Set Phasors to Analyze Circuits

Abe Karplus

California State Science Fair 2011

Abstract

My objective with this project is to create a computer program (ZAP CIRCUIT ANALYZER) for AC and DC analysis of linear circuits, which are circuits with only resistors, capacitors, inductors, voltage sources, and current sources. A circuit analyzer is very useful to circuit designers, enabling them to find out what their circuits would do before actually building them. When I implement the automatic varying of the values of components, the program could also aid in the design of circuits. In addition, a circuit analyzer is a key component of a more general time-domain circuit simulator.

I wrote the program in Python, using the Numpy and PyGUI packages, which I learned how to use for this project. I wrote the interface to a gnuplot plotting program.

I wrote two different algorithms to perform the actual analysis of the circuits. The first algorithm has four steps. First, it sets up systems of linear equations using the branch equations (Ohm's law) and Kirchhoff's laws. It uses Numpy matrices with the node voltages as variables. To deal with voltage sources, it must use supernodes, sets of nodes connected by voltage sources. Second, it solves the matrix. Third, the algorithm calculates the element voltages (a simple matter of subtracting connected node voltages). The fourth step is to calculate current for most elements using the branch equations. For voltage sources, the algorithm must instead use a graph algorithm and Kirchhoff's Current Law to calculate currents from the currents of elements connected to the voltage source.

The second algorithm is much simpler, because it uses a less-common, but much more elegant, method known as the extended sparse tableau. This involves setting up a large but (as the name implies) sparse matrix with not only node voltages, but also element voltages and element currents as variables. To fill the matrix, the algorithm uses Kirchhoff's Current Law, Kirchhoff's Voltage Law, and the branch equations more directly than in the first method. When the matrix equation is solved, the solution contains all the information about the circuit that would have been computed with the first algorithm.

Both algorithms do not actually use standard matrix "solve" methods, but rather the least-squared algorithm (provided with Numpy), which allows them to handle underdetermined or overdetermined circuits. Also, both deal with AC by using complex phasors. These phasors represent AC voltage or current, but can be manipulated in matrices just like real numbers in DC analysis. Phasors represent sinusoidal waves of the form `amplitude * e^(j * (omega * time + phase))`, where `j` is the square root of minus one, and `omega` is the frequency in radians per second. The phasor itself has the form `amplitude * e^(j * phase)`, since frequency and time are circuit-wide properties.

The second algorithm can deal with both AC and DC using completely the same code, because it uses admittance instead of impedance for capacitors to avoid divide-

by-zero problems (DC is frequency zero). Impedance and admittance are the AC equivalent of resistance and conductance in DC analysis, that is, admittance is 1/impedance. The impedance of a resistor is its resistance, the impedance of an inductor is `j * omega * inductance`, and the impedance of a capacitor is `1 / (j * omega * capacitance)`.

Zap is able to "sweep" frequency, by performing the main analysis with the same circuit at multiple frequencies. I plan to add a method for it to sweep element values (such as capacitance) as well. Zap has a GUI interface with RDB and plot (PDF) output.

My dad, a computer scientist, mentored me on this project. He provided me with exercises to help me learn Python, suggested the Numpy package, and helped me learn circuit analysis from the two books *Introduction to Electric Circuits*, from which the node analysis method came, and *Linear and Nonlinear Circuits*, from which the tableau analysis method came. He also suggested the title for the project.

Table of Contents

Introduction

This project, SET PHASORS TO ANALYZE CIRCUITS, is not a hypothesis-driven research project. Rather, it is an engineering project with the intent of building a *circuit analyzer*. A circuit analyzer is a computer program that analyzes electrical circuits, so that an engineer can find out what their circuits will do without actually building them. Of particular interest is how the circuit behaves differently at different frequencies.

The ZAP CIRCUIT ANALYZER can only analyze *linear circuits*, which are circuits with only resistors, capacitors, inductors, and voltage and current sources. It can compute the voltage and current at any point in the circuit, for both *direct* and *alternating currents*.

This poster is **not** a final report. It is a progress report on the development of ZAP CIRCUIT ANALYZER.

Python

I wrote the ZAP CIRCUIT ANALYZER entirely in Python, with the help of the Numpy and PyGUI packages for Python. Python is a popular cross-platform very-high-level object-oriented interpreted scripting language created by Guido van Rossum. I chose it because I had learned it recently and it seemed a good fit for this project. The adjective "*Pythonic*" refers to code that is in the style and philosophy of Python, which emphasizes clarity, readability, and simplicity.

Numpy

The matrices used in this project are from the Numpy package, which provides support for a diverse array of operations on them.

PyGUI

The Graphical User Interface (GUI) for Zap Circuit Analyzer I wrote using PyGUI, a GUI builder for Python written by Greg Ewing with the intent of presenting a Pythonic Application-Programmer Interface (API) and supporting native look and feel on most major platforms.

Gnuplot

All the plots in this report were created with gnuplot, a free plotting program from the GNU Software Foundation.

Circuit Theory

All of the circuit theory explained in this section concerns *steady-state analysis*, which is the behavior of the circuit after it has stabilized. To model circuits before they have stabilized (*transient*), time-domain simulation is required, something ZAP CIRCUIT ANALYZER cannot yet perform.

Elements and Nodes

Circuits are best modeled as *graphs*, which are collections of *vertices* and *edges*. Edges are simply pairs of vertices, indicating a relation between the two vertices. In a circuit, *elements* (sometimes called *branches*) are edges and *nodes* are vertices. Elements have the properties voltage and current, and define a relation between these (a branch equation). Nodes have only the property voltage, which is defined relative to a *ground node*, which has by definition a voltage of zero.

Charge, Current, Voltage

The basic properties of electricity are *charge*, *current*, and *voltage*, measured in Coulombs, Amps, and Volts, respectively. A very useful analogy for understanding these is the "water" analogy. Charge is simply the amount of water (electricity), current is its flow rate—quite literally its current, and voltage corresponds to water pressure. Amps are just Coulombs per second, but the relation between Volts and Amps depends on the element. Current is represented by `i` and voltage by `v`. Charge usually does not come up in circuit analysis.

Kirchhoff's Laws

Current

Nodes cannot store charge—they are simply connections between elements. Therefore, any current entering a node must immediately leave it. This is *Kirchhoff's Current Law* (KCL): The algebraic sum of currents entering and leaving a node is zero. "Algebraic sum" here means that currents leaving the node are subtracted.

