LRC - Examples

```
# LRC - Examples
#
#           This worksheet contains examples of the use of LRC a Python/SageMath class
#           LRC computes series/parallel properties of various inductors, resistors,
and capacitors.
#           The code is highly commented, reading it should be fairly easy for those
with some
#           knowledge of Python and SageMath.
#           I am not an expert at SageMath or Python, the code may be less than ideal,
buyer beware.
#           Examples include:
#                   Inductors and Capacitors
#                   Potentiometer including with load on wiper
#                   Calculating parallel resistor to get a desired value, uses
equations
#                   High Pass Filter
#                   LC resonance
#
#
#
#           The worksheet can be downloaded from
#                 instructables
#
#
#
#      Intended Use:
#           Read to understand LRC and possible applications
#              or
#           Copy examples for use and modification elsewhere
#
#      Version:   Mar 03, 2014  Status: Version still under development, maybe forever
#
```

```
#                 Built and tested on Sage 5.1 Running under Virtual Box on Windows 7
#
#     Estimated Minimum Level Useful for Understanding the Worksheet
#           SageMath  - beginner
#           Electronics - basic knowledge
#
#     Possibly useful references ( Some are more advanced than the material in this
worksheet )
#           may reference later, none now
#
#     some related files:
#
#           see LRC - Instructable
#
#
#
#     Authors:  http://www.instructables.com/member/russ_hensel/  ( contact for
comments, additions, or problems )
#
```

```
# Note
# put a copy of the LRC code in the next cell and execute it to make it available to
subsequent cells
# the worksheet is saved with a copy of LRC which may or may not be the most up to
date
# the code %hideall the next cell will hide it ( including printing ) unless it has
the focus
print "hi"
```

