0);
}
```

Modulation theory predicts that "mixing" two sinusoids in this manner will result in two completely new frequencies: one being the sum of the two mixed frequencies, and the other being the difference of the two mixed frequencies. So, for one sine wave oscillating at 2ω and another at 5ω, we would expect one sinusoid at $(5+2)\omega$ and another at $(5-2)\omega$.

```
Harmonic = 1 -- Normalized weight = 1.000000
Harmonic = 2 -- Normalized weight = 0.077597
Harmonic = 3 -- Normalized weight = 17697600.000000
Harmonic = 4 -- Normalized weight = 0.180189
Harmonic = 5 -- Normalized weight = 0.128424
Harmonic = 6 -- Normalized weight = 0.152171
Harmonic = 7 -- Normalized weight = 17697598.000000
Harmonic = 8 -- Normalized weight = 0.150321
Harmonic = 9 -- Normalized weight = 0.557960
```

True to form, the result is a pair of harmonics in the spectrum, a 3rd harmonic and a 7th harmonic.

This is an excellent example of our **non-linear systems rule**: when signals pass through non-linear systems, new frequencies arise. Multiplication of two independent signals is definitely nonlinear, as doubling both signals' amplitudes does *not* result in a doubled output amplitude. What came into this system was a 2nd and 5th harmonic, but what left was a 3rd and 7th harmonic.

6.4.6 DFT of a full-rectified sine wave

Next, we will re-code `f(x)` to generate the first half (i.e. positive half) of a sine wave. This is all we need to simulate a full-wave rectified sinusoid since all other half-periods of that wave will identical to the first. To represent this in code, we just take the same line used for the sine wave and eliminate the `2` multiplier. In other words, instead of calculating $\sin\left(\frac{2\pi x}{128}\right)$ we compute $\sin\left(\frac{\pi x}{128}\right)$:

```
float f(int x)  // Full-rectified sine wave
{
  return sin(M_PI * x / 128.0);
}
```

The result is shown here:

```
Harmonic = 1 -- Normalized weight = 1.000000
Harmonic = 2 -- Normalized weight = 0.200121
Harmonic = 3 -- Normalized weight = 0.085852
Harmonic = 4 -- Normalized weight = 0.047763
Harmonic = 5 -- Normalized weight = 0.030450
Harmonic = 6 -- Normalized weight = 0.021127
Harmonic = 7 -- Normalized weight = 0.015534
Harmonic = 8 -- Normalized weight = 0.011915
Harmonic = 9 -- Normalized weight = 0.009439
```

Fourier theory predicts the relative amplitudes of each harmonic for a full-rectified sine wave diminish by a factor of $\frac{1}{4n^2-1}$ where n is the harmonic number. Therefore, if the first harmonic has an amplitude of $\frac{1}{3}$, then the second harmonic will have an amplitude of $\frac{1}{15}$, the third harmonic an amplitude of $\frac{1}{35}$, etc. If we normalize all the amplitudes to that of the first harmonic, the relative amplitudes will be as follows:

- 1st harmonic $= \frac{3}{3} = 1$
- 2nd harmonic $= \frac{3}{15} = \frac{1}{5} = 0.200$
- 3rd harmonic $= \frac{3}{35} \approx 0.0857$
- 4th harmonic $= \frac{3}{63} = \frac{1}{21} \approx 0.0476$
- 5th harmonic $= \frac{3}{99} = \frac{1}{33} \approx 0.0303$

As you can see, the output of our simple DFT algorithm closely approximates these theoretical results.

This is a good example of our **symmetry rule**. A rectified sine wave does not have the same shape when inverted, and so we know it must contain even-numbered harmonics. Contrast this against symmetrical waveforms such as the square wave from the original code example, generating a spectrum consisting only of odd-numbered harmonics.

6.5 Spectrum analyzer in C++

This program builds on the foundation of the Discrete Fourier Transform (DFT) from the previous section, but instead of displaying only the normalized harmonic amplitudes this program outputs a comma-separated value (CSV) file that may be plotted using any spreadsheet application (e.g. Microsoft Excel) or mathematical visualizing application (e.g. `gnuplot`).

I happened to use `gnuplot` to generate the spectra. My gnuplot script is as follows, saved to a file named `script.txt`:

```
set datafile separator ","
set xrange [0:10.0]
set style line 1 lw 2 lc rgb "red"
plot 'data.csv' using 1:2 with lines ls 1
```

All C++ programs were compiled using `g++` and run with text output redirected to a file named `data.csv` using the following command-line instructions:

```
g++ main.cpp ; ./a.out > data.csv
```

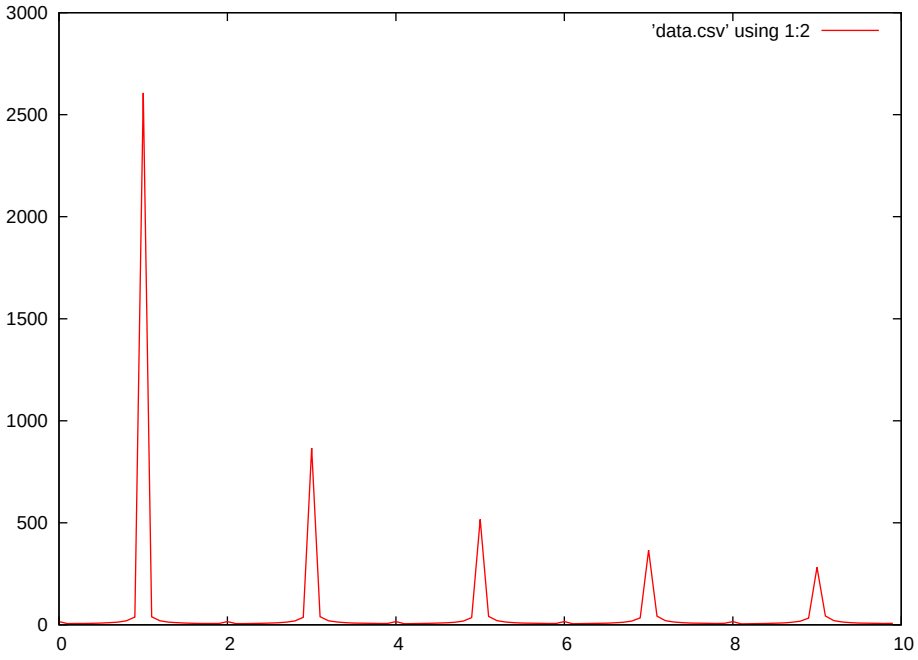
Then, after the comma-separated value file was populated with data from the C++ program's execution, I run `gnuplot` using the following command:

```
gnuplot -p script.txt
```

```
#include <iostream>
#include <math.h>
using namespace std;

#define MAX 4096
#define CYCLES 10

float f(int x);

int main(void)
{
  int sample;
  float iharm, sinsum, cossum, polarsum;

  for (iharm = 0.0; iharm < 10.0; iharm = iharm + 0.1)
    {
      sinsum = 0;
      cossum = 0;

      for (sample = 0; sample < MAX; ++sample)
        {
          sinsum = sinsum + (f(sample) * (sin(CYCLES*sample*iharm*2*M_PI/MAX)));
          cossum = cossum + (f(sample) * (cos(CYCLES*sample*iharm*2*M_PI/MAX)));
        }

      polarsum = sqrt(pow(cossum, 2) + pow(sinsum, 2));

      cout << iharm << " , " << polarsum << endl;
    }

  return 0;
}

float f(int x)
{
  // (return value of function to be analyzed here)
}
```

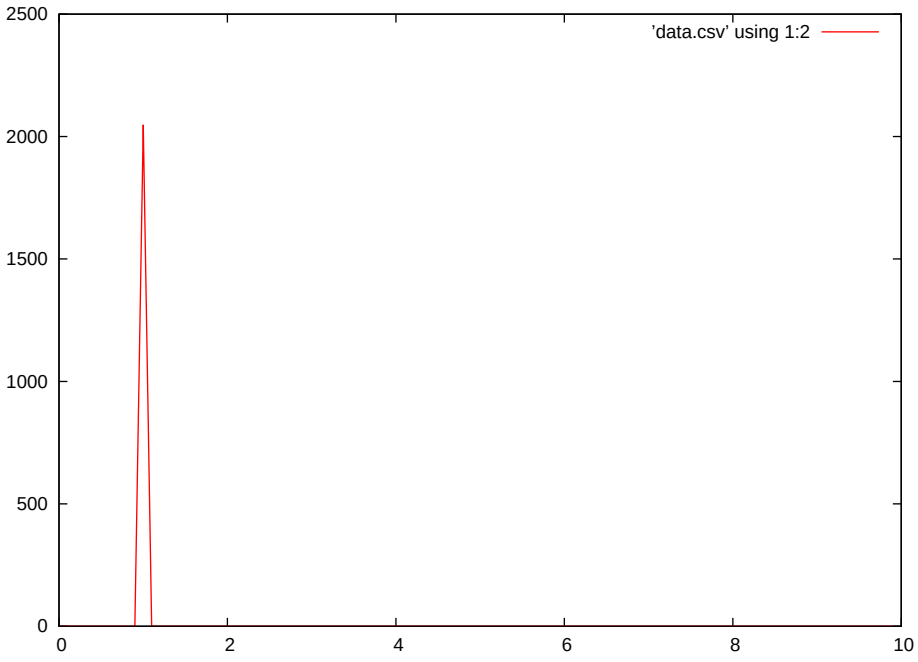
6.5.1 Spectrum of a square wave

```
float f(int x) // Square wave
{
  if ((x % (MAX / CYCLES)) < (0.5 * MAX / CYCLES))
    return 1.0;

  else
    return -1.0;
}
```



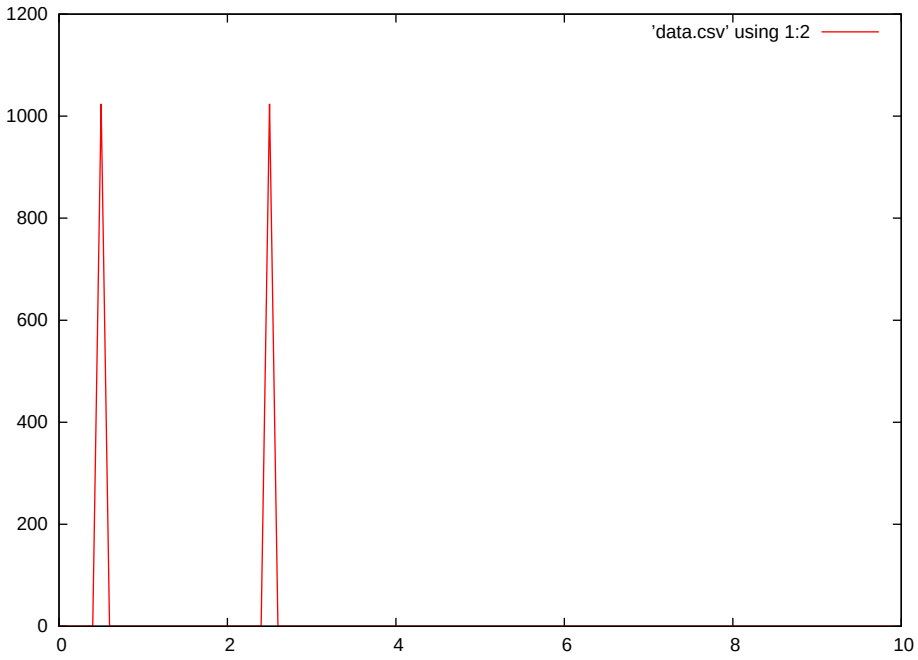
6.5.2 Spectrum of a sine wave

```
float f(int x) // Sine wave
{
  return sin(CYCLES*x*2*M_PI/MAX);
}
```



6.5.3 Spectrum of a sine wave product

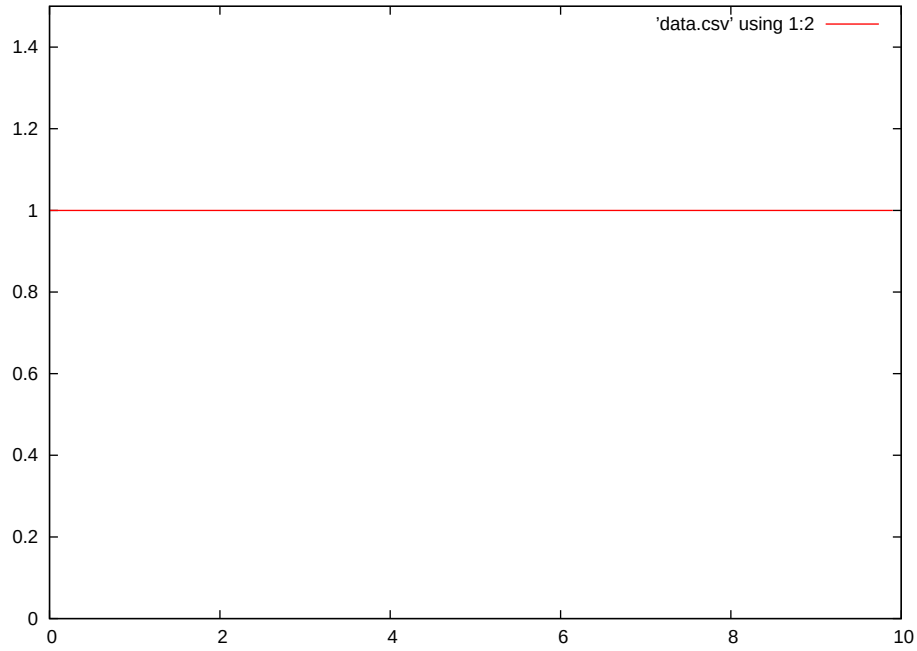
```
float f(int x) // Product of f and 1.5f sine waves
{
  return sin(CYCLES*x*2*M_PI/MAX) * sin(CYCLES*1.5*x*2*M_PI/MAX);
}
```



Note the two peaks at $0.5f$ and $2.5f$: frequencies representing the difference and sum, respectively, of the original sinusoids.

6.5.4 Spectrum of an impulse

```
float f(int x) // Unity impulse function at x = 0
{
  if (x == 0)
    return 1;

  else
    return 0;
}
```



Note how the impulse is equivalent to a spectrum consisting of *all* frequencies. Since the amplitude of the spectrum is much less than in previous examples, I used a different y-axis range in `gnuplot` than in the other simulations:

```
set datafile separator ","
set xrange [0:10.0]
set yrange [0:1.5]
set style line 1 lw 2 lc rgb "red"
plot 'data.csv' using 1:2 with lines ls 1
```

Chapter 7

Questions

This learning module, along with all others in the ModEL collection, is designed to be used in an inverted instructional environment where students independently read[1] the tutorials and attempt to answer questions on their own *prior* to the instructor's interaction with them. In place of lecture[2], the instructor engages with students in Socratic-style dialogue, probing and challenging their understanding of the subject matter through inquiry.

Answers are not provided for questions within this chapter, and this is by design. Solved problems may be found in the Tutorial and Derivation chapters, instead. The goal here is *independence*, and this requires students to be challenged in ways where others cannot think for them. Remember that you always have the tools of *experimentation* and *computer simulation* (e.g. SPICE) to explore concepts!

The following lists contain ideas for Socratic-style questions and challenges. Upon inspection, one will notice a strong theme of *metacognition* within these statements: they are designed to foster a regular habit of examining one's own thoughts as a means toward clearer thinking. As such these sample questions are useful both for instructor-led discussions as well as for self-study.

[1] Technical reading is an essential academic skill for any technical practitioner to possess for the simple reason that the most comprehensive, accurate, and useful information to be found for developing technical competence is in textual form. Technical careers in general are characterized by the need for continuous learning to remain current with standards and technology, and therefore any technical practitioner who cannot read well is handicapped in their professional development. An excellent resource for educators on improving students' reading prowess through intentional effort and strategy is the book textitReading For Understanding – How Reading Apprenticeship Improves Disciplinary Learning in Secondary and College Classrooms by Ruth Schoenbach, Cynthia Greenleaf, and Lynn Murphy.

[2] Lecture is popular as a teaching method because it is easy to implement: any reasonably articulate subject matter expert can talk to students, even with little preparation. However, it is also quite problematic. A good lecture always makes complicated concepts seem easier than they are, which is bad for students because it instills a false sense of confidence in their own understanding; reading and re-articulation requires more cognitive effort and serves to verify comprehension. A culture of teaching-by-lecture fosters a debilitating dependence upon direct personal instruction, whereas the challenges of modern life demand independent and critical thought made possible only by gathering information and perspectives from afar. Information presented in a lecture is ephemeral, easily lost to failures of memory and dictation; text is forever, and may be referenced at any time.

General challenges following tutorial reading

- Summarize as much of the text as you can in one paragraph of your own words. A helpful strategy is to explain ideas as you would for an intelligent child: as simple as you can without compromising too much accuracy.
- Simplify a particular section of the text, for example a paragraph or even a single sentence, so as to capture the same fundamental idea in fewer words.
- Where did the text make the most sense to you? What was it about the text's presentation that made it clear?
- Identify where it might be easy for someone to misunderstand the text, and explain why you think it could be confusing.
- Identify any new concept(s) presented in the text, and explain in your own words.
- Identify any familiar concept(s) such as physical laws or principles applied or referenced in the text.
- Devise a proof of concept experiment demonstrating an important principle, physical law, or technical innovation represented in the text.
- Devise an experiment to disprove a plausible misconception.
- Did the text reveal any misconceptions you might have harbored? If so, describe the misconception(s) and the reason(s) why you now know them to be incorrect.
- Describe any useful problem-solving strategies applied in the text.
- Devise a question of your own to challenge a reader's comprehension of the text.

General follow-up challenges for assigned problems

- Identify where any fundamental laws or principles apply to the solution of this problem, especially before applying any mathematical techniques.
- Devise a thought experiment to explore the characteristics of the problem scenario, applying known laws and principles to mentally model its behavior.
- Describe in detail your own strategy for solving this problem. How did you identify and organized the given information? Did you sketch any diagrams to help frame the problem?
- Is there more than one way to solve this problem? Which method seems best to you?
- Show the work you did in solving this problem, even if the solution is incomplete or incorrect.
- What would you say was the most challenging part of this problem, and why was it so?
- Was any important information missing from the problem which you had to research or recall?
- Was there any extraneous information presented within this problem? If so, what was it and why did it not matter?
- Examine someone else's solution to identify where they applied fundamental laws or principles.
- Simplify the problem from its given form and show how to solve this simpler version of it. Examples include eliminating certain variables or conditions, altering values to simpler (usually whole) numbers, applying a limiting case (i.e. altering a variable to some extreme or ultimate value).
- For quantitative problems, identify the real-world meaning of all intermediate calculations: their units of measurement, where they fit into the scenario at hand. Annotate any diagrams or illustrations with these calculated values.
- For quantitative problems, try approaching it qualitatively instead, thinking in terms of "increase" and "decrease" rather than definite values.
- For qualitative problems, try approaching it quantitatively instead, proposing simple numerical values for the variables.
- Were there any assumptions you made while solving this problem? Would your solution change if one of those assumptions were altered?
- Identify where it would be easy for someone to go astray in attempting to solve this problem.
- Formulate your own problem based on what you learned solving this one.

General follow-up challenges for experiments or projects

- In what way(s) was this experiment or project easy to complete?
- Identify some of the challenges you faced in completing this experiment or project.

- Show how thorough documentation assisted in the completion of this experiment or project.
- Which fundamental laws or principles are key to this system's function?
- Identify any way(s) in which one might obtain false or otherwise misleading measurements from test equipment in this system.
- What will happen if (component X) fails (open/shorted/etc.)?
- What would have to occur to make this system unsafe?

7.1 Conceptual reasoning

These questions are designed to stimulate your analytic and synthetic thinking[3]. In a Socratic discussion with your instructor, the goal is for these questions to prompt an extended dialogue where assumptions are revealed, conclusions are tested, and understanding is sharpened. Your instructor may also pose additional questions based on those assigned, in order to further probe and refine your conceptual understanding.

Questions that follow are presented to challenge and probe your understanding of various concepts presented in the tutorial. These questions are intended to serve as a guide for the Socratic dialogue between yourself and the instructor. Your instructor's task is to ensure you have a sound grasp of these concepts, and the questions contained in this document are merely a means to this end. Your instructor may, at his or her discretion, alter or substitute questions for the benefit of tailoring the discussion to each student's needs. The only absolute requirement is that each student is challenged and assessed at a level equal to or greater than that represented by the documented questions.

It is far more important that you convey your *reasoning* than it is to simply convey a correct answer. For this reason, you should refrain from researching other information sources to answer questions. What matters here is that *you* are doing the thinking. If the answer is incorrect, your instructor will work with you to correct it through proper reasoning. A correct answer without an adequate explanation of how you derived that answer is unacceptable, as it does not aid the learning or assessment process.

You will note a conspicuous lack of answers given for these conceptual questions. Unlike standard textbooks where answers to every other question are given somewhere toward the back of the book, here in these learning modules students must rely on other means to check their work. The best way by far is to debate the answers with fellow students and also with the instructor during the Socratic dialogue sessions intended to be used with these learning modules. Reasoning through challenging questions with other people is an excellent tool for developing strong reasoning skills.

Another means of checking your conceptual answers, where applicable, is to use circuit simulation software to explore the effects of changes made to circuits. For example, if one of these conceptual questions challenges you to predict the effects of altering some component parameter in a circuit, you may check the validity of your work by simulating that same parameter change within software and seeing if the results agree.

[3] *Analytical* thinking involves the "disassembly" of an idea into its constituent parts, analogous to dissection. *Synthetic* thinking involves the "assembly" of a new idea comprised of multiple concepts, analogous to construction. Both activities are high-level cognitive skills, extremely important for effective problem-solving, necessitating frequent challenge and regular practice to fully develop.

7.1.1 Reading outline and reflections

"Reading maketh a full man; conference a ready man; and writing an exact man" – Francis Bacon

Francis Bacon's advice is a blueprint for effective education: reading provides the learner with knowledge, writing focuses the learner's thoughts, and critical dialogue equips the learner to confidently communicate and apply their learning. Independent acquisition and application of knowledge is a powerful skill, well worth the effort to cultivate. To this end, students should read these educational resources closely, journal their own reflections on the reading, and discuss in detail their findings with classmates and instructor(s). You should be able to do all of the following after reading any instructional text:

☑ Briefly SUMMARIZE THE TEXT in the form of a journal entry documenting your learning as you progress through the course of study. Share this summary in dialogue with your classmates and instructor. Journaling is an excellent self-test of thorough reading because you cannot clearly express what you have not read or did not comprehend.