Voltage

Normally, *Kirchhoff's Voltage Law* (KVL) is explained as this rule: The algebraic sum of element voltages around any loop is zero. "Algebraic sum" here means that the direction (sign) of the element voltage is allowed for. However, this holds algebraically true for any graph where edge values are the difference between vertex values. Therefore, what KVL really means is this: The voltage of an element is the difference between the voltages of its connected nodes. This version is the one my program actually uses.

DC Branch Equations

Direct current (DC) is when the voltage and current of a circuit are constant (*time-invariant*). The three DC circuit element types are *resistor*, *voltage source*, and *current source*.

Resistors

Resistors have the property *resistance* (`R`), which relates current to voltage by the equation `v=iR` (Ohm's law).

Voltage Sources

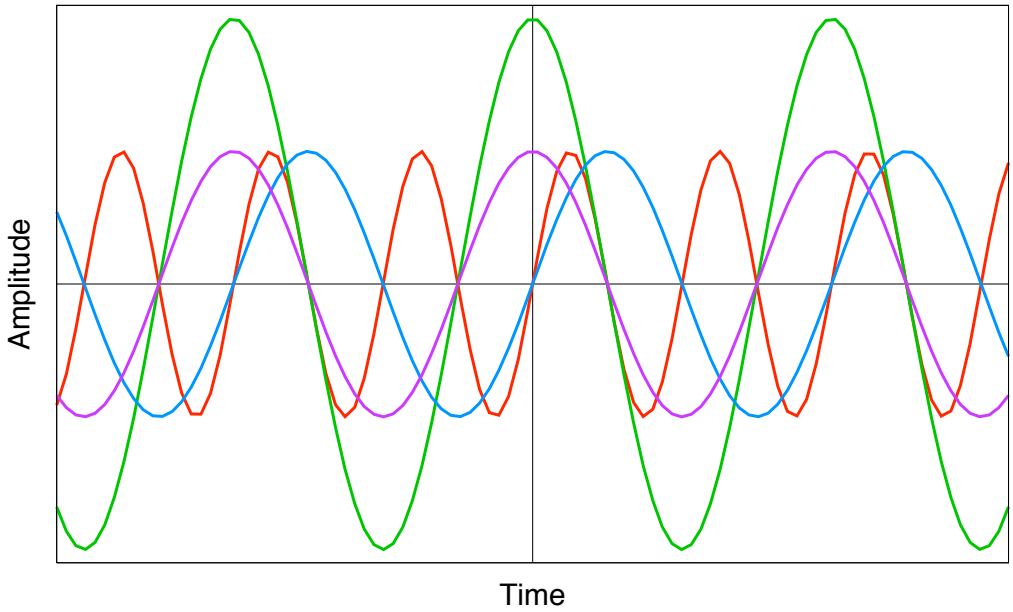
Voltage sources have the property *voltage supplied*, and they guarantee their voltage to be equal to it. The current of a voltage source is not simply determined by its voltage.

Current Sources

Current sources have the property *current supplied*, and they guarantee their current to be equal to it. The voltage of a current source is not simply determined by its current.

Sinusoids and Phasors

So far, everything has assumed that the current and voltage are constant, not changing with time (once they have stabilized). However, this situation does not occur in circuits that use *alternating current* (AC). In AC circuit analysis, the voltage and current are *sinusoids* instead of constants. Sinusoids look like this:



The *amplitude* of a sinusoid is the peak value, the *period* is the time until it repeats, and the *phase* how much it is offset in time from a "model" sinusoid. The purple (`cos(x)`) and blue (`sin(x)`) sinusoids shown have the same amplitude, and frequency, but different phases. The green (`2cos(x)`) and purple sinusoids differ only in amplitude, and the red (`sin(2x)`) and blue sinusoids only in period. The *frequency* of a sinusoid is defined as the reciprocal of its phase; that is, how frequently it repeats. It is a fact of linear circuits that the frequency is constant throughout the circuit, with only the amplitude and phase varying for individual node voltages, element voltages, and element currents. To greatly simplify analysis of AC circuits, the amplitude and phase are represented together, as a single complex number. This complex number is known

as a phasor.

Any sinusoid can be represented as an equation of the form `B*e^(j(ωt+ϕ))`, where `B` is the amplitude, `e` is Euler's number (approximately `2.718281828459`), `j` is the unit imaginary number (the square root of negative one), `ω` is the frequency in radians per second (see "Hertz and Radians per Second" below), `t` is the time, and `ϕ` is the phase. A phasor removes the circuit-wide information, leaving amplitude and phase, so has the form `B*e^(jϕ)`. What is really neat about this is that the same equations and laws used in DC circuit analysis can be used in AC circuit analysis, but with complex phasors instead of real numbers.

Hertz and Radians per Second

Hertz and *radians per second* are both measures of frequency, the former easier for humans, and the second easier for circuit analysis. Hertz, or cycles per second, is simply how often the sinusoid repeats every second. Radians are a measure of angle defined so that on the unit circle (a circle with radius `1`), an angle of `R` radians cuts off an arc of length `R`. A complete circle (360°) is then 2π radians, the circumference of the unit circle. One period of a sinusoid is equivalent to a complete revolution, so frequency in radians per second is 2π times the frequency in Hertz.

AC Branch Equations

The property *impedance* of an element is quite similar to resistance, but it is complex and depends on the frequency of the circuit. Two elements that are thus *frequency-dependent* are *capacitors* and *inductors*.

Capacitors

Capacitors have the property *capacitance* (`C`). Their impedance is `1/(jωC)`, so current is related to voltage by the equation `v=i/(jωC)`. To avoid divide-by-zero errors when frequency is zero, it is sometimes written `i=v(jωC)`.

Inductors

Inductors have the property *inductance* (`L`). Their impedance is `jωL`, so current is related to voltage by the equation `v=i(jωL)`.

DC Behavior

When frequency is zero (DC), the equations for capacitors and inductors reduce to `i=0` and `v=0`, respectively. This means that capacitors in a DC circuit behave like gaps or breaks in the circuit (*open circuit*), while inductors behave like ideal wires (*short circuit*).