☑ Demonstrate ACTIVE READING STRATEGIES, including verbalizing your impressions as you read, simplifying long passages to convey the same ideas using fewer words, annotating text and illustrations with your own interpretations, working through mathematical examples shown in the text, cross-referencing passages with relevant illustrations and/or other passages, identifying problem-solving strategies applied by the author, etc. Technical reading is a special case of problem-solving, and so these strategies work precisely because they help solve any problem: paying attention to your own thoughts (metacognition), eliminating unnecessary complexities, identifying what makes sense, paying close attention to details, drawing connections between separated facts, and noting the successful strategies of others.

☑ Identify IMPORTANT THEMES, especially GENERAL LAWS and PRINCIPLES, expounded in the text and express them in the simplest of terms as though you were teaching an intelligent child. This emphasizes connections between related topics and develops your ability to communicate complex ideas to anyone.

☑ Form YOUR OWN QUESTIONS based on the reading, and then pose them to your instructor and classmates for their consideration. Anticipate both correct and incorrect answers, the incorrect answer(s) assuming one or more plausible misconceptions. This helps you view the subject from different perspectives to grasp it more fully.

☑ Devise EXPERIMENTS to test claims presented in the reading, or to disprove misconceptions. Predict possible outcomes of these experiments, and evaluate their meanings: what result(s) would confirm, and what would constitute disproof? Running mental simulations and evaluating results is essential to scientific and diagnostic reasoning.

☑ Specifically identify any points you found CONFUSING. The reason for doing this is to help diagnose misconceptions and overcome barriers to learning.

7.1.2 Foundational concepts

Correct analysis and diagnosis of electric circuits begins with a proper understanding of some basic concepts. The following is a list of some important concepts referenced in this module's full tutorial. Define each of them in your own words, and be prepared to illustrate each of these concepts with a description of a practical example and/or a live demonstration.

Energy

Conservation of Energy

Capacitance

Inductance

Reactance

Limiting cases as a problem-solving strategy

Open

Short

Voltage divider

Bode plot

Cutoff

Passband

Stopband

Bandwidth

Roll-off

Frequency domain

Sinusoidal decomposition (i.e. Fourier's Theorem)

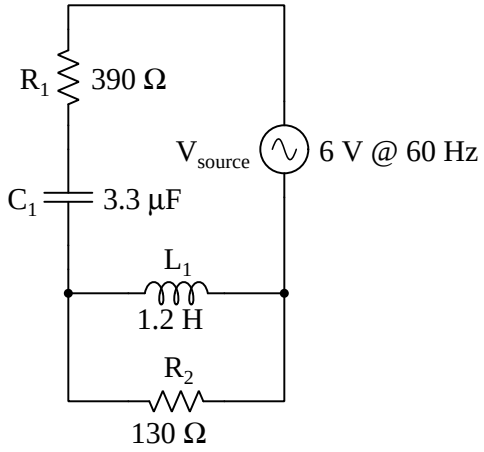
Time domain

7.1.3 Explaining the meaning of calculations

An unfortunate tendency among beginning students in any quantitative discipline is to perform calculations without regard for the real-world meanings of the values, and also to follow mathematical formulae without considering the general principles embodied in each. To ignore concepts while performing calculations is a serious error for a variety of reasons, not the least of which being an increased likelihood of computing results that turn out to be nonsense.

In the spirit of honoring concepts, I present to you a quantitative problem where all the calculations have been done for you, but all variable labels, units, and other identifying data have been stripped away. Your task is to *assign proper meaning* to each of the numbers, identifying the correct unit of measurement in each case, explaining the significance of each value by describing where it "fits" into the circuit being analyzed, and identifying the general principle employed at each step.

Here is the schematic diagram of the circuit:



Here are all the calculations performed in order from first to last:

1. $\frac{1}{(2\pi 60)(3.3\times 10^{-6})} = 803.82$
2. $(2\pi 60)(1.2) = 452.39$
3. $\frac{1}{\frac{1}{130\ \angle 0^o}+\frac{1}{452.39\ \angle 90^o}} = 124.94\ \angle 16.03^o$
4. $124.94\ \angle 16.03^o + 390\ \angle 0^o + 803.82\ \angle -90^o = 923.05\ \angle -56.45^o$
5. $\frac{6\ \angle 0^o}{923.05\ \angle -56.45^o} = 6.500 \times 10^{-3}\ \angle 56.45^o$
6. $(6.500 \times 10^{-3}\ \angle 56.45^o)(390\ \angle 0^o) = 2.535\ \angle 56.45^o$
7. $(6.500 \times 10^{-3}\ \angle 56.45^o)(803.82\ \angle -90^o) = 5.225\ \angle -33.55^o$
8. $(6.500 \times 10^{-3}\ \angle 56.45^o)(124.94\ \angle 16.03^o) = 0.8122\ \angle 72.49^o$

9. $2.535 \angle 56.45^o + 5.225 \angle -33.55^o + 0.8122 \angle 72.49^o = 6 \angle 0^o$
10. $\frac{0.8122 \angle 72.49^o}{130 \angle 0^o} = 6.247 \times 10^{-3} \angle 72.49^o$
11. $\frac{0.8122 \angle 72.49^o}{452.39 \angle 90^o} = 1.795 \times 10^{-3} \angle -17.51^o$
12. $6.247 \times 10^{-3} \angle 72.49^o + 1.795 \times 10^{-3} \angle -17.51^o = 6.500 \times 10^{-3} \angle 56.45^o$
13. $\cos -56.45^o = 0.5526$
14. $(6)(6.500 \times 10^{-3}) = 39.00 \times 10^{-3}$
15. $(39.00 \times 10^{-3})(\cos -56.45^o) = 21.55 \times 10^{-3}$
16. $(39.00 \times 10^{-3})(\sin -56.45^o) = 32.51 \times 10^{-3}$
17. $\sqrt{(21.55 \times 10^{-3})^2 + (32.51 \times 10^{-3})^2} = 39.00 \times 10^{-3}$

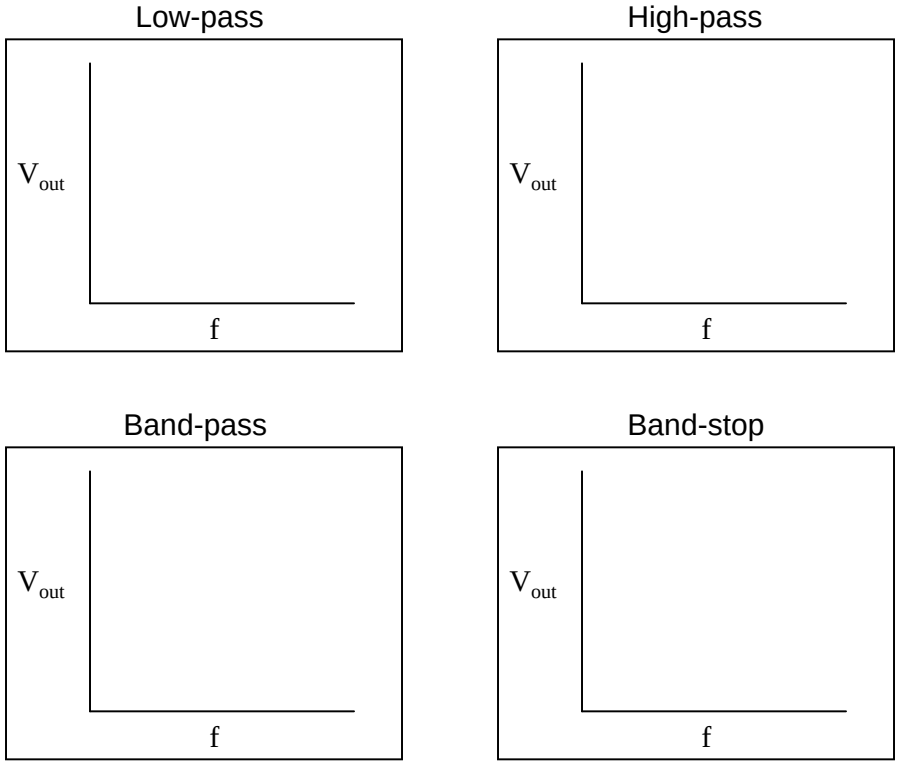
Explain what each value means in the circuit, identify its unit of measurement, and identify the general principle used to compute it!

Challenges

- Explain how you can check your own thinking as you solve quantitative problems, to avoid the dilemma of just "crunching numbers" to get an answer.
- Do you see any alternative paths to a solution, involving specific calculations not shown above?

7.1.4 Bode plots and bandwidths

Plot the typical frequency responses of four different filter circuits, showing signal output (amplitude) on the vertical axis and frequency on the horizontal axis:



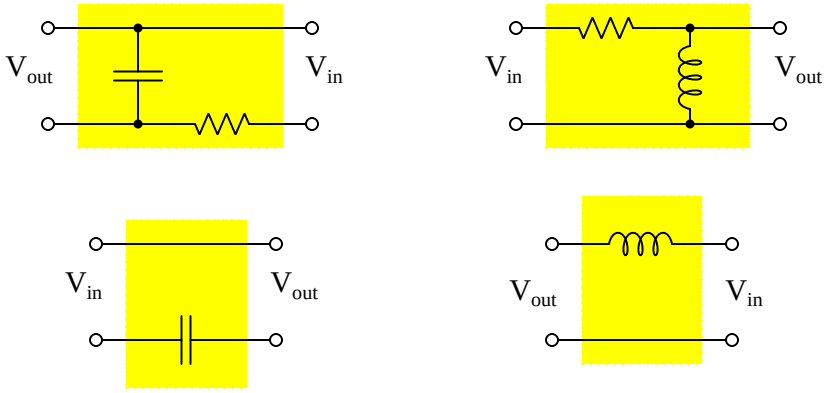
Also, identify and label the *bandwidth* of the filter circuit on each plot.

Challenges

- Why is the 70.7% signal amplitude point considered the definition of cutoff frequency for a filter? Where does this odd number come from?

7.1.5 Identifying filter types

Examine the following schematic diagrams of filter networks, and classify the purpose of each as either *low-pass*, *high-pass*, *band-pass*, or *band-stop*:



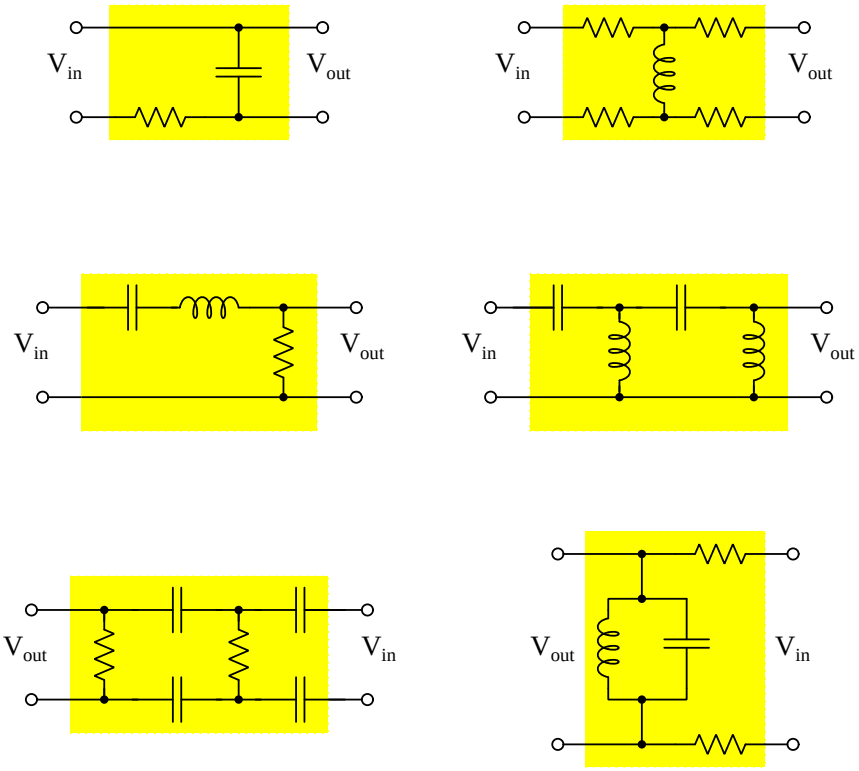
For each case, explain the reasoning behind your classification of filter type.

Challenges

- An applicable problem-solving strategy for a qualitative problem such as this is to convert it into a *quantitative* problem. Describe how you could apply this strategy here, and explain how that strategy could help you classify each filter type.
- After identifying each filter classification, propose a modification to the network that would alter it to be another type of filter.
- After identifying each filter classification, identify a change you could make to its component value(s) to *increase* its characteristic frequency.
- Describe a procedure by which you could empirically determine the classification of a filter circuit, assuming you had no access to its schematic diagram nor to its internal construction.
- Describe a procedure by which you could empirically determine the cutoff frequency(ies) of a filter circuit, assuming you had no access to its schematic diagram nor to its internal construction.

7.1.6 Identifying (more) filter types

Examine the following schematic diagrams of filter networks, and classify the purpose of each as either *low-pass*, *high-pass*, *band-pass*, or *band-stop*:



For each case, explain the reasoning behind your classification of filter type.

Challenges

- An applicable problem-solving strategy for a qualitative problem such as this is to convert it into a *quantitative* problem. Describe how you could apply this strategy here, and explain how that strategy could help you classify each filter type.
- After identifying each filter classification, propose a modification to the network that would alter it to be another type of filter.
- After identifying each filter classification, identify a change you could make to its component value(s) to *increase* its characteristic frequency.
- For any band-type filter shown here, identify a change you could make to its component value(s) to *increase* its quality factor without altering its center frequency.

- Describe a procedure by which you could empirically determine the classification of a filter circuit, assuming you had no access to its schematic diagram nor to its internal construction.
- Describe a procedure by which you could empirically determine the cutoff frequency(ies) of a filter circuit, assuming you had no access to its schematic diagram nor to its internal construction.
- Which of these filter networks appear designed with *balanced* signals in mind?

7.1.7 Filter truth table

Elementary filter networks fall into one of four basic types: *low-pass*, *high-pass*, *band-pass*, and *band-stop*. Each of these may be related to "limiting case" conditions of low and high frequency. If we consider these limiting cases and draw the possibilities as a *truth table* similar to how we characterize digital logic functions, we see the four possibilities map to four combinations of passing/blocking for those limit cases:

$f = 0$ **Hz**	$f = \infty$ **Hz**	**Filter type**
Block	Block	
Block	Pass	
Pass	Block	
Pass	Pass	

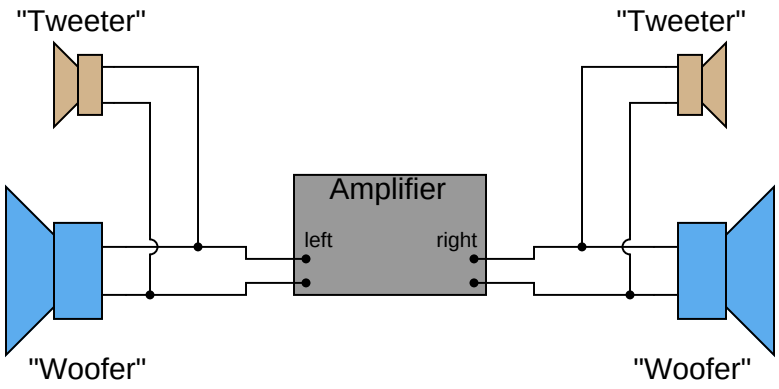
Complete this truth table.

Challenges

- Explain why the "limiting cases" problem-solving strategy is such a powerful tool for analyzing filter circuits.

7.1.8 Tweeter enhancement

Suppose you were installing a high-power stereo system in your car, and you wanted to build a simple filter for the "tweeter" (high-frequency) speakers so that no bass (low-frequency) power is wasted in these speakers. Modify the schematic diagram below with a filter circuit of your choice:

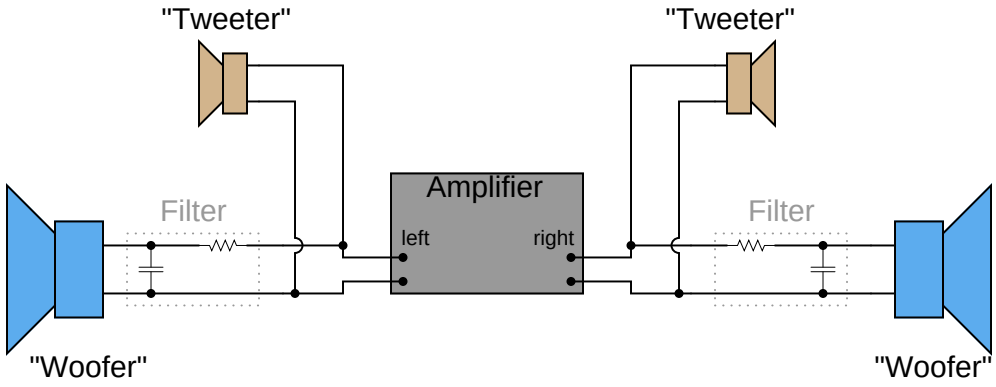


Challenges

- Explain how you would determine the correct *sizing* of the additional component(s).

7.1.9 Woofer enhancement

Suppose a friend wanted to install filter networks in the "woofer" section of their stereo system, to prevent high-frequency power from being wasted in speakers incapable of reproducing those frequencies. To this end, your friend installs the following resistor-capacitor networks:



After examining this schematic, you see that your friend has the right idea in mind, but implemented it incorrectly. These filter circuits would indeed block high-frequency signals from getting to the woofers, but they would not actually accomplish the stated goal of minimizing wasted power.

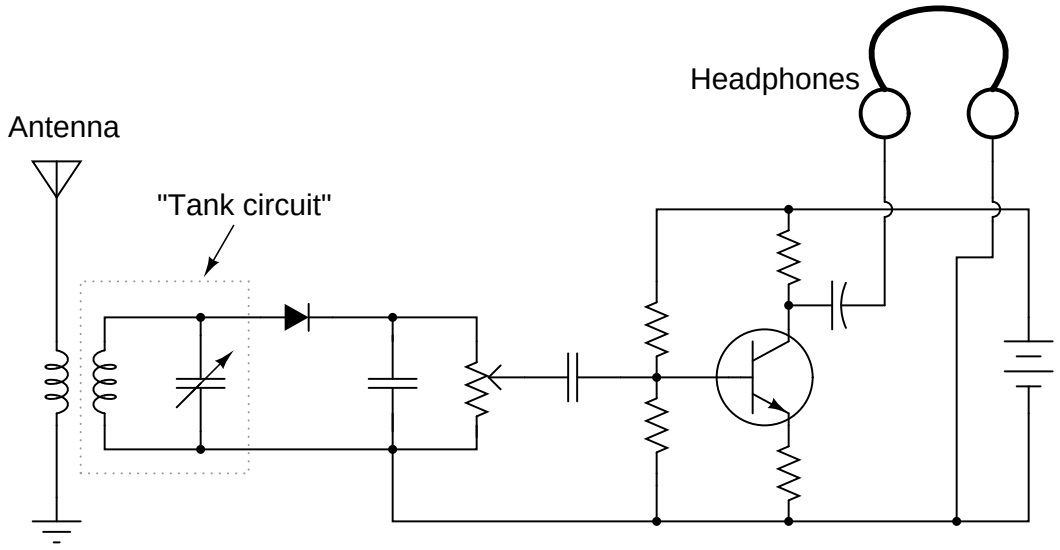
What would you recommend to your friend in lieu of this circuit design?

Challenges

- Explain how you would determine the correct *sizing* of the new component(s).

7.1.10 AM radio tuner

The following schematic shows the workings of a simple AM radio receiver, with transistor amplifier:



The "tank circuit" formed of a parallel-connected inductor and capacitor network performs a very important filtering function in this circuit. Describe what this filtering function is.

Challenges

- Wow might a variable capacitor be constructed, to suit the needs of a circuit such as this? Note that the capacitance range for a tuning capacitor such as this is typically in the pico-Farad range.
- If we happen to be listening to a station broadcasting at 1000 kHz and we want to change to a station broadcasting at 1150 kHz, what do we have to alter in the circuit?

7.1.11 Output jack on analog VOMs

A *VOM* (Volt-Ohm-Milliammeter) is the analog equivalent of a digital multimeter (DMM), and was once[4] the test instrument of choice for electrical/electronic technicians and engineers. A example of a high-quality VOM (the Simpson model 260) is shown in the following photograph:

A sensitive moving-coil analog meter mechanism provides visual indication of the measured quantity: the stronger the signal, the farther toward the right the pointer (needle) moves. Multiple measurement ranges are achieved by placing precision "multiplier" resistors in series with this sensitive analog meter: the more resistance in series with the meter coil, the greater the voltage necessary to drive the pointer full-scale to the right.

Most of the measurements made with such an instrument are with the red and black test leads plugged into the `Common (-)` and `(+)` jacks on the VOM's face. However, certain specialized functions and ranges require that the red test lead (and sometimes the black test lead as well) be plugged into other jacks[5].

[4] It is worth noting that VOMs are still manufactured at the time of this writing. The Simpson Electric Company, for example, *still* manufactures several VOM models in the year 2019. While this may seem anachronistic, there is a lot to be said in defense of using a high-quality VOM in the 21st century. VOMs require no battery to measure voltage or current, and an analog meter needle still surpasses digital displays for being able to visually indicate transient pulses.

[5] For example, to use this VOM to measure currents upward of 10 Amperes, the red and black test leads would need to be plugged into the `+10A` and `-10A` jacks, respectively, and the selector switch turned to the `10 mA/Amperes` position.

One such specialized function is called `Output`, and is represented by its own dedicated jack on the VOM face, into which the red test lead is plugged. This function is so named because it is generally used when testing the output of an audio amplifier. A description of this function is quoted from page 17 of the Simpson 270 Series 5 VOM instruction manual:

> Measuring the AC component of an Output Voltage where both AC and DC voltage levels exist is sometimes necessary. This occurs primarily in amplifier circuits. The 270 has a 0.1 mfd, 400 volt capacitor in series with the OUTPUT jack. The capacitor blocks the DC component of the current in the test circuit, but allows the AC or desired component to pass on to the indicating instrument circuit. The blocking capacitor may alter the AC response at low frequencies but is usually ignored at audio frequencies.

Explain how the 0.1 μF capacitor performs a *filtering* function within this VOM. Specifically, what type of filter network does the capacitor form when connected to the rest of the meter circuitry?

Also, explain how the presence of this "blocking" capacitor may affect the meter's response at low frequencies. Do you think it will make the meter register falsely low at low frequencies, or falsely high? Explain your reasoning.

This altered low-frequency AC meter response is not uniform for all AC voltage measurement ranges. The error is greatest when the meter is placed in its most sensitive voltage range (0-2.5 Volts) and is least when the meter is placed in its least sensitive voltage range (0-250 Volts). Explain why this is.

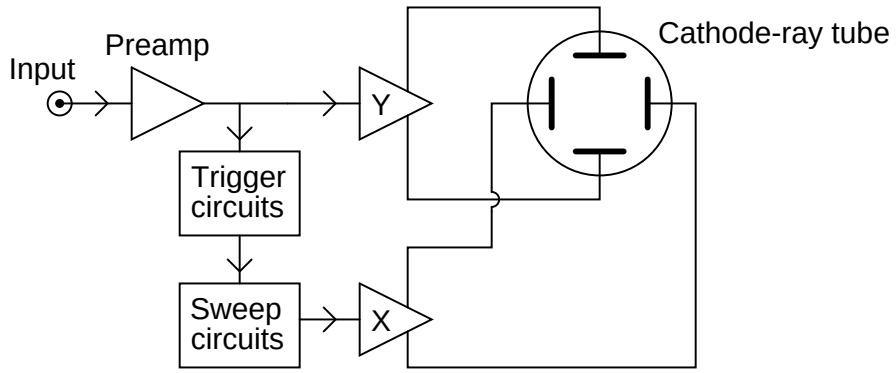
Challenges

- The *Output* function as described filters out DC from a mixed AC-DC signal, so that just the AC portion may be measured. Devise a means for doing the opposite: filtering out AC from a mixed AC-DC signal so that just the DC portion may be measured.

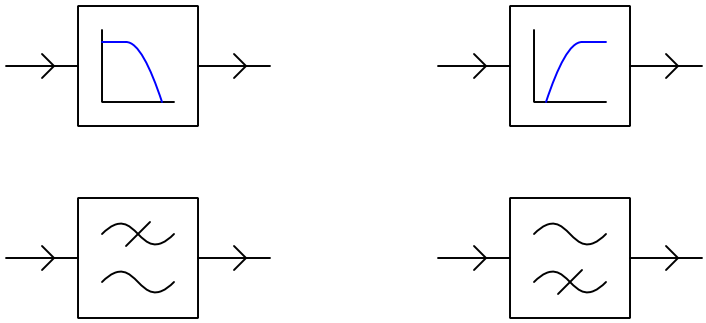
7.1.12 Filter block diagrams

A common way of representing complex electronic systems is the *block diagram*, where specific functional sections of a system are outlined as squares or rectangles, each with a certain purpose and each having input(s) and output(s). For an example, here is a block diagram of an analog ("Cathode Ray") oscilloscope, or *CRO*:

Block diagram of Cathode-Ray Oscilloscope

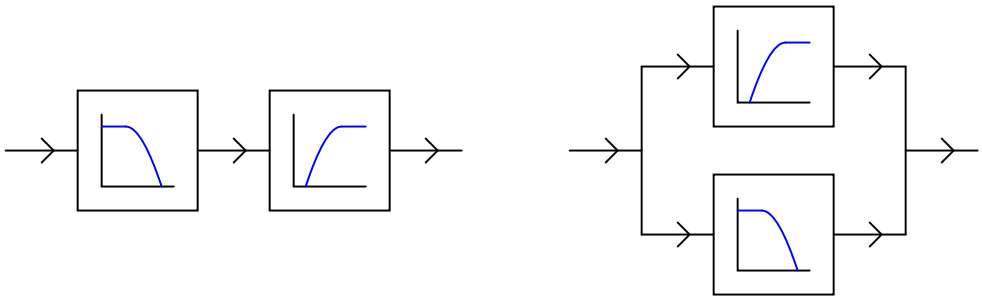


Block diagrams may also be helpful in representing and understanding filter circuits. Consider these symbols, for instance:



Which of these represents a *low-pass* filter, and which represents a *high-pass filter*? Explain your reasoning.

Also, identify the new filter functions created by the compounding of low- and high-pass filter "blocks":



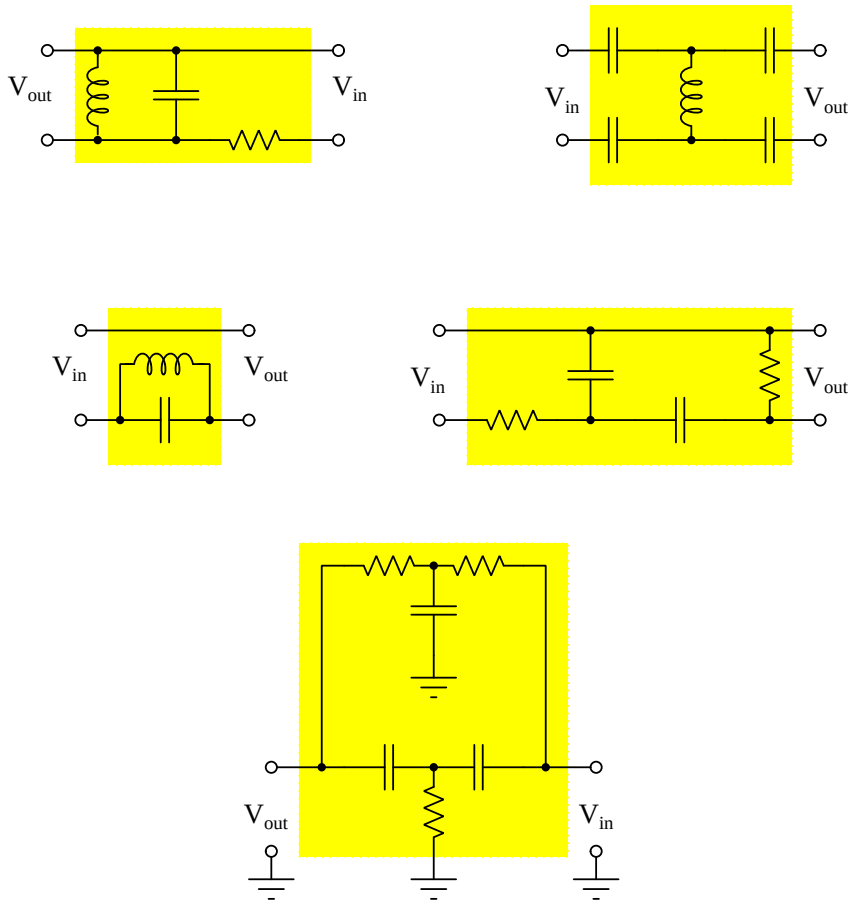
Sketch your own block diagrams to show these new filter functions (each function as a *single* block diagram).

Challenges

- Describe how you could apply the problem-solving technique of a "thought experiment" to the identification of these filtering functions.

7.1.13 Identifying (even more) filter types

Examine the following schematic diagrams of filter networks, and classify the purpose of each as either *low-pass*, *high-pass*, *band-pass*, or *band-stop*:



For each case, explain the reasoning behind your classification of filter type.

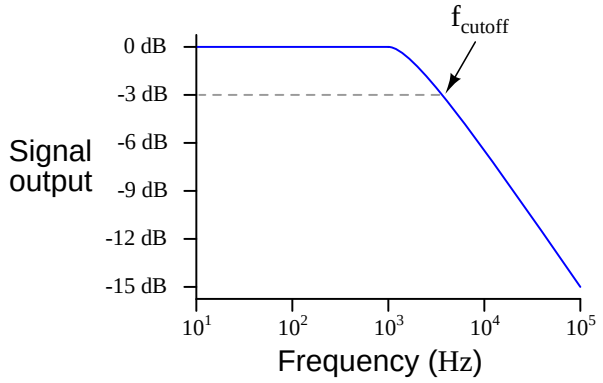
Challenges

- An applicable problem-solving strategy for a qualitative problem such as this is to convert it into a *quantitative* problem. Describe how you could apply this strategy here, and explain how that strategy could help you classify each filter type.
- After identifying each filter classification, propose a modification to the network that would alter it to be another type of filter.
- After identifying each filter classification, identify a change you could make to its component value(s) to *increase* its characteristic frequency.

- For any band-type filter shown here, identify a change you could make to its component value(s) to *increase* its quality factor without altering its center frequency.
- Describe a procedure by which you could empirically determine the classification of a filter circuit, assuming you had no access to its schematic diagram nor to its internal construction.
- Describe a procedure by which you could empirically determine the cutoff frequency(ies) of a filter circuit, assuming you had no access to its schematic diagram nor to its internal construction.

7.1.14 Roll-off

Real filters never exhibit perfect "square-edge" Bode plot responses. A typical low-pass filter circuit, for example, might have a frequency response that looks like this:



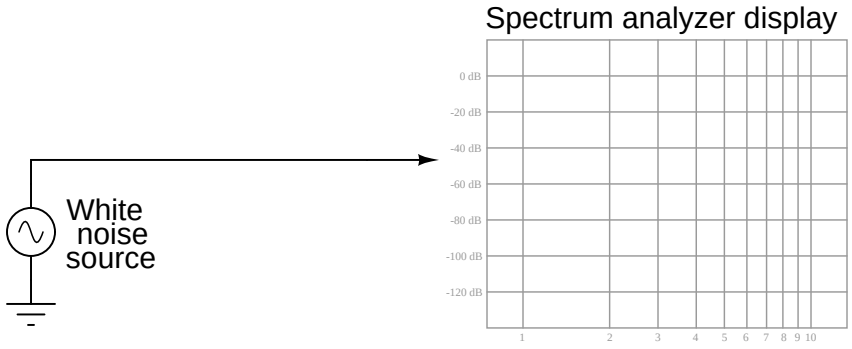
What does the term *roll-off* refer to, in the context of filter circuits and Bode plots? Why would this parameter be important to a technician or engineer?

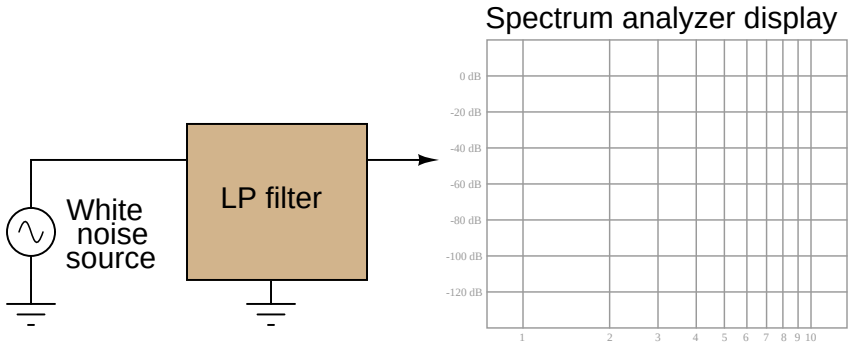
Challenges

- Describe a scenario where a filter circuit needs to have a steep roll-off characteristic.

7.1.15 White noise

A *white noise* source is a special type of AC signal voltage source which outputs a broad band of frequencies ("noise") with a constant amplitude across its rated range. Determine what the display of a spectrum analyzer would show if directly connected to a white noise source, and also if connected to a low-pass filter fed by a white noise source:





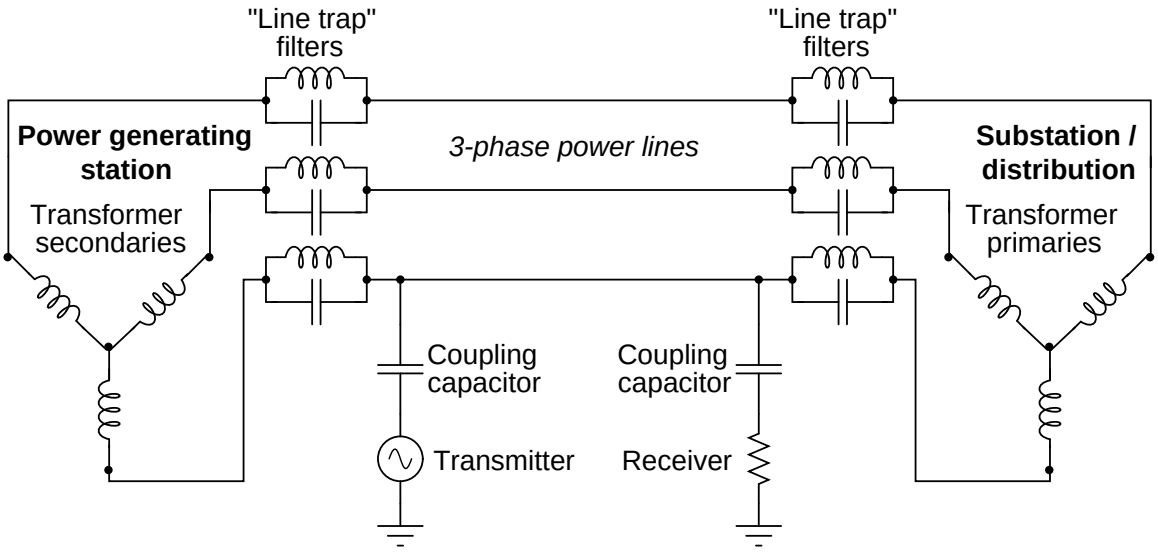
Challenges

- How does a spectrum analyzer differ from an oscilloscope?

7.1.16 Power line carrier communications

An interesting technology dating back at least as far as the 1940's, but which is still of interest today is *power line carrier*: the ability to communicate information as well as electrical power over power line conductors. Hard-wired electronic data communication consists of high-frequency, low voltage AC signals, while electrical power is low-frequency, high-voltage AC. For rather obvious reasons, it is important to be able to separate these two types of AC voltage quantities from entering the wrong equipment (especially the high-voltage AC power from reaching sensitive electronic communications circuitry).

Here is a simplified diagram of a power-line carrier system:



The communications transmitter is shown in simplified form as an AC voltage source, while the receiver is shown as a resistor. Though each of these components is much more complex than what is suggested by these symbols, the purpose here is to show the transmitter as a *source* of high-frequency AC, and the receiver as a *load* of high-frequency AC.

Trace the complete circuit for the high-frequency AC signal generated by the "Transmitter" in the diagram. How many power line conductors are being used in this communications circuit? Explain how the combination of "line trap" LC networks and "coupling" capacitors ensure the communications equipment never becomes exposed to high-voltage electrical power carried by the power lines, and visa-versa.

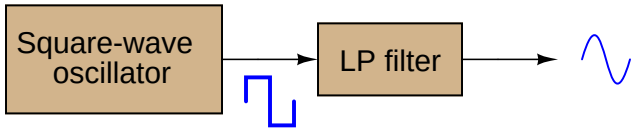
Challenges

- Trace the path of line-frequency (50 Hz or 60 Hz) load current in this system, identifying which component of the line trap filters (L or C) is more important to the passage of power

to the load. Remember that the line trap filters are tuned to resonate at the frequency of the communication signal (50-150 kHz is typical).

7.1.17 Square wave to sine wave

A clever way to produce sine waves is to pass the output of a square-wave oscillator through a low-pass filter circuit:



Explain how this works.

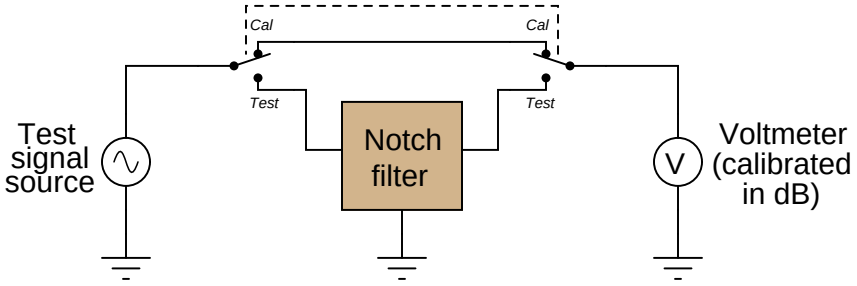
Then, explain how a *band-pass filter* might be used in place of the low-pass filter to do much the same.

Challenges

- Does this LP filter have strict roll-off requirements? Why or why not?
- Would this BP filter have strict quality factor requirements? Why or why not?

7.1.18 Simple harmonic analyzer

A crude measurement circuit for harmonic content of a signal uses a notch filter tuned to the fundamental frequency of the signal being measured. Examine the following circuit and then explain how you think it would work:

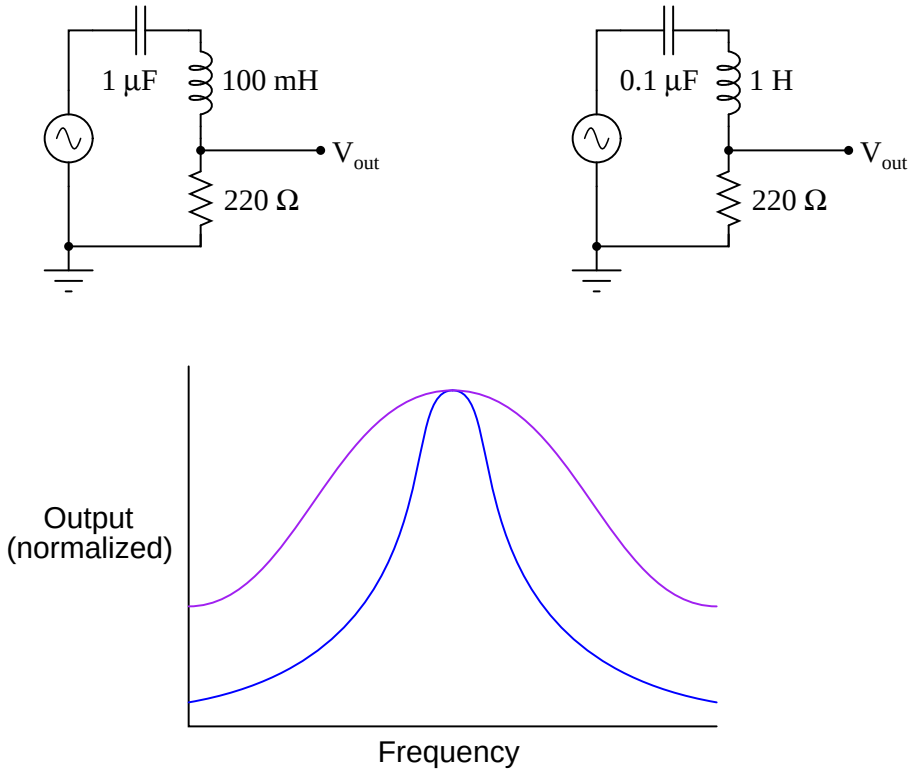


Challenges

- Propose a circuit design to fulfill this notch filter function.

7.1.19 Two resonant circuits of identical frequency

Shown here are two frequency response plots (known as *Bode plots*) for a pair of series resonant circuits having the same resonant frequency. The "output" is voltage measured across the resistor of each circuit:



Determine which plot is associated with which circuit, and explain your answer.

Challenges

- What kind of instrument(s) would you use to plot the response of a real resonant circuit in a lab environment? Would an oscilloscope be helpful with this task? Why or why not?

7.2 Quantitative reasoning

These questions are designed to stimulate your computational thinking. In a Socratic discussion with your instructor, the goal is for these questions to reveal your mathematical approach(es) to problem-solving so that good technique and sound reasoning may be reinforced. Your instructor may also pose additional questions based on those assigned, in order to observe your problem-solving firsthand.

Mental arithmetic and estimations are strongly encouraged for all calculations, because without these abilities you will be unable to readily detect errors caused by calculator misuse (e.g. keystroke errors).