Frequency Response and Frequency Sweeping

All the circuit theory so far described concerns single frequencies. However, one of the most important properties of an AC circuit is its *frequency response*, the change in current and voltage for varying frequency. (The frequency still is the same in each part of the circuit.) The technique of analyzing a circuit at multiple frequencies is known as *frequency sweeping*.

Algorithms

I wrote two different algorithms to perform the actual analysis of the circuits, presented in the order I wrote them. The second is much cleaner and is the one that the current version of ZAP CIRCUIT ANALYZER uses. Both algorithms do not actually use standard matrix "solve" methods, but rather the least-squared algorithm (provided with Numpy), which allows them to handle insufficiently defined circuits or inconsistent circuits.

Node Matrix

The basic idea of the node matrix algorithm is to have one equation per node, all in a matrix. It uses other methods to find the element voltage and current. The first step for the algorithm is to set up the node voltage matrix using the branch equations and Kirchhoff's laws. To allow for voltage sources, supernodes must be used. These are sets of nodes connected by voltage sources and treated as single nodes. The algorithm then solves the matrix, and uses the calculated node voltages to calculate elements voltages (a simple matter of subtracting connected node voltages). Finally, it calculates element current. For most elements, it can simply use the branch equations, but voltage source branch equations do not define current. For voltage sources, it must then use a graph algorithm and KCL to calculate the current of a voltage source from those of connected elements.

Extended Sparse Tableau

The extended sparse tableau algorithm is much simpler and cleaner than the node matrix algorithm, but surprisingly uncommon. It puts all the variables—node voltage, element voltage, element current—in one large ("extended") matrix ("tableau"), which then has a much lower density of numbers in it ("sparse"). The algorithm puts in an equation for KCL for each node, KVL for each elements, and the branch equation for each element. It then solves the matrix and gets all the answers. It can deal with DC as well as AC with the exact same code, simply by treating DC as frequency `0`.

Future Work

I accomplished one of my Future Work goals from county science fair: integrating the gnuplot control into the user interface. Now, a potential feature would be interactive plotting (not just to a file). To do this I could not use gnuplot, but would have to create a plotting program using a PyGUI canvas backend (which I have started on). The most useful circuit analysis feature to add is component sweeping, automatically altering the value of components for a specific frequency response. This would not be extremely difficult to add. An ambitious user-interface project would be to add a graphical means of circuit entry (to supplement or replace the textual one). One last feature, which could be an entire project, would be the creation of Zap Circuit Simulator for time-domain simulation (instead of frequency-domain analysis) of circuits. This would allow nonlinear components, multiple frequencies, and more.

References and External Modules & Software

- *Introduction to Electric Circuits, 8th Edition*
  by Richard C. Dorf and James A. Svoboda.
  Published by John Wiley and Sons. © 2010.
- *Linear and Nonlinear Circuits*
  by Leon O. Chua, Charles A. Desoer, and Ernest S. Kuh.
  Published by McGraw-Hill. © 1987.
- Gnuplot 3.8j.
  `http://gnuplot.info/`
- Numpy 1.5.1.
  `http://numpy.scipy.org/`
- PyGui 2.3.3 by Greg Ewing.
  `http://cosc.canterbury.ac.nz/greg.ewing/python_gui/`

Acknowledgements

I wish to thank my father, Kevin Karplus, who mentored me on this project and helped me learn Python and circuit theory (see his Mentor Statement). I also wish to thank my mother, Michele Hart, for her support on writing. Lastly, I wish to thank the creators of Python, Numpy, Gnuplot, and PyGUI, as well as the authors of *Introduction to Electric Circuits* and *Linear and Nonlinear Circuits*.

Mentor Statement

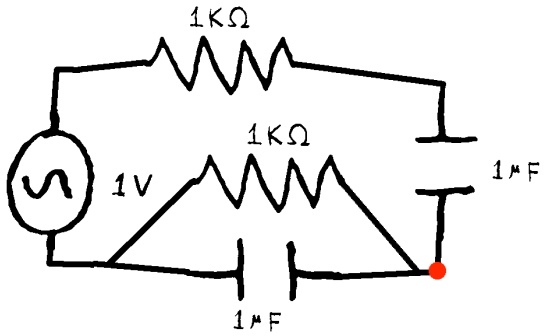
Abe started learning Python in summer 2010. I provided him with *Python in a Nutshell*, but he did most of his learning from the online Python documentation. When he decided to try doing circuit analysis, I pointed him to the Numpy package and checked out three intro books on circuit theory from the university library for him. I explained to him the notion of phasors for representing sinusoids and `e^(jωt)=cos(ωt)+j*sin(ωt)`. I also helped him install new versions of Python, Numpy, PyObjC, and PyGUI.

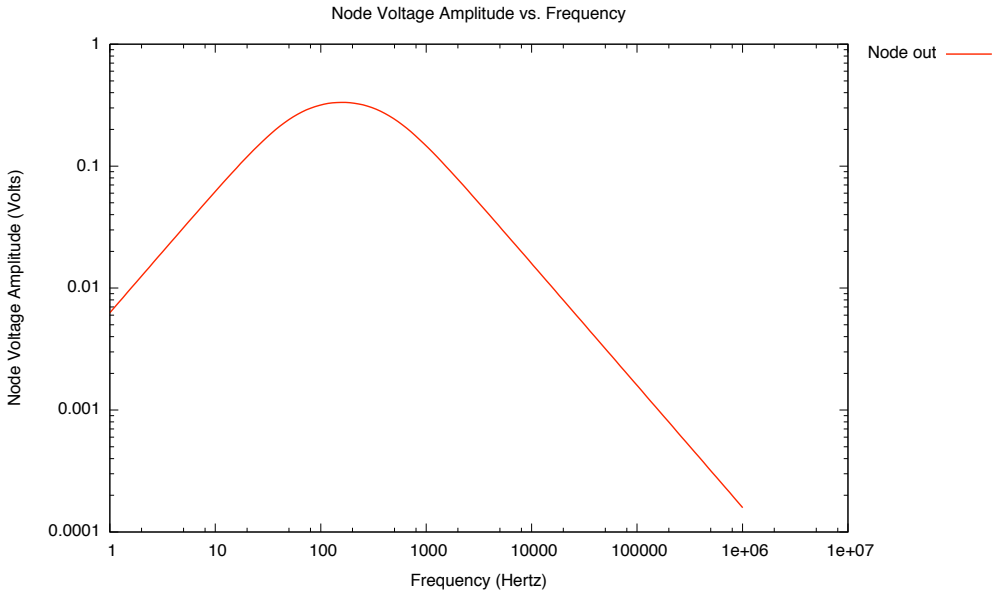
We discussed some of the algorithms, with me suggesting both the graph algorithm for solving for currents of voltage sources and the switch to the tableau method. All the code is his own. I provided only minimal debugging help once or twice —mainly trivial typo finding.

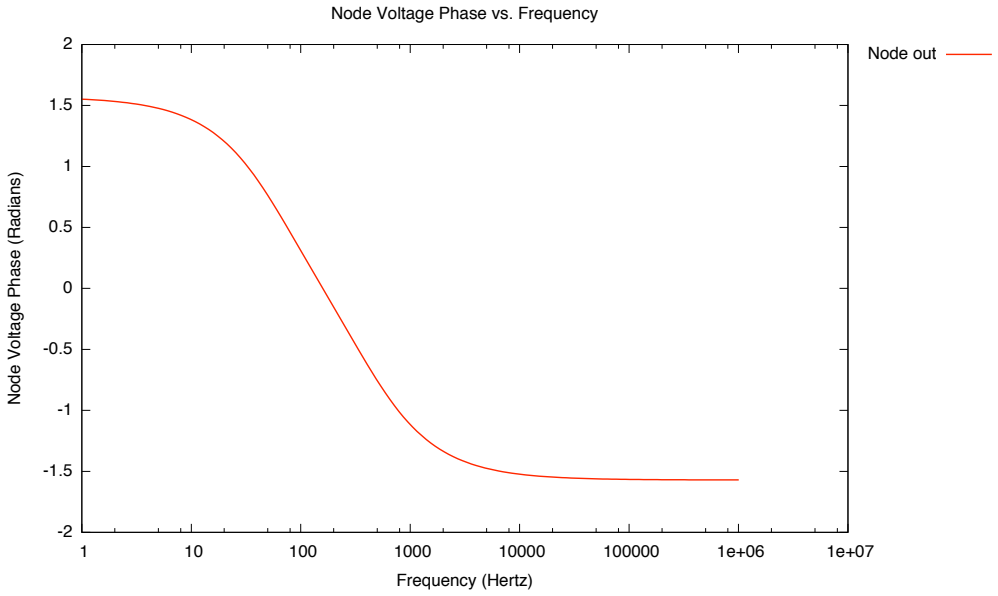
I provided some suggested circuits for him to analyze and suggested the title for the project.