You will note a conspicuous lack of answers given for these quantitative questions. Unlike standard textbooks where answers to every other question are given somewhere toward the back of the book, here in these learning modules students must rely on other means to check their work. My advice is to use circuit simulation software such as SPICE to check the correctness of quantitative answers. Refer to those learning modules within this collection focusing on SPICE to see worked examples which you may use directly as practice problems for your own study, and/or as templates you may modify to run your own analyses and generate your own practice problems.

Completely worked example problems found in the Tutorial may also serve as "test cases[6]" for gaining proficiency in the use of circuit simulation software, and then once that proficiency is gained you will never need to rely[7] on an answer key!

[6] In other words, set up the circuit simulation software to analyze the same circuit examples found in the Tutorial. If the simulated results match the answers shown in the Tutorial, it confirms the simulation has properly run. If the simulated results disagree with the Tutorial's answers, something has been set up incorrectly in the simulation software. Using every Tutorial as practice in this way will quickly develop proficiency in the use of circuit simulation software.

[7] This approach is perfectly in keeping with the instructional philosophy of these learning modules: *teaching students to be self-sufficient thinkers*. Answer keys can be useful, but it is even more useful to your long-term success to have a set of tools on hand for checking your own work, because once you have left school and are on your own, there will no longer be "answer keys" available for the problems you will have to solve.

7.2.1 Miscellaneous physical constants

Note: constants shown in **bold** type are *exact*, not approximations. Values inside of parentheses show one standard deviation (σ) of uncertainty in the final digits: for example, the magnetic permeability of free space value given as $1.25663706212(19) \times 10^{-6}$ H/m represents a center value (i.e. the location parameter) of $1.25663706212 \times 10^{-6}$ Henrys per meter with one standard deviation of uncertainty equal to $0.0000000000019 \times 10^{-6}$ Henrys per meter.

Avogadro's number (N_A) = **6.02214076** $\times 10^{23}$ **per mole** (mol^{-1})

Boltzmann's constant (k) = **1.380649** $\times 10^{-23}$ **Joules per Kelvin** (J/K)

Electronic charge (e) = **1.602176634** $\times 10^{-19}$ **Coulomb** (C)

Faraday constant (F) = **96,485.33212...** $\times 10^{4}$ **Coulombs per mole** (C/mol)

Magnetic permeability of free space (μ_0) = $1.25663706212(19) \times 10^{-6}$ Henrys per meter (H/m)

Electric permittivity of free space (ϵ_0) = $8.8541878128(13) \times 10^{-12}$ Farads per meter (F/m)

Characteristic impedance of free space (Z_0) = 376.730313668(57) Ohms (Ω)

Gravitational constant (G) = $6.67430(15) \times 10^{-11}$ cubic meters per kilogram-seconds squared ($m^3/kg\text{-}s^2$)

Molar gas constant (R) = **8.314462618... Joules per mole-Kelvin** (J/mol-K) = 0.08205746(14) liters-atmospheres per mole-Kelvin

Planck constant (h) = **6.62607015** $\times 10^{-34}$ **joule-seconds** (J-s)

Stefan-Boltzmann constant (σ) = **5.670374419...** $\times 10^{-8}$ **Watts per square meter-Kelvin**^{4} ($W/m^2 \cdot K^4$)

Speed of light in a vacuum (c) = **299,792,458 meters per second** (m/s) = 186282.4 miles per second (mi/s)

Note: All constants taken from NIST data "Fundamental Physical Constants – Complete Listing", from `http://physics.nist.gov/constants`, National Institute of Standards and Technology (NIST), 2018 CODATA Adjustment.

7.2.2 Introduction to spreadsheets

A powerful computational tool you are encouraged to use in your work is a *spreadsheet*. Available on most personal computers (e.g. Microsoft Excel), *spreadsheet* software performs numerical calculations based on number values and formulae entered into cells of a grid. This grid is typically arranged as lettered columns and numbered rows, with each cell of the grid identified by its column/row coordinates (e.g. cell `B3`, cell `A8`). Each cell may contain a string of text, a number value, or a mathematical formula. The spreadsheet automatically updates the results of all mathematical formulae whenever the entered number values are changed. This means it is possible to set up a spreadsheet to perform a series of calculations on entered data, and those calculations will be re-done by the computer any time the data points are edited in any way.

For example, the following spreadsheet calculates average speed based on entered values of distance traveled and time elapsed:

	A	B	C	D
1	Distance traveled	46.9	Kilometers	
2	Time elapsed	1.18	Hours	
3	Average speed	= B1 / B2	km/h	
4				
5				

Text labels contained in cells `A1` through `A3` and cells `C1` through `C3` exist solely for readability and are not involved in any calculations. Cell `B1` contains a sample distance value while cell `B2` contains a sample time value. The formula for computing speed is contained in cell `B3`. Note how this formula begins with an "equals" symbol (`=`), references the values for distance and speed by lettered column and numbered row coordinates (`B1` and `B2`), and uses a forward slash symbol for division (`/`). The coordinates `B1` and `B2` function as *variables*[8] would in an algebraic formula.

When this spreadsheet is executed, the numerical value `39.74576` will appear in cell `B3` rather than the formula `= B1 / B2`, because 39.74576 is the computed speed value given 46.9 kilometers traveled over a period of 1.18 hours. If a different numerical value for distance is entered into cell `B1` or a different value for time is entered into cell `B2`, cell `B3`'s value will automatically update. All you need to do is set up the given values and any formulae into the spreadsheet, and the computer will do all the calculations for you.

Cell `B3` may be referenced by other formulae in the spreadsheet if desired, since it is a variable just like the given values contained in `B1` and `B2`. This means it is possible to set up an entire chain of calculations, one dependent on the result of another, in order to arrive at a final value. The arrangement of the given data and formulae need not follow any pattern on the grid, which means you may place them anywhere.

[8] Spreadsheets may also provide means to attach text labels to cells for use as variable names (Microsoft Excel simply calls these labels "names"), but for simple spreadsheets such as those shown here it's usually easier just to use the standard coordinate naming for each cell.

Common[9] arithmetic operations available for your use in a spreadsheet include the following:

- Addition (`+`)
- Subtraction (`-`)
- Multiplication (`*`)
- Division (`/`)
- Powers (`^`)
- Square roots (`sqrt()`)
- Logarithms (`ln()` , `log10()`)

Parentheses may be used to ensure[10] proper order of operations within a complex formula. Consider this example of a spreadsheet implementing the *quadratic formula*, used to solve for roots of a polynomial expression in the form of $ax^2 + bx + c$:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

	A	B
1	x_1	= (-B4 + sqrt((B4^2) - (4*B3*B5))) / (2*B3)
2	x_2	= (-B4 - sqrt((B4^2) - (4*B3*B5))) / (2*B3)
3	a =	9
4	b =	5
5	c =	-2

This example is configured to compute roots[11] of the polynomial $9x^2 + 5x - 2$ because the values of 9, 5, and -2 have been inserted into cells `B3`, `B4`, and `B5`, respectively. Once this spreadsheet has been built, though, it may be used to calculate the roots of *any* second-degree polynomial expression simply by entering the new a, b, and c coefficients into cells `B3` through `B5`. The numerical values appearing in cells `B1` and `B2` will be automatically updated by the computer immediately following any changes made to the coefficients.

[9] Modern spreadsheet software offers a bewildering array of mathematical functions you may use in your computations. I recommend you consult the documentation for your particular spreadsheet for information on operations other than those listed here.

[10] Spreadsheet programs, like text-based programming languages, are designed to follow standard order of operations by default. However, my personal preference is to use parentheses even where strictly unnecessary just to make it clear to any other person viewing the formula what the intended order of operations is.

[11] Reviewing some algebra here, a *root* is a value for x that yields an overall value of zero for the polynomial. For this polynomial ($9x^2 + 5x - 2$) the two roots happen to be $x = 0.269381$ and $x = -0.82494$, with these values displayed in cells `B1` and `B2`, respectively upon execution of the spreadsheet.

Alternatively, one could break up the long quadratic formula into smaller pieces like this:

$$y = \sqrt{b^2 - 4ac} \qquad z = 2a$$

$$x = \frac{-b \pm y}{z}$$

	A	B	C
1	x_1	= (-B4 + C1) / C2	= sqrt((B4^2) - (4*B3*B5))
2	x_2	= (-B4 - C1) / C2	= 2*B3
3	a =	9	
4	b =	5	
5	c =	-2	

Note how the square-root term (y) is calculated in cell C1, and the denominator term (z) in cell C2. This makes the two final formulae (in cells B1 and B2) simpler to interpret. The positioning of all these cells on the grid is completely arbitrary[12] – all that matters is that they properly reference each other in the formulae.

Spreadsheets are particularly useful for situations where the same set of calculations representing a circuit or other system must be repeated for different initial conditions. The power of a spreadsheet is that it automates what would otherwise be a tedious set of calculations. One specific application of this is to simulate the effects of various components within a circuit failing with abnormal values (e.g. a shorted resistor simulated by making its value nearly zero; an open resistor simulated by making its value extremely large). Another application is analyzing the behavior of a circuit design given new components that are out of specification, and/or aging components experiencing drift over time.

[12] My personal preference is to locate all the "given" data in the upper-left cells of the spreadsheet grid (each data point flanked by a sensible name in the cell to the left and units of measurement in the cell to the right as illustrated in the first distance/time spreadsheet example), sometimes coloring them in order to clearly distinguish which cells contain entered data versus which cells contain computed results from formulae. I like to place all formulae in cells below the given data, and try to arrange them in logical order so that anyone examining my spreadsheet will be able to figure out *how* I constructed a solution. This is a general principle I believe all computer programmers should follow: *document and arrange your code to make it easy for other people to learn from it.*

7.2.3 Practice: complex number calculations

These complex-number arithmetic problems are presented to you, complete with answers (shown in **bold**), for the purpose of practice, since nearly all AC circuit calculations will need to be performed using complex numbers. Use these practice calculations to check your ability either to perform these calculations "by hand" (using trigonometric functions) or your ability to use your calculator's complex-number functionality.