Examples

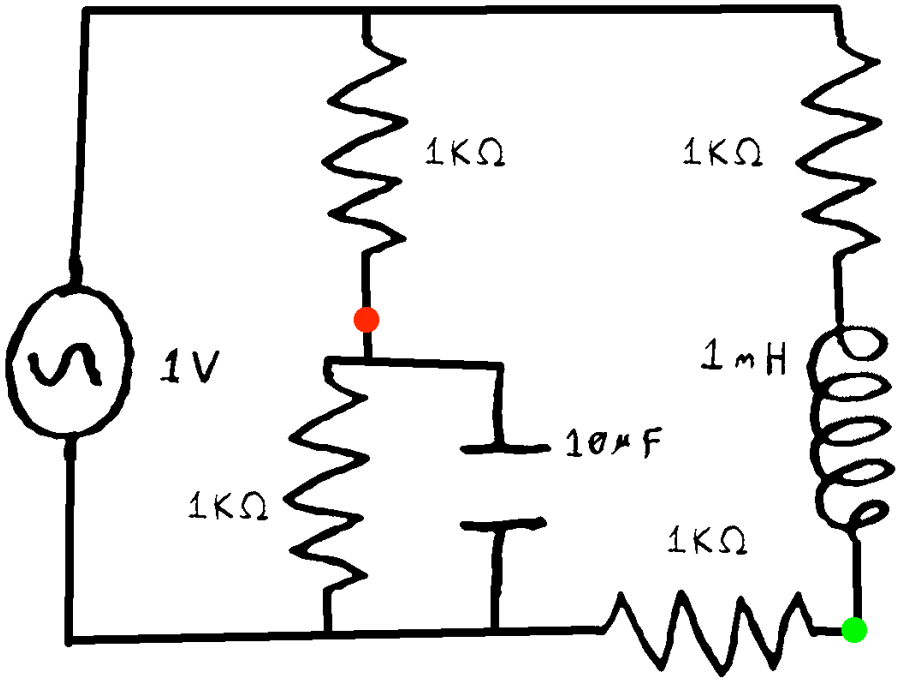
Wien Bridge

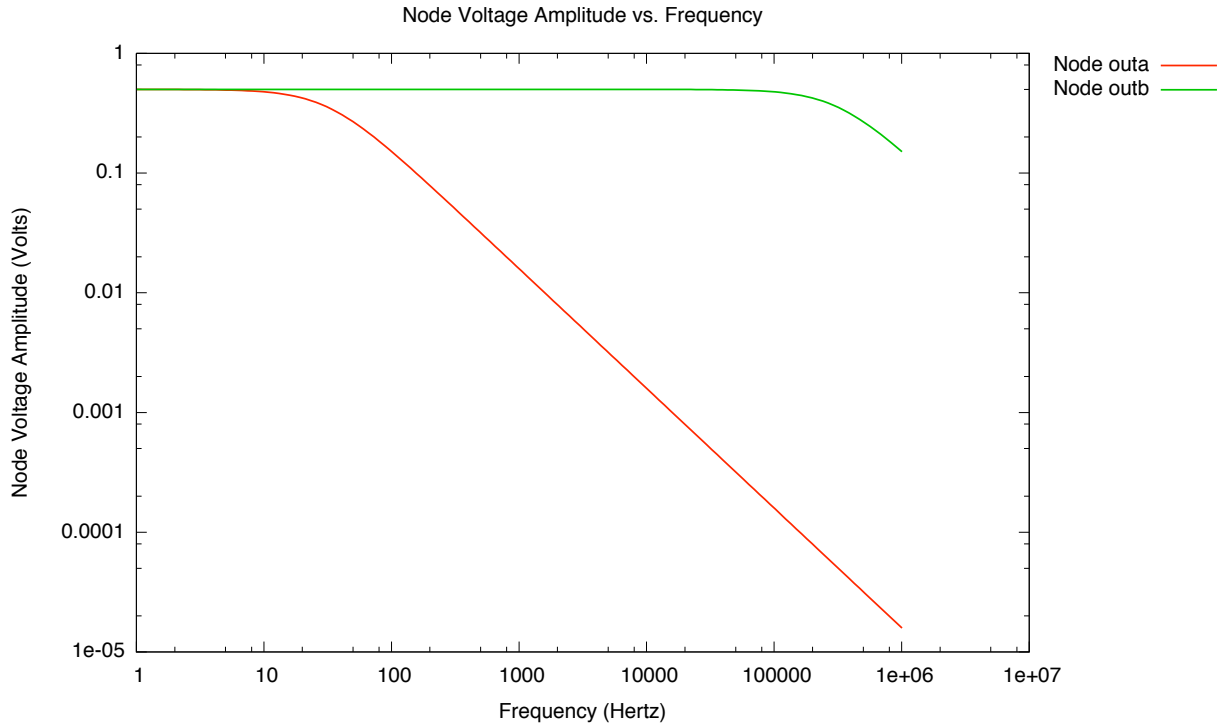




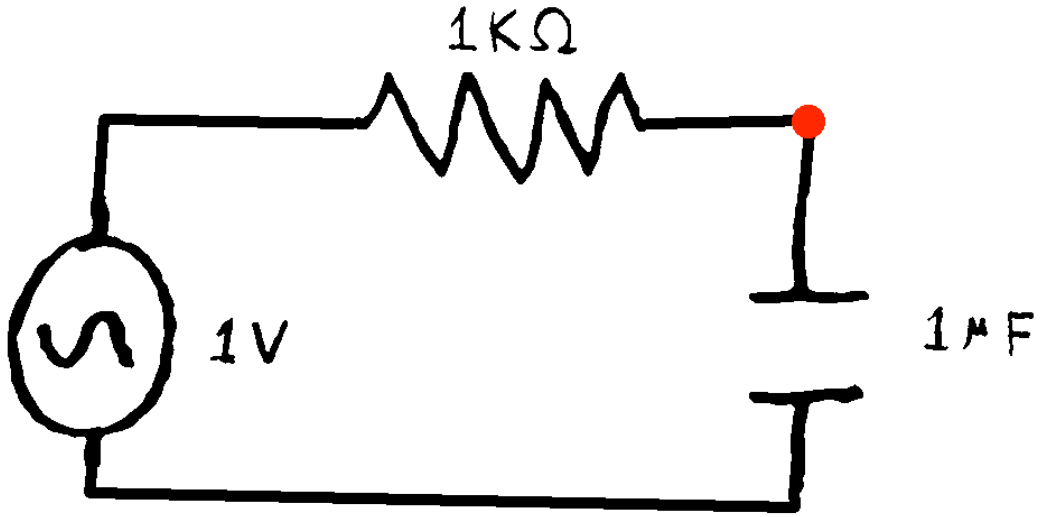


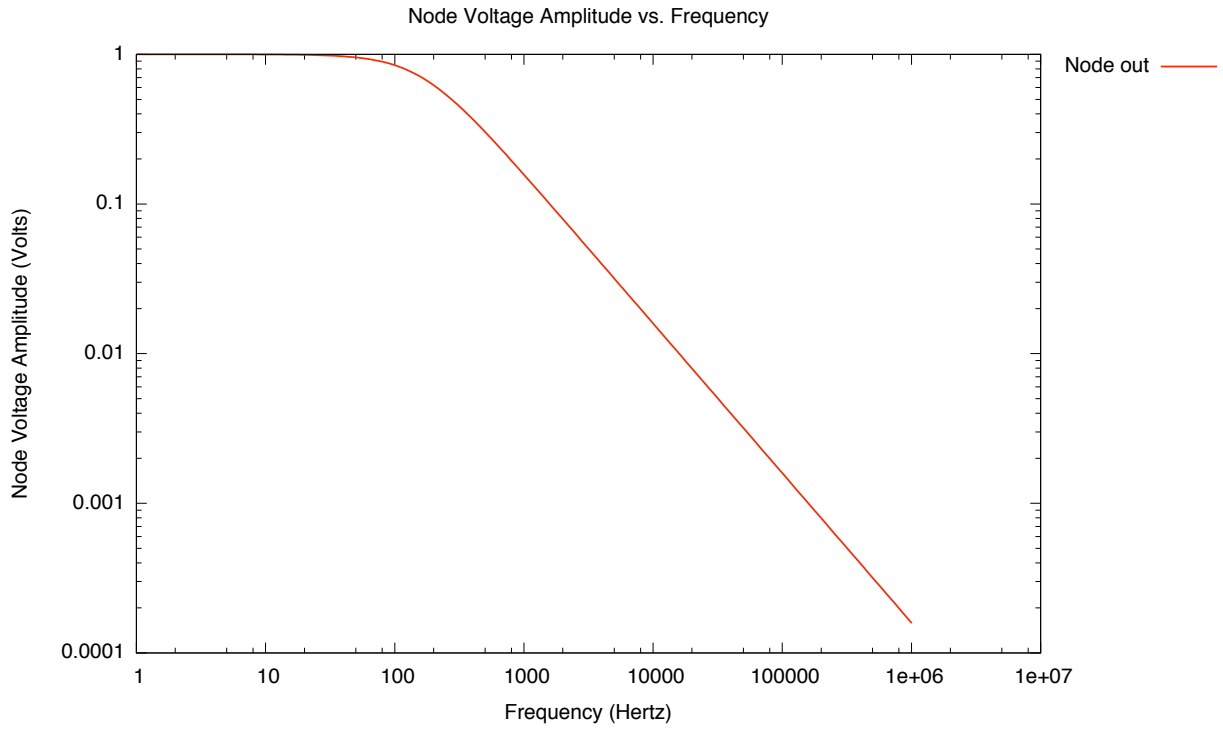
Maxwell Bridge



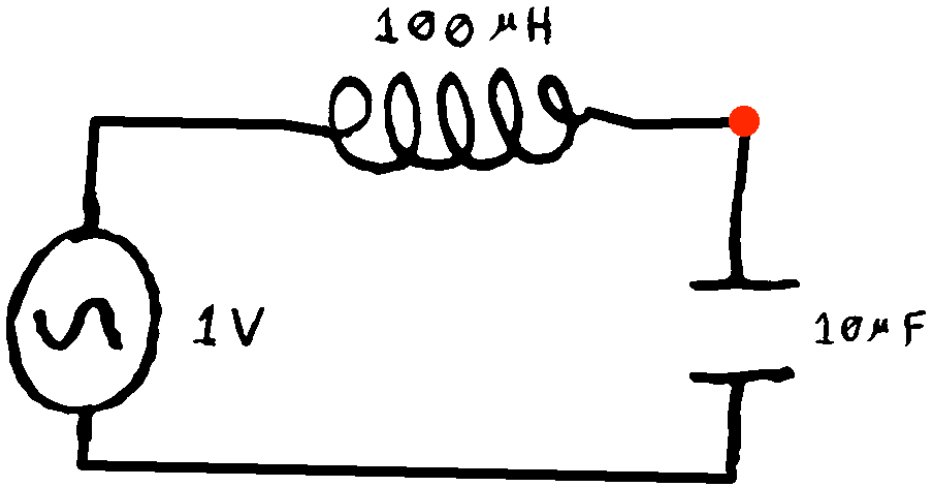


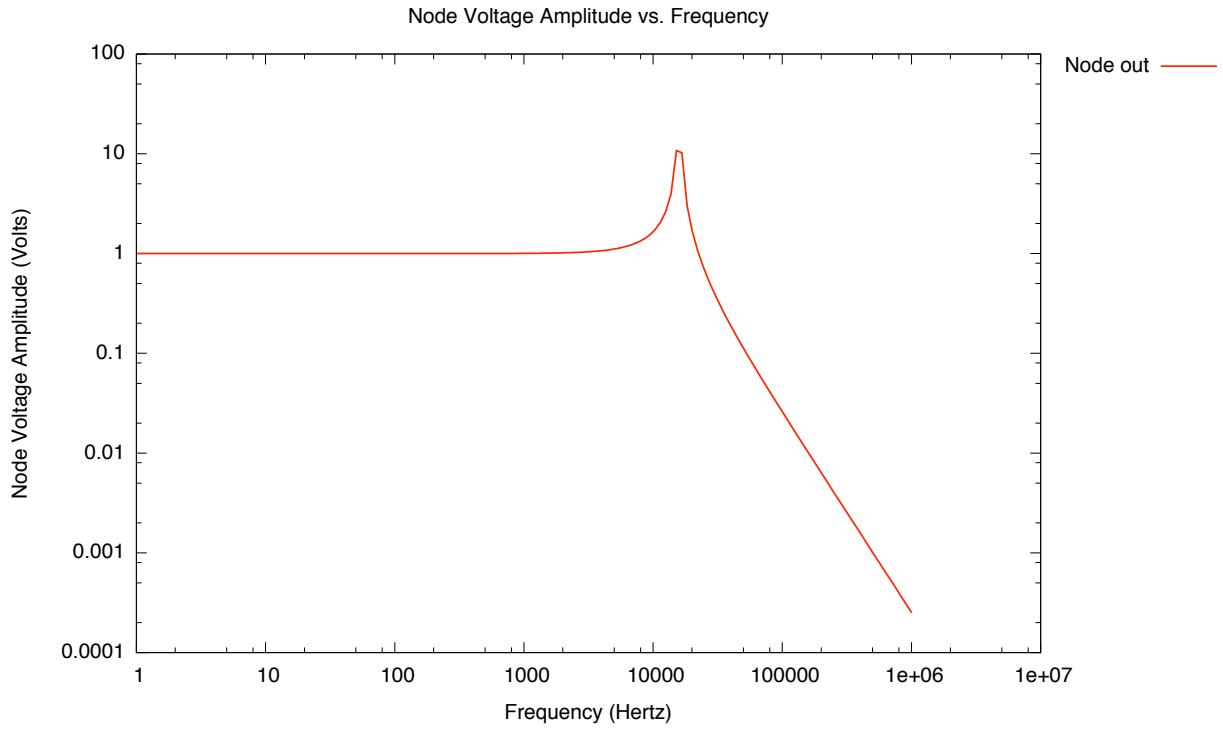
RC Low Pass


1KΩ
1V
1μF

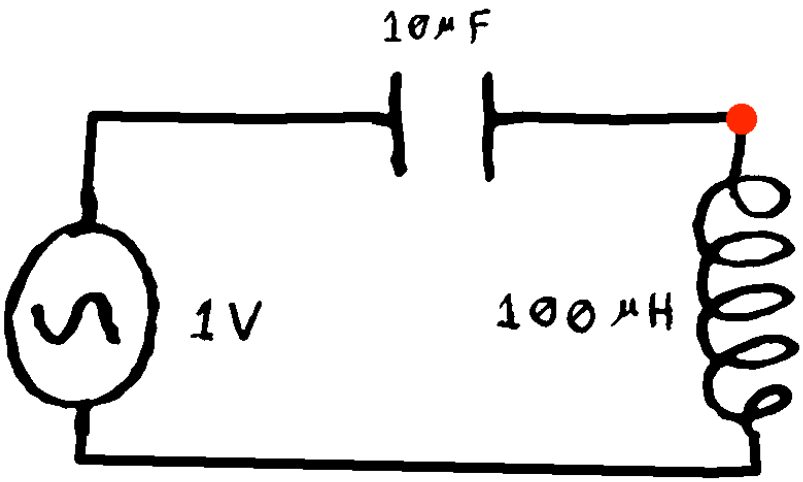

Node Voltage Amplitude vs. Frequency
Node out
Node Voltage Amplitude (Volts)
1
0.1
0.01
0.001
0.0001
1
10
100
1000
10000
100000
1e+06
1e+07
Frequency (Hertz)