Note: electronic hand calculators and computer-based calculation programs use the proper mathematical notation i to represent imaginary numbers rather than j. The letter j is used in electrical engineering work in order to avoid confusion with i being misinterpreted as "current". Also note that calculators and software programs usually default to *radians* for angle measurement rather than *degrees*, and will have to be configured (or converted) for degrees in order to handle the polar-form complex quantities shown here. Check the "mode" options of your hand calculator to ensure angles are in the correct unit and also that it will display in either rectangular or polar.

Addition and subtraction:

$(5 + j6) + (2 - j1) = \mathbf{7 + j5}$
$(10 - j8) + (4 - j3) = \mathbf{14 - j11}$
$(-3 + j0) + (9 - j12) = \mathbf{6 - j12}$

$(3 + j5) - (0 - j9) = \mathbf{3 + j14}$
$(25 - j84) - (4 - j3) = \mathbf{21 - j81}$
$(-1500 + j40) + (299 - j128) = \mathbf{-1201 - j88}$

$(25\angle 15^o) + (10\angle 74^o) = \mathbf{31.35\angle 30.87^o}$
$(1000\angle 43^o) + (1200\angle -20^o) = \mathbf{1878.7\angle 8.311^o}$
$(522\angle 71^o) - (85\angle 30^o) = \mathbf{461.23\angle 77.94^o}$

Multiplication and division:

$(25\angle 15^o) \times (12\angle 10^o) = \mathbf{300\angle 25^o}$
$(1\angle 25^o) \times (500\angle -30^o) = \mathbf{500\angle -5^o}$
$(522\angle 71^o) \times (33\angle 9^o) = \mathbf{17226\angle 80^o}$

$\frac{10\angle -80^o}{1\angle 0^o} = \mathbf{10\angle -80^o}$

$\frac{25\angle 120^o}{3.5\angle -55^o} = \mathbf{7.142\angle 175^o}$

$\frac{-66\angle 67^o}{8\angle -42^o} = \mathbf{8.25\angle -71^o}$

$(3 + j5) \times (2 - j1) = \mathbf{11 + j7}$
$(10 - j8) \times (4 - j3) = \mathbf{16 - j62}$

$\frac{(3+j4)}{(12-j2)} = \mathbf{0.1892 + j0.3649}$

Reciprocation:

$\frac{1}{(15\angle 60^o)} = \mathbf{0.0667\angle -60^o}$

$\frac{1}{(750\angle -38^o)} = \mathbf{0.00133\angle 38^o}$

$\frac{1}{(10+j3)} = \mathbf{0.0917 - j0.0275}$

$\frac{1}{\frac{1}{15\angle 45^o}+\frac{1}{92\angle -25^o}} = \mathbf{14.06\angle 36.74^o}$

$\frac{1}{\frac{1}{1200\angle 73^o}+\frac{1}{574\angle 21^o}} = \mathbf{425.7\angle 37.23^o}$

$\frac{1}{\frac{1}{23k\angle -67^o}+\frac{1}{10k\angle -81^o}} = \mathbf{7.013k\angle -76.77^o}$

$\frac{1}{\frac{1}{110\angle -34^o}+\frac{1}{80\angle 19^o}+\frac{1}{70\angle 10}} = \mathbf{29.89\angle 2.513^o}$

$\frac{1}{\frac{1}{89k\angle -5^o}+\frac{1}{15k\angle 33^o}+\frac{1}{9.35k\angle 45}} = \mathbf{5.531k\angle 37.86^o}$

$\frac{1}{\frac{1}{512\angle 34^o}+\frac{1}{1k\angle -25^o}+\frac{1}{942\angle -20}+\frac{1}{2.2k\angle 44^o}} = \mathbf{256.4\angle 9.181^o}$

Sign reversal:

$-(45\angle 70^o) = \mathbf{45\angle -110^o}$

$-(90\angle -20^o) = \mathbf{90\angle 160^o}$

$-(5 + j8) = \mathbf{-5 - j8}$

$-(-3 + j9) = \mathbf{3 - j9}$

$-(10 - j15) = \mathbf{-10 + j15}$

Practical suggestions for using your calculator to perform these operations:

- Surround each complex-number quantity with parentheses when setting up arithmetic operations; e.g., `(3 + j5) * (4 - j2)` instead of `3 + j5 * 4 - j2`. This habit will guarantee your calculator executes the desired order of operations rather than assert its own. For instance, in the example given here the calculator may choose to multiply $j5$ by 4 and then add on 3 and $-j2$ since multiplication typically precedes addition, if the two complex numbers are not encapsulated in their own sets of parentheses.

- Store all calculated results in memory and then recall from memory when re-using those values, rather than re-entering previously-calculated values by hand or sampling previously-calculated values from the multi-line display. Manually re-entering values invites rounding errors and keystroke errors in all cases, and I've found certain calculators (I'm looking at you, *TI*!) fail to properly enter complex-number values when sampled from their multi-line displays. Getting in the habit of using your calculator's memory locations is an all-around good habit that will serve you very well!

Some Texas Instruments brand calculators such as the TI-84 offer an exponential key and imaginary (i) key which allows you to enter numbers in complex exponential form (i.e. $e^{i\theta}$). With the TI-84, for example, the complex number $10 - j8$ may be entered in either of the two following forms:

`(10 - i8)` or `(10 - 8i)`

The result may be displayed in either rectangular or polar forms according to the complex-number display *mode* the TI-84 calculator has been set to. In rectangular mode the displayed result for $10 - j8$ will be `10 - 8i`, whereas in polar mode the displayed result will be `12.806` $\mathtt{e}^{-38.66i}$. Note how the TI-84 uses exponential notation for polar display, where the angle (-38.66 degrees, in this example) is an imaginary power of e.

If you wish to enter a complex number in polar form on a TI-84, you must unfortunately express the angle in units of *radians* (even though the calculator is able to display the result in *degrees*). For example, to enter the number $25\angle 15^o$ into a TI-84 calculator, you must type:

`25` $\mathtt{e}^{i15\pi/180}$

The fraction $\pi/180$ is the conversion factor from degrees to radians, since there are 2π radians to a full circle, or π radians to every 180 degrees. Thus, writing $15\pi/180$ multiplies the desired angle (15 degrees) by the conversion factor $\pi/180$ to yield a power in radians. The obligatory i simply makes this power an imaginary quantity, which is mathematically necessary with exponential notation for describing a complex number. It should be noted that the order of entry for the power matters little. $i15\pi/180$ works just as well as $15i\pi/180$ or $15\pi/180i$.

A time-saving step some students find useful is to save the imaginary quantity $i\pi/180$ to a memory location in the TI-84 such as Z. That way, they can recall that imaginary factor from memory instead of typing the whole thing by hand every time they wish to enter a polar-form complex number. Supposing the memory location Z contains $i\pi/180$, entering the number $25\angle 15^o$ becomes as simple as:

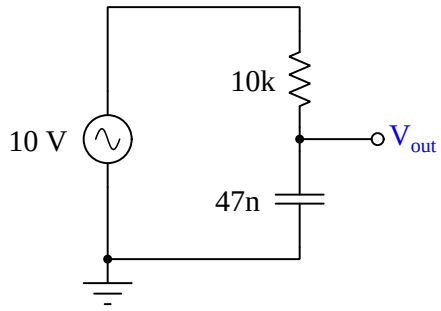
25 e^{15Z} or 25 e^{Z15}

It should be understood that *any* memory location in your calculator is suitable for storing $i\pi/180$, not just Z. The TI-84 calculator even provides a Θ memory location (<Alpha>-<3>) that you may use and find easy to remember because of its common association with angles. It should also be understood that this imaginary quantity is not the same as i or j, which the calculator already provides a dedicated function for. The imaginary quantity we're storing in memory for the purpose of entering polar-notation angles contains not only i but also the $\pi/180$ conversion factor necessary for translating your *degree* entry into *radians*.

Some calculators provide easier means of entering and displaying complex numbers in polar form. Both the Texas Instruments TI-36X Pro and TI-89 offer an angle symbol ($\angle$) for this purpose, the TI-36X Pro being a much less expensive and less complex calculator than the TI-89. Entering the number $25 \angle 15^o$ into one of these calculators is as easy as typing 25 $\angle$ 15, and likewise the result will be displayed in this same form when the calculator is set to "polar" complex mode.

7.2.4 Frequency response of an RC network

Calculate and plot V_{out} for the following source frequencies:



- $f = 0$ Hz ; $V_{out} =$
- $f = 100$ Hz ; $V_{out} =$
- $f = 200$ Hz ; $V_{out} =$
- $f = 300$ Hz ; $V_{out} =$
- $f = 400$ Hz ; $V_{out} =$
- $f = 500$ Hz ; $V_{out} =$
- $f = \infty$ Hz ; $V_{out} =$

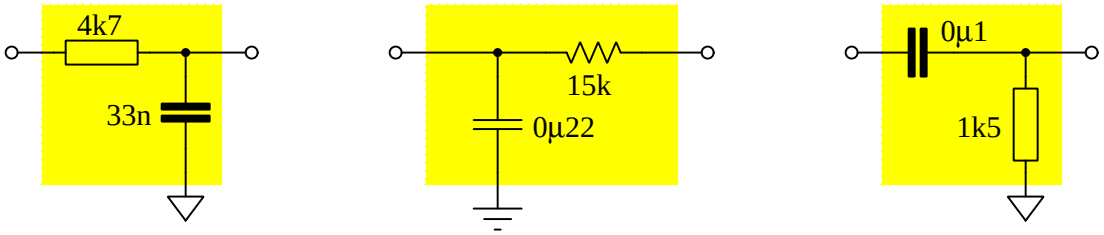
If we consider the 10 Volt AC source to be the "input" signal for this network, would you say that it passes low-frequency signals better than high-frequency signals, vice-versa, or passes all frequencies equally well?

Challenges

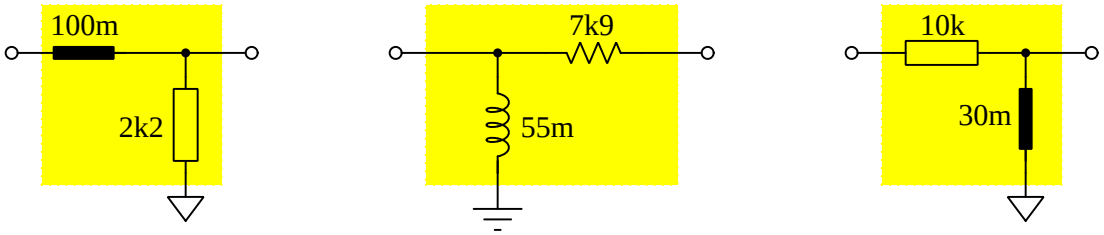
- Properly re-draw the schematic without using a ground symbol.

7.2.5 Filter type and cutoff identifications

Identify the type of filter for each RC network shown below, calculate the cutoff frequency of each, and distinguish the input terminals from the output terminals:



Identify the type of filter for each LR network shown below, calculate the cutoff frequency of each, and distinguish the input terminals from the output terminals:



Challenges

- What do the different symbol styles represent?
- Why don't any the the specified component values appear with decimal points?
- What happens to f_c as C is made larger, and why?
- What happens to f_c as L is made larger, and why?
- What happens to f_c as R is made larger, and why?

7.2.6 Designing simple RC low-pass and high-pass filters

For each of the following example applications, design a suitable low-pass or high-pass filter circuit, each circuit using nothing but a single resistor and a single capacitor:

Example #1

Design a simple low-pass filter using a 0.01 μF capacitor and a resistor of your own choosing, to have a cutoff frequency of 3.5 kHz.

Example #2

Design a simple high-pass filter using a 10 kΩ resistor and a capacitor of your own choosing, to have a cutoff frequency of 1.1 kHz.

Example #3

Design a simple high-pass filter using a 3.3 nF capacitor and a resistor of your own choosing, to have a cutoff frequency of 9.4 kHz.

Example #4

Design a simple low-pass filter using a 4.7 kΩ resistor and a capacitor of your own choosing, to have a cutoff frequency of 800 Hz.

Challenges

- What, exactly, does "cutoff frequency" mean for a low-pass or a high-pass filter circuit?
- Design a *switchable* filter circuit where a toggle switch's position determines whether the circuit will be a low-pass or a high-pass filter.

7.2.7 Designing filters using IEC standard component values

Sketch schematic diagrams and select components to build a low-pass filter and a high-pass filter using only IEC-60063 (E12) component values, making sure the cutoff frequency of your filter is within ± 5% of the requested value in each case. Be sure your filter manifests an input impedance somewhere between 5 kΩ and 50 kΩ at cutoff:

- Low-pass filter with f_{-3dB} of 400 kHz
- High-pass filter with f_{-3dB} of 25 kHz

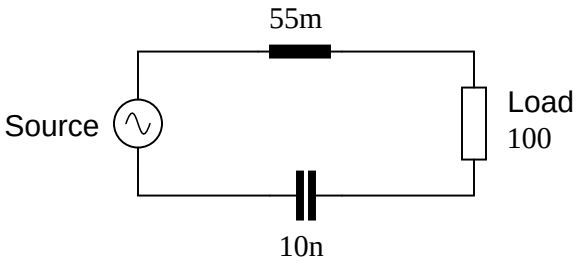
Identify a practical strategy for selecting component values based on the limited E12-series offerings.

Challenges

- What, exactly, does "cutoff frequency" mean for a low-pass or a high-pass filter circuit?
- Design a *switchable* filter circuit where a toggle switch's position determines whether the circuit will be a low-pass or a high-pass filter.

7.2.8 Resonant filter type and cutoff

Calculate the resonant frequency, bandwidth, and half-power points of the following filter circuit:



Challenges

- How would a decrease in the Q ("quality factor") of the circuit affect the bandwidth, or would it at all?
- Identify how to vary the Q of this filter circuit without affecting its resonant frequency.

7.2.9 Deriving a formula for Q

The Q factor of a series inductive circuit is given by the following equation:

$$Q = \frac{X_L}{R_{series}}$$

Likewise, we know that inductive reactance may be found by the following equation:

$$X_L = 2\pi f L$$

We also know that the resonant frequency of a series LC circuit is given by this equation:

$$f_r = \frac{1}{2\pi\sqrt{LC}}$$

Through algebraic substitution, write an equation that gives the Q factor of a series resonant LC circuit exclusively in terms of L, C, and R, without reference to reactance (X) or frequency (f).

Challenges

- What is the practical significance of Q for a filter circuit?

7.2.10 Using C to analyze a filter network

The following computer program (written in the C language) analyzes a simple reactive filter network at a single frequency:

```
#include <stdio.h>
#include <math.h>

int main (void)
{
  float Vsrc = 100E-3;
  float R = 3.3E3;
  float C = 1E-9;
  float f = 40E3;
  float X, Z, I, Vout;

  X = 1/(2 * M_PI * f * C);
  Z = sqrt(pow(R,2) + pow(X,2));
  I = Vsrc / Z;
  Vout = I * R;

  printf("Output voltage = %f at %f Hz\n", Vout, f);

  return 0;
}
```

Identify the meaning of each line of code pertaining to a filter network, identifying the foundational concept applied in each, as well as the proper unit of measurement for each result.

Next, answer the following questions:

- Sketch a schematic diagram representing the filter network simulated by this program.
- Identify what characteristic this filter exhibits (e.g. low-pass, high-pass, etc.).
- Modify the program to perform a "frequency sweep" from 30 kHz to 100 kHz in 10 kHz increments.
- Modify the program to simulate a different characteristic of filter.

Challenges

- ???

7.3 Diagnostic reasoning

These questions are designed to stimulate your deductive and inductive thinking, where you must apply general principles to specific scenarios (deductive) and also derive conclusions about the failed circuit from specific details (inductive). In a Socratic discussion with your instructor, the goal is for these questions to reinforce your recall and use of general circuit principles and also challenge your ability to integrate multiple symptoms into a sensible explanation of what's wrong in a circuit. Your instructor may also pose additional questions based on those assigned, in order to further challenge and sharpen your diagnostic abilities.

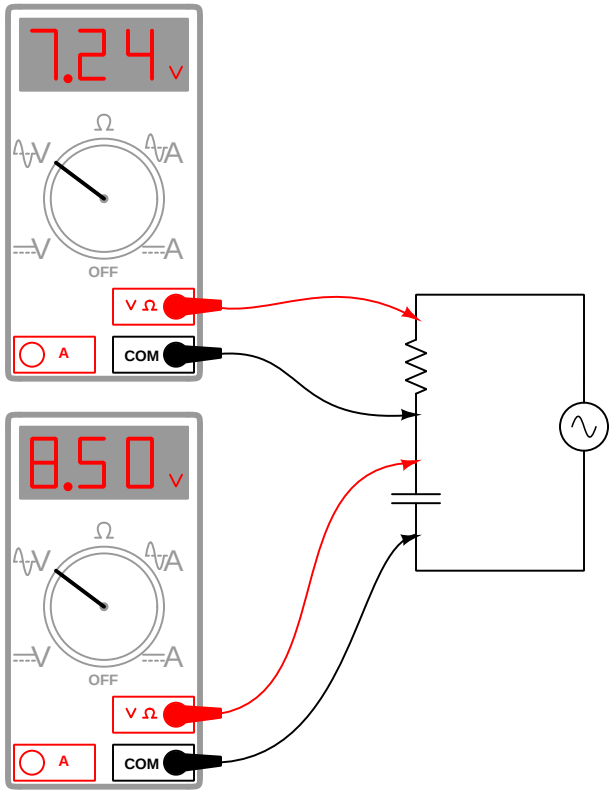
As always, your goal is to fully *explain* your analysis of each problem. Simply obtaining a correct answer is not good enough – you must also demonstrate sound reasoning in order to successfully complete the assignment. Your instructor's responsibility is to probe and challenge your understanding of the relevant principles and analytical processes in order to ensure you have a strong foundation upon which to build further understanding.

You will note a conspicuous lack of answers given for these diagnostic questions. Unlike standard textbooks where answers to every other question are given somewhere toward the back of the book, here in these learning modules students must rely on other means to check their work. The best way by far is to debate the answers with fellow students and also with the instructor during the Socratic dialogue sessions intended to be used with these learning modules. Reasoning through challenging questions with other people is an excellent tool for developing strong reasoning skills.

Another means of checking your diagnostic answers, where applicable, is to use circuit simulation software to explore the effects of faults placed in circuits. For example, if one of these diagnostic questions requires that you predict the effect of an open or a short in a circuit, you may check the validity of your work by simulating that same fault (substituting a very high resistance in place of that component for an open, and substituting a very low resistance for a short) within software and seeing if the results agree.

7.3.1 Incorrect voltage calculation

A student measures voltage drops in an AC circuit using two AC voltmeters and arrives at the following measurements:



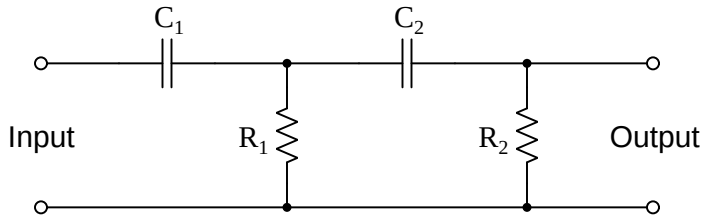
Knowing that voltages add in series circuits, the student sums 7.24 Volts and 8.50 Volts to arrive at 15.74 Volts. However, this result is incorrect. Explain what the student did wrong, and then calculate the proper series-total voltage in this circuit.

Challenges

- Calculate a set of possible values for the capacitor and resistor that would generate these same voltage drops in a real circuit. Hint: you must also decide on a value of frequency for the power source.

7.3.2 Component failures in a second-order filter circuit

Predict how the operation of this second-order passive filter circuit will be affected as a result of the following faults. Consider each fault independently (i.e. one at a time, no coincidental faults):



- Capacitor C_1 fails open:
- Resistor R_1 fails open:
- Resistor R_2 fails open:
- Solder bridge (short) across resistor R_2:

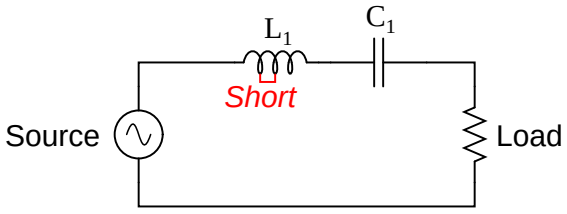
For each of these conditions, explain *why* the resulting effects will occur.

Challenges

- An applicable problem-solving strategy for a qualitative problem such as this is to convert it into a *quantitative* problem. Describe how you could apply this strategy here, and explain how that strategy could help you classify each filter type.

7.3.3 Partially failed inductor

Suppose a few turns of wire within the inductor in this filter circuit suddenly became short-circuited, so that the inductor effectively has fewer turns of wire than it did before:



What type of filter is this (e.g. LP, HP, BP, BS), and what sort of effect would this fault have on the filtering action of this circuit?

Challenges

- What would happen to the Q of this filter circuit as a result of the fault within the inductor?

Appendix A

Problem-Solving Strategies

The ability to solve complex problems is arguably one of the most valuable skills one can possess, and this skill is particularly important in any science-based discipline.

- Study principles, not procedures. Don't be satisfied with merely knowing how to compute solutions – learn *why* those solutions work.
- Identify what it is you need to solve, identify all relevant data, identify all units of measurement, identify any general principles or formulae linking the given information to the solution, and then identify any "missing pieces" to a solution. Annotate all diagrams with this data.
- Sketch a diagram to help visualize the problem. When building a real system, always devise a plan for that system and analyze its function *before* constructing it.
- Follow the units of measurement and meaning of every calculation. If you are ever performing mathematical calculations as part of a problem-solving procedure, and you find yourself unable to apply each and every intermediate result to some aspect of the problem, it means you don't understand what you are doing. Properly done, every mathematical result should have practical meaning for the problem, and not just be an abstract number. You should be able to identify the proper units of measurement for each and every calculated result, and show where that result fits into the problem.
- Perform "thought experiments" to explore the effects of different conditions for theoretical problems. When troubleshooting real systems, perform *diagnostic tests* rather than visually inspecting for faults, the best diagnostic test being the one giving you the most information about the nature and/or location of the fault with the fewest steps.
- Simplify the problem until the solution becomes obvious, and then use that obvious case as a model to follow in solving the more complex version of the problem.
- Check for exceptions to see if your solution is incorrect or incomplete. A good solution will work for *all* known conditions and criteria. A good example of this is the process of testing scientific hypotheses: the task of a scientist is not to find support for a new idea, but rather to *challenge* that new idea to see if it holds up under a battery of tests. The philosophical

principle of *reductio ad absurdum* (i.e. disproving a general idea by finding a specific case where it fails) is useful here.

- Work "backward" from a hypothetical solution to a new set of given conditions.
- Add quantities to problems that are qualitative in nature, because sometimes a little math helps illuminate the scenario.
- Sketch graphs illustrating how variables relate to each other. These may be quantitative (i.e. with realistic number values) or qualitative (i.e. simply showing increases and decreases).
- Treat quantitative problems as qualitative in order to discern the relative magnitudes and/or directions of change of the relevant variables. For example, try determining what happens if a certain variable were to increase or decrease before attempting to precisely calculate quantities: how will each of the dependent variables respond, by increasing, decreasing, or remaining the same as before?
- Consider limiting cases. This works especially well for qualitative problems where you need to determine which direction a variable will change. Take the given condition and magnify that condition to an extreme degree as a way of simplifying the direction of the system's response.
- Check your work. This means regularly testing your conclusions to see if they make sense. This does *not* mean repeating the same steps originally used to obtain the conclusion(s), but rather to use some other means to check validity. Simply repeating procedures often leads to *repeating the same errors* if any were made, which is why alternative paths are better.

Appendix B

Instructional philosophy

"The unexamined circuit is not worth energizing" – Socrates (if he had taught electricity)

These learning modules, although useful for self-study, were designed to be used in a formal learning environment where a subject-matter expert challenges students to digest the content and exercise their critical thinking abilities in the answering of questions and in the construction and testing of working circuits.

The following principles inform the instructional and assessment philosophies embodied in these learning modules:

- The first goal of education is to enhance clear and independent thought, in order that every student reach their fullest potential in a highly complex and inter-dependent world. Robust reasoning is *always* more important than particulars of any subject matter, because its application is universal.
- Literacy is fundamental to independent learning and thought because text continues to be the most efficient way to communicate complex ideas over space and time. Those who cannot read with ease are limited in their ability to acquire knowledge and perspective.
- Articulate communication is fundamental to work that is complex and interdisciplinary.
- Faulty assumptions and poor reasoning are best corrected through challenge, not presentation. The rhetorical technique of *reductio ad absurdum* (disproving an assertion by exposing an absurdity) works well to discipline student's minds, not only to correct the problem at hand but also to learn how to detect and correct future errors.
- Important principles should be repeatedly explored and widely applied throughout a course of study, not only to reinforce their importance and help ensure their mastery, but also to showcase the interconnectedness and utility of knowledge.

These learning modules were expressly designed to be used in an "inverted" teaching environment[1] where students first read the introductory and tutorial chapters on their own, then individually attempt to answer the questions and construct working circuits according to the experiment and project guidelines. The instructor never lectures, but instead meets regularly with each individual student to review their progress, answer questions, identify misconceptions, and challenge the student to new depths of understanding through further questioning. Regular meetings between instructor and student should resemble a Socratic[2] dialogue, where questions serve as scalpels to dissect topics and expose assumptions. The student passes each module only after consistently demonstrating their ability to logically analyze and correctly apply all major concepts in each question or project/experiment. The instructor must be vigilant in probing each student's understanding to ensure they are truly *reasoning* and not just *memorizing*. This is why "Challenge" points appear throughout, as prompts for students to think deeper about topics and as starting points for instructor queries. Sometimes these challenge points require additional knowledge that hasn't been covered in the series to answer in full. This is okay, as the major purpose of the Challenges is to stimulate analysis and synthesis on the part of each student.

The instructor must possess enough mastery of the subject matter and awareness of students' reasoning to generate their own follow-up questions to practically any student response. Even completely correct answers given by the student should be challenged by the instructor for the purpose of having students practice articulating their thoughts and defending their reasoning. Conceptual errors committed by the student should be exposed and corrected not by direct instruction, but rather by reducing the errors to an absurdity[3] through well-chosen questions and thought experiments posed by the instructor. Becoming proficient at this style of instruction requires time and dedication, but the positive effects on critical thinking for both student and instructor are spectacular.

An inspection of these learning modules reveals certain unique characteristics. One of these is a bias toward thorough explanations in the tutorial chapters. Without a live instructor to explain concepts and applications to students, the text itself must fulfill this role. This philosophy results in lengthier explanations than what you might typically find in a textbook, each step of the reasoning process fully explained, including footnotes addressing common questions and concerns students raise while learning these concepts. Each tutorial seeks to not only explain each major concept in sufficient detail, but also to explain the logic of each concept and how each may be developed

[1] In a traditional teaching environment, students first encounter new information via *lecture* from an expert, and then independently apply that information via *homework*. In an "inverted" course of study, students first encounter new information via *homework*, and then independently apply that information under the scrutiny of an expert. The expert's role in lecture is to simply *explain*, but the expert's role in an inverted session is to *challenge*, *critique*, and if necessary *explain* where gaps in understanding still exist.

[2] Socrates is a figure in ancient Greek philosophy famous for his unflinching style of questioning. Although he authored no texts, he appears as a character in Plato's many writings. The essence of Socratic philosophy is to leave no question unexamined and no point of view unchallenged. While purists may argue a topic such as electric circuits is too narrow for a true Socratic-style dialogue, I would argue that the essential thought processes involved with scientific reasoning on *any* topic are not far removed from the Socratic ideal, and that students of electricity and electronics would do very well to challenge assumptions, pose thought experiments, identify fallacies, and otherwise employ the arsenal of critical thinking skills modeled by Socrates.

[3] This rhetorical technique is known by the Latin phrase *reductio ad absurdum*. The concept is to expose errors by counter-example, since only one solid counter-example is necessary to disprove a universal claim. As an example of this, consider the common misconception among beginning students of electricity that voltage cannot exist without current. One way to apply *reductio ad absurdum* to this statement is to ask how much current passes through a fully-charged battery connected to nothing (i.e. a clear example of voltage existing without current).

from "first principles". Again, this reflects the goal of developing clear and independent thought in students' minds, by showing how clear and logical thought was used to forge each concept. Students benefit from witnessing a model of clear thinking in action, and these tutorials strive to be just that.

Another characteristic of these learning modules is a lack of step-by-step instructions in the Project and Experiment chapters. Unlike many modern workbooks and laboratory guides where step-by-step instructions are prescribed for each experiment, these modules take the approach that students must learn to closely read the tutorials and apply their own reasoning to identify the appropriate experimental steps. Sometimes these steps are plainly declared in the text, just not as a set of enumerated points. At other times certain steps are implied, an example being assumed competence in test equipment use where the student should not need to be told *again* how to use their multimeter because that was thoroughly explained in previous lessons. In some circumstances no steps are given at all, leaving the entire procedure up to the student.

This lack of prescription is not a flaw, but rather a feature. Close reading and clear thinking are foundational principles of this learning series, and in keeping with this philosophy all activities are designed to *require* those behaviors. Some students may find the lack of prescription frustrating, because it demands more from them than what their previous educational experiences required. This frustration should be interpreted as an unfamiliarity with autonomous thinking, a problem which must be corrected if the student is ever to become a self-directed learner and effective problem-solver. Ultimately, the need for students to read closely and think clearly is more important both in the near-term and far-term than any specific facet of the subject matter at hand. If a student takes longer than expected to complete a module because they are forced to outline, digest, and reason on their own, so be it. The future gains enjoyed by developing this mental discipline will be well worth the additional effort and delay.

Another feature of these learning modules is that they do not treat topics in isolation. Rather, important concepts are introduced early in the series, and appear repeatedly as stepping-stones toward other concepts in subsequent modules. This helps to avoid the "compartmentalization" of knowledge, demonstrating the inter-connectedness of concepts and simultaneously reinforcing them. Each module is fairly complete in itself, reserving the beginning of its tutorial to a review of foundational concepts.

This methodology of assigning text-based modules to students for digestion and then using Socratic dialogue to assess progress and hone students' thinking was developed over a period of several years by the author with his Electronics and Instrumentation students at the two-year college level. While decidedly unconventional and sometimes even unsettling for students accustomed to a more passive lecture environment, this instructional philosophy has proven its ability to convey conceptual mastery, foster careful analysis, and enhance employability so much better than lecture that the author refuses to ever teach by lecture again.

Problems which often go undiagnosed in a lecture environment are laid bare in this "inverted" format where students must articulate and logically defend their reasoning. This, too, may be unsettling for students accustomed to lecture sessions where the instructor cannot tell for sure who comprehends and who does not, and this vulnerability necessitates sensitivity on the part of the "inverted" session instructor in order that students never feel discouraged by having their errors exposed. *Everyone* makes mistakes from time to time, and learning is a lifelong process! Part of the instructor's job is to build a culture of learning among the students where errors are not seen as shameful, but rather as opportunities for progress.

To this end, instructors managing courses based on these modules should adhere to the following principles:

- Student questions are always welcome and demand thorough, honest answers. The only type of question an instructor should refuse to answer is one the student should be able to easily answer on their own. Remember, *the fundamental goal of education is for each student to learn to think clearly and independently.* This requires hard work on the part of the student, which no instructor should ever circumvent. Anything done to bypass the student's responsibility to do that hard work ultimately limits that student's potential and thereby does real harm.

- It is not only permissible, but encouraged, to answer a student's question by asking questions in return, these follow-up questions designed to guide the student to reach a correct answer through their own reasoning.

- All student answers demand to be challenged by the instructor and/or by other students. This includes both correct and incorrect answers – the goal is to practice the articulation and defense of one's own reasoning.

- No reading assignment is deemed complete unless and until the student demonstrates their ability to accurately summarize the major points in their own terms. Recitation of the original text is unacceptable. This is why every module contains an "Outline and reflections" question as well as a "Foundational concepts" question in the Conceptual reasoning section, to prompt reflective reading.

- No assigned question is deemed answered unless and until the student demonstrates their ability to consistently and correctly apply the concepts to *variations* of that question. This is why module questions typically contain multiple "Challenges" suggesting different applications of the concept(s) as well as variations on the same theme(s). Instructors are encouraged to devise as many of their own "Challenges" as they are able, in order to have a multitude of ways ready to probe students' understanding.

- No assigned experiment or project is deemed complete unless and until the student demonstrates the task in action. If this cannot be done "live" before the instructor, video-recordings showing the demonstration are acceptable. All relevant safety precautions must be followed, all test equipment must be used correctly, and the student must be able to properly explain all results. The student must also successfully answer all Challenges presented by the instructor for that experiment or project.

Students learning from these modules would do well to abide by the following principles:

- No text should be considered fully and adequately read unless and until you can express every idea *in your own words, using your own examples.*
- You should always articulate your thoughts as you read the text, noting points of agreement, confusion, and epiphanies. Feel free to print the text on paper and then write your notes in the margins. Alternatively, keep a journal for your own reflections as you read. This is truly a helpful tool when digesting complicated concepts.
- Never take the easy path of highlighting or underlining important text. Instead, *summarize* and/or *comment* on the text using your own words. This actively engages your mind, allowing you to more clearly perceive points of confusion or misunderstanding on your own.
- A very helpful strategy when learning new concepts is to place yourself in the role of a teacher, if only as a mental exercise. Either explain what you have recently learned to someone else, or at least *imagine* yourself explaining what you have learned to someone else. The simple act of having to articulate new knowledge and skill forces you to take on a different perspective, and will help reveal weaknesses in your understanding.
- Perform each and every mathematical calculation and thought experiment shown in the text on your own, referring back to the text to see that your results agree. This may seem trivial and unnecessary, but it is critically important to ensuring you actually understand what is presented, especially when the concepts at hand are complicated and easy to misunderstand. Apply this same strategy to become proficient in the use of *circuit simulation software*, checking to see if your simulated results agree with the results shown in the text.
- Above all, recognize that learning is hard work, and that a certain level of frustration is unavoidable. There are times when you will struggle to grasp some of these concepts, and that struggle is a natural thing. Take heart that it will yield with persistent and varied[4] effort, and never give up!

Students interested in using these modules for self-study will also find them beneficial, although the onus of responsibility for thoroughly reading and answering questions will of course lie with that individual alone. If a qualified instructor is not available to challenge students, a workable alternative is for students to form study groups where they challenge[5] one another.

To high standards of education,

Tony R. Kuphaldt

[4] As the old saying goes, "Insanity is trying the same thing over and over again, expecting different results." If you find yourself stumped by something in the text, you should attempt a different approach. Alter the thought experiment, change the mathematical parameters, do whatever you can to see the problem in a slightly different light, and then the solution will often present itself more readily.