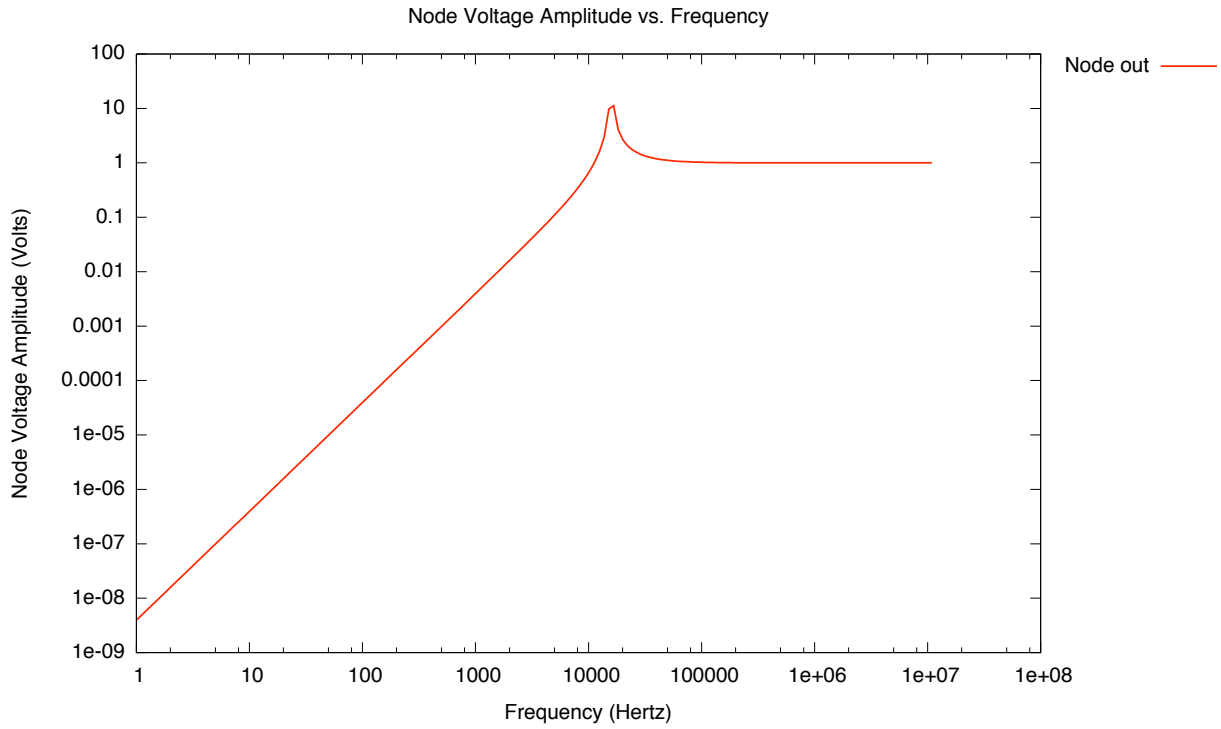
LC Low Pass


100 μH
1V
10μF

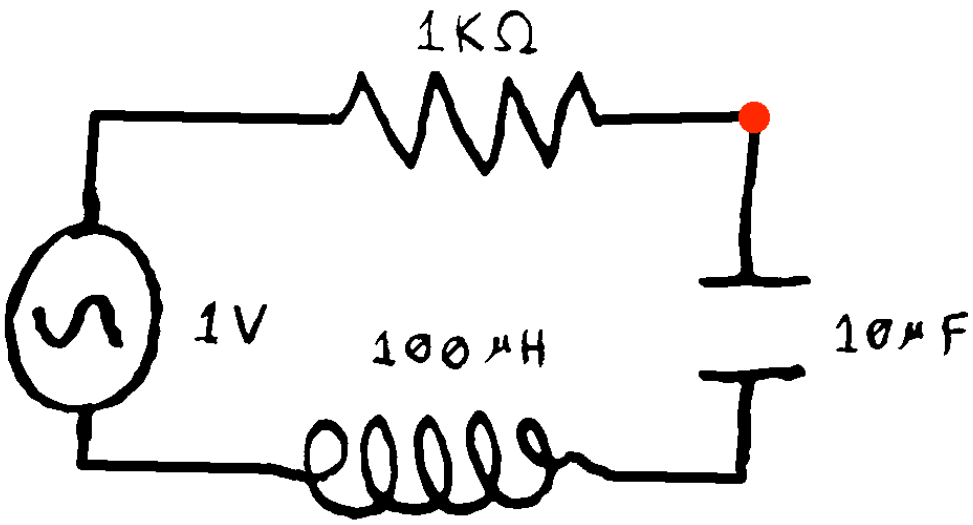

Node Voltage Amplitude vs. Frequency
Node out
Node Voltage Amplitude (Volts)
100
10
1
0.1
0.01
0.001
0.0001
1
10
100
1000
10000
100000
1e+06
1e+07
Frequency (Hertz)