[5] Avoid the temptation to simply share answers with study partners, as this is really counter-productive to learning. Always bear in mind that the answer to any question is far less important in the long run than the method(s) used to obtain that answer. The goal of education is to empower one's life through the improvement of clear and independent thought, literacy, expression, and various practical skills.

Appendix C

Tools used

I am indebted to the developers of many open-source software applications in the creation of these learning modules. The following is a list of these applications with some commentary on each.

You will notice a theme common to many of these applications: a bias toward *code*. Although I am by no means an expert programmer in any computer language, I understand and appreciate the flexibility offered by code-based applications where the user (you) enters commands into a plain ASCII text file, which the software then reads and processes to create the final output. Code-based computer applications are by their very nature *extensible*, while WYSIWYG (What You See Is What You Get) applications are generally limited to whatever user interface the developer makes for you.

The `GNU/Linux` computer operating system

> There is so much to be said about Linus Torvalds' `Linux` and Richard Stallman's `GNU` project. First, to credit just these two individuals is to fail to do justice to the *mob* of passionate volunteers who contributed to make this amazing software a reality. I first learned of `Linux` back in 1996, and have been using this operating system on my personal computers almost exclusively since then. It is *free*, it is completely *configurable*, and it permits the continued use of highly efficient `Unix` applications and scripting languages (e.g. shell scripts, Makefiles, `sed`, `awk`) developed over many decades. `Linux` not only provided me with a powerful computing platform, but its open design served to inspire my life's work of creating open-source educational resources.

Bram Moolenaar's `Vim` text editor

> Writing code for any code-based computer application requires a *text editor*, which may be thought of as a word processor strictly limited to outputting plain-ASCII text files. Many good text editors exist, and one's choice of text editor seems to be a deeply personal matter within the programming world. I prefer `Vim` because it operates very similarly to `vi` which is ubiquitous on `Unix`/`Linux` operating systems, and because it may be entirely operated via keyboard (i.e. no mouse required) which makes it fast to use.

Donald Knuth's TEX typesetting system

Developed in the late 1970's and early 1980's by computer scientist extraordinaire Donald Knuth to typeset his multi-volume magnum opus *The Art of Computer Programming*, this software allows the production of formatted text for screen-viewing or paper printing, all by writing plain-text code to describe how the formatted text is supposed to appear. TEX is not just a markup language for documents, but it is also a Turing-complete programming language in and of itself, allowing useful algorithms to be created to control the production of documents. Simply put, TEX *is a programmer's approach to word processing*. Since TEX is controlled by code written in a plain-text file, this means anyone may read that plain-text file to see exactly how the document was created. This openness afforded by the code-based nature of TEX makes it relatively easy to learn how other people have created their own TEX documents. By contrast, examining a beautiful document created in a conventional WYSIWYG word processor such as Microsoft `Word` suggests nothing to the reader about *how* that document was created, or what the user might do to create something similar. As Mr. Knuth himself once quipped, conventional word processing applications should be called WYSIAYG (What You See Is *All* You Get).

Leslie Lamport's LATEX extensions to TEX

Like all true programming languages, TEX is inherently extensible. So, years after the release of TEX to the public, Leslie Lamport decided to create a massive extension allowing easier compilation of book-length documents. The result was LATEX, which is the markup language used to create all ModEL module documents. You could say that TEX is to LATEX as `C` is to `C++`. This means it is permissible to use any and all TEX commands within LATEX source code, and it all still works. Some of the features offered by LATEX that would be challenging to implement in TEX include automatic index and table-of-content creation.

Tim Edwards' `Xcircuit` drafting program

This wonderful program is what I use to create all the schematic diagrams and illustrations (but not photographic images or mathematical plots) throughout the ModEL project. It natively outputs PostScript format which is a true vector graphic format (this is why the images do not pixellate when you zoom in for a closer view), and it is so simple to use that I have never had to read the manual! Object libraries are easy to create for `Xcircuit`, being plain-text files using PostScript programming conventions. Over the years I have collected a large set of object libraries useful for drawing electrical and electronic schematics, pictorial diagrams, and other technical illustrations.

`Gimp` graphic image manipulation program

Essentially an open-source clone of Adobe's `PhotoShop`, I use `Gimp` to resize, crop, and convert file formats for all of the photographic images appearing in the ModEL modules. Although `Gimp` does offer its own scripting language (called `Script-Fu`), I have never had occasion to use it. Thus, my utilization of `Gimp` to merely crop, resize, and convert graphic images is akin to using a sword to slice bread.

`SPICE` circuit simulation program

`SPICE` is to circuit analysis as TEX is to document creation: it is a form of markup language designed to describe a certain object to be processed in plain-ASCII text. When the plain-text "source file" is compiled by the software, it outputs the final result. More modern circuit analysis tools certainly exist, but I prefer `SPICE` for the following reasons: it is *free*, it is *fast*, it is *reliable*, and it is a fantastic tool for *teaching* students of electricity and electronics how to write simple code. I happen to use rather old versions of `SPICE`, version 2g6 being my "go to" application when I only require text-based output. `NGSPICE` (version 26), which is based on Berkeley `SPICE` version 3f5, is used when I require graphical output for such things as time-domain waveforms and Bode plots. In all `SPICE` example netlists I strive to use coding conventions compatible with all `SPICE` versions.

Andrew D. Hwang's `ePiX` mathematical visualization programming library

This amazing project is a `C++` library you may link to any `C/C++` code for the purpose of generating PostScript graphic images of mathematical functions. As a completely free and open-source project, it does all the plotting I would otherwise use a Computer Algebra System (CAS) such as `Mathematica` or `Maple` to do. It should be said that `ePiX` is *not* a Computer Algebra System like `Mathematica` or `Maple`, but merely a mathematical *visualization* tool. In other words, it won't determine integrals for you (you'll have to implement that in your own `C/C++` code!), but it can graph the results, and it does so beautifully. What I really admire about `ePiX` is that it is a `C++` programming library, which means it builds on the existing power and toolset available with that programming language. Mr. Hwang could have probably developed his own stand-alone application for mathematical plotting, but by creating a `C++` library to do the same thing he accomplished something much greater.

`gnuplot` mathematical visualization software

Another open-source tool for mathematical visualization is `gnuplot`. Interestingly, this tool is *not* part of Richard Stallman's GNU project, its name being a coincidence. For this reason the authors prefer "gnu" *not* be capitalized at all to avoid confusion. This is a much "lighter-weight" alternative to a spreadsheet for plotting tabular data, and the fact that it easily outputs directly to an X11 console or a file in a number of different graphical formats (including PostScript) is very helpful. I typically set my `gnuplot` output format to default (X11 on my Linux PC) for quick viewing while I'm developing a visualization, then switch to PostScript file export once the visual is ready to include in the document(s) I'm writing. As with my use of `Gimp` to do rudimentary image editing, my use of `gnuplot` only scratches the surface of its capabilities, but the important points are that it's *free* and that it *works well*.

`Python` programming language

Both Python and C++ find extensive use in these modules as instructional aids and exercises, but I'm listing Python here as a *tool* for myself because I use it almost daily as a *calculator*. If you open a Python interpreter console and type `from math import *` you can type mathematical expressions and have it return results just as you would on a hand calculator. Complex-number (i.e. *phasor*) arithmetic is similarly supported if you include the complex-math library (`from cmath import *`). Examples of this are shown in the Programming References chapter (if included) in each module. Of course, being a fully-featured programming language, Python also supports conditionals, loops, and other structures useful for calculation of quantities. Also, running in a console environment where all entries and returned values show as text in a chronologically-ordered list makes it easy to copy-and-paste those calculations to document exactly how they were performed.

Appendix D

Creative Commons License

Creative Commons Attribution 4.0 International Public License

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution 4.0 International Public License ("Public License"). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions.

Section 1 – Definitions.

a. **Adapted Material** means material subject to Copyright and Similar Rights that is derived from or based upon the Licensed Material and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound recording, Adapted Material is always produced where the Licensed Material is synched in timed relation with a moving image.

b. **Adapter's License** means the license You apply to Your Copyright and Similar Rights in Your contributions to Adapted Material in accordance with the terms and conditions of this Public License.

c. **Copyright and Similar Rights** means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section 2(b)(1)-(2) are not Copyright and Similar Rights.

d. **Effective Technological Measures** means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.

e. **Exceptions and Limitations** means fair use, fair dealing, and/or any other exception or

limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.

f. **Licensed Material** means the artistic or literary work, database, or other material to which the Licensor applied this Public License.

g. **Licensed Rights** means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.

h. **Licensor** means the individual(s) or entity(ies) granting rights under this Public License.

i. **Share** means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.

j. **Sui Generis Database Rights** means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.

k. **You** means the individual or entity exercising the Licensed Rights under this Public License. **Your** has a corresponding meaning.

Section 2 – Scope.

a. License grant.

1. Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:

A. reproduce and Share the Licensed Material, in whole or in part; and

B. produce, reproduce, and Share Adapted Material.

2. Exceptions and Limitations. For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.

3. Term. The term of this Public License is specified in Section 6(a).

4. Media and formats; technical modifications allowed. The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures.

For purposes of this Public License, simply making modifications authorized by this Section 2(a)(4) never produces Adapted Material.

5. Downstream recipients.

A. Offer from the Licensor – Licensed Material. Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.

B. No downstream restrictions. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed Material.

6. No endorsement. Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 3(a)(1)(A)(i).

b. Other rights.

1. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.

2. Patent and trademark rights are not licensed under this Public License.

3. To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties.

Section 3 – License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

a. Attribution.

1. If You Share the Licensed Material (including in modified form), You must:

A. retain the following if it is supplied by the Licensor with the Licensed Material:

i. identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);

ii. a copyright notice;

iii. a notice that refers to this Public License;

iv. a notice that refers to the disclaimer of warranties;

v. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;

B. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and

C. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.

2. You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.

3. If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable.

4. If You Share Adapted Material You produce, the Adapter's License You apply must not prevent recipients of the Adapted Material from complying with this Public License.

Section 4 – Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

a. for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database;

b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material; and

c. You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

Section 5 – Disclaimer of Warranties and Limitation of Liability.

a. Unless otherwise separately undertaken by the Licensor, to the extent possible, the Licensor offers the Licensed Material as-is and as-available, and makes no representations or warranties of any kind concerning the Licensed Material, whether express, implied, statutory, or other. This includes, without limitation, warranties of title, merchantability, fitness for a particular purpose, non-infringement, absence of latent or other defects, accuracy, or the presence or absence of errors,

whether or not known or discoverable. Where disclaimers of warranties are not allowed in full or in part, this disclaimer may not apply to You.

b. To the extent possible, in no event will the Licensor be liable to You on any legal theory (including, without limitation, negligence) or otherwise for any direct, special, indirect, incidental, consequential, punitive, exemplary, or other losses, costs, expenses, or damages arising out of this Public License or use of the Licensed Material, even if the Licensor has been advised of the possibility of such losses, costs, expenses, or damages. Where a limitation of liability is not allowed in full or in part, this limitation may not apply to You.

c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

Section 6 – Term and Termination.

a. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.

b. Where Your right to use the Licensed Material has terminated under Section 6(a), it reinstates:

1. automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or

2. upon express reinstatement by the Licensor.

For the avoidance of doubt, this Section 6(b) does not affect any right the Licensor may have to seek remedies for Your violations of this Public License.

c. For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License.

d. Sections 1, 5, 6, 7, and 8 survive termination of this Public License.

Section 7 – Other Terms and Conditions.

a. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.

b. Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License.

Section 8 – Interpretation.

a. For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose conditions on any use of the Licensed Material that could lawfully

be made without permission under this Public License.

b. To the extent possible, if any provision of this Public License is deemed unenforceable, it shall be automatically reformed to the minimum extent necessary to make it enforceable. If the provision cannot be reformed, it shall be severed from this Public License without affecting the enforceability of the remaining terms and conditions.

c. No term or condition of this Public License will be waived and no failure to comply consented to unless expressly agreed to by the Licensor.

d. Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal processes of any jurisdiction or authority.

Creative Commons is not a party to its public licenses. Notwithstanding, Creative Commons may elect to apply one of its public licenses to material it publishes and in those instances will be considered the "Licensor." Except for the limited purpose of indicating that material is shared under a Creative Commons public license or as otherwise permitted by the Creative Commons policies published at `creativecommons.org/policies`, Creative Commons does not authorize the use of the trademark "Creative Commons" or any other trademark or logo of Creative Commons without its prior written consent including, without limitation, in connection with any unauthorized modifications to any of its public licenses or any other arrangements, understandings, or agreements concerning use of licensed material. For the avoidance of doubt, this paragraph does not form part of the public licenses.

Creative Commons may be contacted at `creativecommons.org`.

Appendix E

References

The ARRL Antenna Book, Eleventh Edition, The American Radio Relay League, Inc., Newington, CT, 1968.

Bible, Steven, "Crystal Oscillator Basics and Crystal Selection for rfPICTM and PICmicro Devices", Application Note AN826, Microchip Technology Inc., 2002.

Industrial Electronics Reference Book, Fourth Printing, John Wiley & Sons, June 1953.

"Introduction to SAW Filter Theory & Design Techniques" whitepaper, document 102020ED, API Technologies Corporation, 2018.

Kay, Art and Green, Tim, *Analog Engineer's Pocket Reference*, Fifth Edition, document SLYW038C, Texas Instruments, Dallas, TX, 2019.

Ruppel, Clemens C.W. and Reindl, Leonhard, "SAW Devices for Spread Spectrum Applications", Proceedings of ISSSTA'95 International Symposium on Spread Spectrum Techniques and Applications, pages 713-719, Volume 2, 1996.
"SAW Products", document number SAW-05-18, Microsemi Corporation, 2018.

"Simpson 270 Series 5 Volt-Ohm-Milliammeter INSTRUCTION MANUAL", part number 06-111642, edition 15, Simpson Electric Company, Lac du Flambeau, WI, July 2017.

Steinmetz, Charles Proteus, *Theory and Calculation of Electric Circuits*, First Edition, Sixth Impression, McGraw-Hill Book Company, Inc., New York, 1917.

Appendix F

Version history

This is a list showing all significant additions, corrections, and other edits made to this learning module. Each entry is referenced by calendar date in reverse chronological order (newest version first), which appears on the front cover of every learning module for easy reference. Any contributors to this open-source document are listed here as well.

9 May 2025 – corrected a typographical error in image_5811 where I showed Ω rather than "S" for the unit of admittance. This correction is courtesy of Brian Sharpes.

13 March 2025 – added V_{out} labels to several of the Bode plot illustrations.

24 September 2024 – added comments about power limitations of mechanical-electrical filters.

16 September 2024 – divided the Introduction chapter into sections, one with recommendations for students, one with a listing of challenging concepts, and one with recommendations for instructors.

23 February 2024 – added more content to the "Square wave to sine wave" Conceptual Reasoning question.

31 January 2024 – fixed a typographical error in image_5496 showing interposed digits in the total impedance value, courtesy of Gavin Koppel. Also made some minor edits to text formatting in that section to improve clarity.

19 October 2023 – added a new Tutorial section on output-limited filter networks.

13 May 2023 – added Case Tutorial section on harmonic filters in HVDC substations.

22 February 2023 – added more questions to the Introduction chapter.

28-29 November 2022 – placed questions at the top of the itemized list in the Introduction chapter prompting students to devise experiments related to the tutorial content.

18 July 2022 – typographical error correction in the Introduction chapter, courtesy of Caleb Colby.

11 July 2022 – minor edit to image_5532 and image_5583 to make them more suitable for examples in non-filter-related tutorials.

18 May 2022 – placed the "Practice: complex number calculations" questions into its own file (case_complexpractice.latex) so it may be shared amongst multiple modules.

2 March 2022 – minor edits to two equations in the "Example: RC filter design" Case Tutorial chapter, to include units of Hz and Ω in the formulae.

17 January 2022 – minor edits to the "Identifying (more) filter types" Conceptual Reasoning question.

26-27 December 2021 – added new Quantitative Reasoning question on filter design using IEC-60063 (E24) standard values, and also edited the Tutorial to elaborate on the definition of cutoff frequency. Also elaborated on passbands and stopbands, and corrected an error in the log-axis Bode plots where cutoff was not happening at the -3 dB attenuation point.

22-24 December 2021 – added review_16 as an introduction to the Tutorial, also added more section titles and questions.

18 November 2021 – added Discrete Fourier Transform and Spectrum Analyzer C++ programming examples to the Programming References chapter.

24 October 2021 – added comment about "brick wall" idealized filter response.

14 October 2021 – minor edits to that new Case Tutorial section showing how to manually test and plot the response of a low-pass RC filter network.

29 September 2021 – added a new Case Tutorial section showing how to manually test and plot the response of a low-pass RC filter network.

13 September 2021 – added another reference in the Tutorial to the effect that $X = R$ at cutoff frequency for a simple (i.e. single-pole) RC or LR filter.

10 June 2021 – divided Tutorial into sections, and added content regarding SAW filters.

8 May 2021 – commented out or deleted empty chapters.

23 February 2021 – minor edits to the Tutorial, including the addition of "ideal" versus "real" filter Bode plot responses shown in image_3460.

12 November 2020 – added instructor notes to some of the Questions.

26 October 2020 – significantly edited the Introduction chapter to make it more suitable as a pre-study guide and to provide cues useful to instructors leading "inverted" teaching sessions.

15 September 2020 – added more instructor notes to the "Deriving a formula for Q" Quantitative reasoning question.

15 August 2020 – added alternative filter block diagrams to the "Filter Block Diagrams" Conceptual question.

17 April 2020 – added gnuplot label on code to distinguish it as gnuplot code and not C++.

16 April 2020 – added new Conceptual Reasoning question on filter types as a truth table.

15 April 2020 – minor edits.

19 March 2020 – edited one series-resonant filter to be parallel-resonant instead in the "Identifying (more) filter types" Conceptual Reasoning problem.

12 March 2020 – added another Conceptual Reasoning problem.

9 March 2020 – added more pages to the tutorial about the effects of filtering in the frequency domain, and how filters may be used for wave-shaping.

4 March 2020 – corrected typographical error on the definition of cutoff frequency (I said it was 0.707% of the input, but I should have said 70.7% of the input).

31 January 2020 – added roll-off as a concept in the Tutorial.

5 January 2020 – added bullet-list of relevant programming principles to the Programming References section.

3 January 2020 – added Programming References chapter, with a section showing how to model simple filter Bode plots using C++.

17 December 2019 – added Technical Reference on the topic of decibels.

27 November 2019 – added some questions.

16 June 2019 – minor edits to diagnostic questions, replacing "no multiple faults" with "no coincidental faults".

10 June 2019 – minor edits.

6 June 2019 – Added inductor-resistor filtering networks to "Filter type and cutoff identifications" question.

29 May 2019 – Added questions.

28 May 2019 – converted Simplified Tutorial into Tutorial, since there is no Full Tutorial written (yet). Also, added questions.

20 May 2019 – changed "Amps" to "Amperes".

3 February 2019 – added conceptual question about Output jack on analog Volt-Ohm-Milliammeters, which uses filtering to block the DC component of a mixed AC-DC voltage signal.

5 November 2018 – retitled Historical References section(s) so as to not be redundant to the "Historical References" chapter.

September 2018 – renamed "Derivations and Technical References" chapter to "Historical References".

August 2018 – document first created.

Index