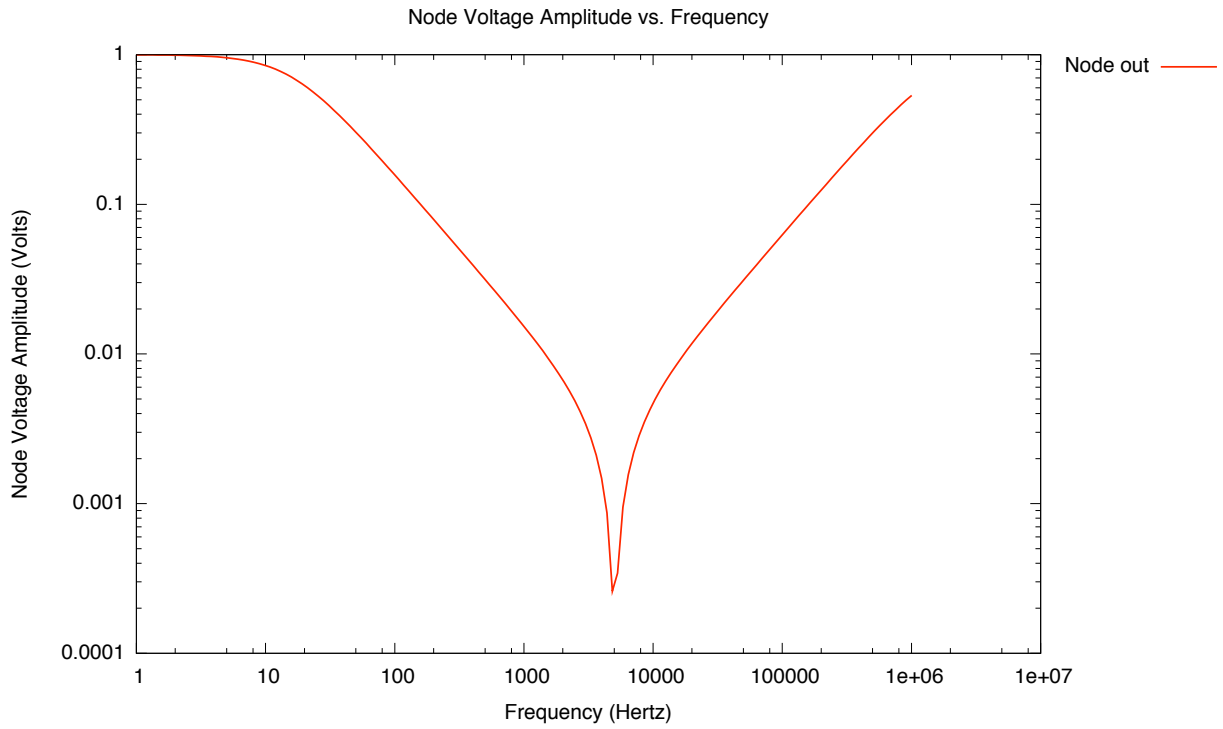
LC High Pass


10μF
1V
100μH

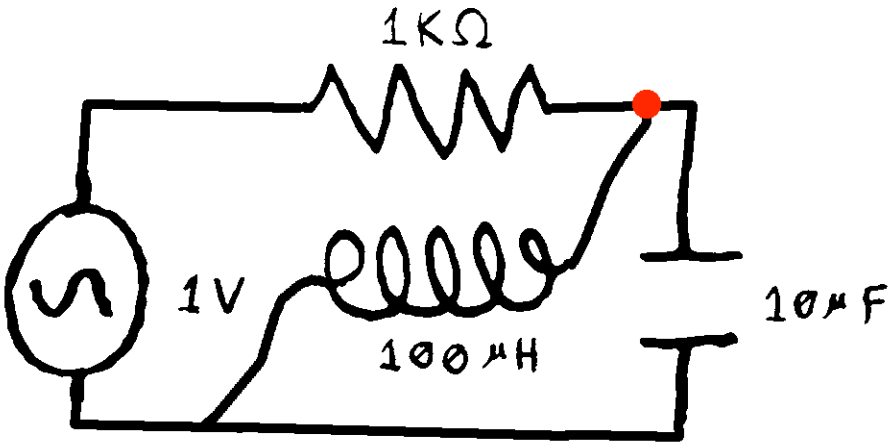

Node Voltage Amplitude vs. Frequency
Node out
Node Voltage Amplitude (Volts)
100
10
1
0.1
0.01
0.001
0.0001
1e-05
1e-06
1e-07
1e-08
1e-09
1
10
100
1000
10000
100000
1e+06
1e+07
1e+08
Frequency (Hertz)

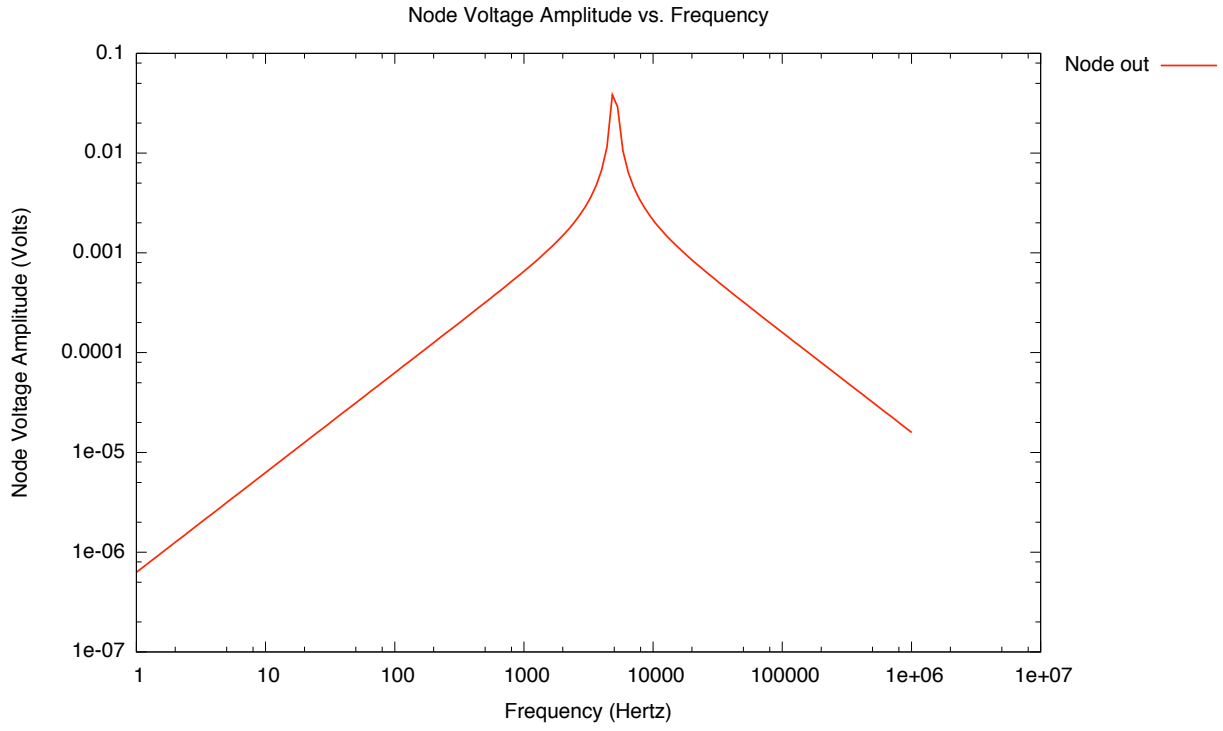
LC Band Reject





LC Band Pass


1KΩ
1V
100μH
10μF


Node Voltage Amplitude vs. Frequency
Node out
Node Voltage Amplitude (Volts)
0.1
0.01
0.001
0.0001
1e-05
1e-06
1e-07
1
10
100
1000
10000
100000
1e+06
1e+07
Frequency (Hertz)