hi

LRC defined

```
print "Use PTN, power of ten, or scientific notation"
print "10k and 20k in parallel, k = kilo = 10 to the third"
print
```

```
lrc     = LRC()
print "ignore print about frequency, we will get to it later"
print

lrc.add_parallel_r( 10e3 )   # the e here is exponent, not the number e
lrc.add_parallel_r( 20e3 )   # 20 K

print
print "Equivalent resistance = ", lrc.get_z()
#
```

```
Use PTN, power of ten, or scientific notation
10k and 20k in parallel, k = kilo = 10 to the third

LRC() using internal frequency lrc_freq in Hz
ignore print about frequency, we will get to it later

LRC.add_parallel_r() 10000.0000000000
LRC.add_parallel_r() 20000.0000000000

Equivalent resistance =  6666.66666666667
```

```
print "Now a more complicated circuit with numerical values"
# I will try the schematic in ascii characters
# lots of print statements, not necessary, just to help explain what is going on
#
print "- example of series and parallel resistor combination:"
print
print "Calculate resistance from x to x"
print
print "    |---------200-------200----------50----|"
print "x---|                                      |----------44--------x"
print "    |----------83-----------77------------|"
print

# several ways to do this, but lets make 3 resistors, one for top parallel,
# one for bottom and one for whole thing.
```

```
print "Begin..."
print "in this example we will use 3 calculators "
rTop        = LRC(  )
rBottom     = LRC(  )
rWhole      = LRC(  )


print
print "Do the top resistor 200 + 200 + 50 ohms..."


rTop.add_series_r( 200 )
rTop.add_series_r( 200 )
rTop.add_series_r( 50 )

print
print "and we have rTop"
#
print "      |---------200---------200----------50----|"
print


# repeat the output of current resistance and put in a blank line ( not at all
necessary, but for clarity )
print "rTop ", rTop.get_z()  # get_z, get the value of the resistor
print

print "Now the bottom resistor 83 + 77 ohms.."

rBottom.add_series_r( 83 )
rBottom.add_series_r( 77 )

# now we have rbottom
#
#
```

```
#
print "|----------83--------------77------------|"

# some more output
print "rBottom ",rBottom.get_z()
print

print "Now do whole thing, build the top and bottom as parallel resistor..."
# note that we have to get the values of the top and bottom resistors to put in the
whole thing.

rWhole.add_parallel_r( rTop.get_z()  )
rWhole.add_parallel_r( rBottom.get_z()  )

# now we have rWhole
#
print "|--------- 200 ---------- 200 ----------50----|"
print "|                                             |"
print "|----------83------------------77------------|"
print


# now add the final value the 44 ohm resistor in series
print "Next final value for circuit..."
rWhole.add_series_r( 44  )

print "And we have the whole thing, rWhole"
print
print "    |---------200-------200----------50----|"
print "x---|                                      |----------44--------x"
print "    |----------83------------77------------|"
print

print
# done but a final step using n()
print "Final value of combined resistance = ", rWhole.get_nz(  )
```

```
#
```

```
Now a more complicated circuit with numerical values
- example of series and parallel resistor combination:

Calculate resistance from x to x

    |---------200-------200----------50----|
x---|                                      |----------44--------x
    |----------83------------77------------|

Begin...
in this example we will use 3 calculators
LRC() using internal frequency lrc_freq in Hz
LRC() using internal frequency lrc_freq in Hz
LRC() using internal frequency lrc_freq in Hz

Do the top resistor 200 + 200 + 50 ohms...
LRC.add_series_r() 200
LRC.add_series_r() 200
LRC.add_series_r() 50

and we have rTop
      |---------200---------200----------50----|

rTop  450

Now the bottom resistor 83 + 77 ohms..
LRC.add_series_r() 83
LRC.add_series_r() 77
|----------83---------------77------------|
rBottom  160

Now do whole thing, build the top and bottom as parallel resistor...
LRC.add_parallel_r() 450
LRC.add_parallel_r() 160
|--------- 200 ---------- 200 ----------50----|
|                                             |
|----------83------------------77-------------|
```

```
Next final value for circuit...
LRC.add_series_r() 44
And we have the whole thing, rWhole

    |---------200-------200----------50----|
x---|                                      |----------44--------x
    |----------83------------77------------|


Final value of combined resistance =  162.032786885246
```

```
print "Resistance of a capacitor, more properly called Impedance, symbol Z"
print "at DC it is infinite, so lets use 440 Hz which is middle A on the musical scale"
print "This is the first time we explicitly use the frequency argument in LRC()"


lrc     = LRC( 440 )   # argument is the assumed frquency

lrc.add_parallel_c( 1e-3 )   # the e here is exponent, not the number e


print
print "impedance is: ", lrc.get_nz()      # this pushes evaluation to numeric

print
print "Notice that the impedance is a pure imaginary number, always the case for a ideal cap. or inductor"

print
print "For some purposes we want the absolute value, which is easy enough"
print "Absolute value of impedance is: ", abs( lrc.get_nz() )
```

```
Resistance of a capacitor, more properly called Impedance, symbol Z
at DC it is infinite, so lets use 440 Hz which is middle A on the
musical scale
This is the first time we explicitly use the frequency argument in
```

```
LRC()
LRC() using frequency:  440  in Hz
LRC.add_parallel_c() 0.00100000000000000

impedance is:  -0.361715779754308*I

Notice that the impedance is a pure imaginary number, always the
case for a ideal cap. or inductor

For some purposes we want the absolute value, which is easy enough
Absolute value of impedance is:  0.361715779754308
```

```
print "Impedance of a Capacitor and Resistor in series"
print "1 micro farad and 10K resistor"
print "at a frequency of 1 k Hz "
print ""
lrc     = LRC( 1e3)  # 1 k Hz
print

lrc.add_series_c( 1e-6 )   # 1 micro farad
lrc.add_series_r( 10e3 )   # 10 k ohms

print  # easy way to get a blank line

val   = lrc.get_nz()  # this pushes evaluation to numeric
print "impedance is ", val

print
print "Notice that the impedance is a complex number not purely imaginary or real"

print
val    = abs( lrc.get_nz() )
print "Absolute value of impedance is ", val
```

```
Impedance of a Capacitor and Resistor in series
1 micro farad and 10K resistor
at a frequency of 1 k Hz
```

```
LRC() using frequency:  1000.00000000000  in Hz

LRC.add_series_c() 1.00000000000000e-6
LRC.add_series_r() 10000.0000000000

impedance is  10000.0000000000 - 159.154943091895*I

Notice that the impedance is a complex number not purely imaginary
or real

Absolute value of impedance is  10001.2664346027
```

```
print "Some example calculations with a Potentiometer"
print "It is an adjustable voltage divider."
print "We will look at it that way, but then see the effect"
print "of loading the output, an effect that many ignore."
print "This is carried out in the next 4 cells"
print "   there is dependency between the cells execute from top to bottom"
print "   continue in next cell...."
```

```
Some example calculations with a Potentiometer
It is an adjustable voltage divider.
We will look at it that way, but then see the effect
of loading the output, an effect that many ignore.
This is carried out in the next 4 cells
   there is dependency between the cells execute from top to bottom
   continue in next cell....
```

```
print "Potentiometer example part 1/4 "
print "basic setup"
print

# A potentiometer is an adjustable voltage divider.
# it consists of 2 resistors in series with constant total.
# we assume it can be adjusted by an amount we will model as a value
# from 0 to 1 which we will name alpha


# the top and bottom resistors are given symbolic values
```

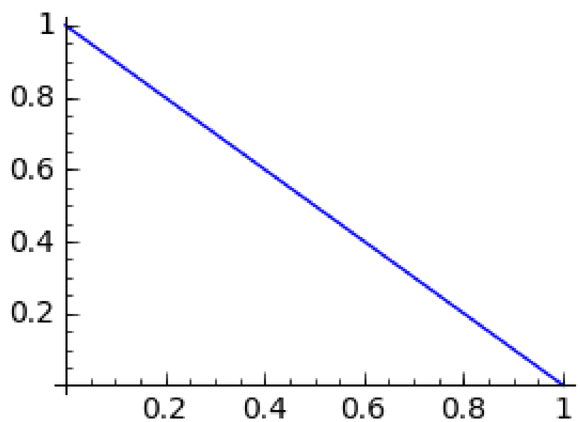
```
var( "rt" )
var( "rb" )

# just to have them handy, define 2 instances of LRC for the top and bottom
zt     = LRC()
zB     = LRC()

# here is the value we will use for the pot total, this is a 10K pot.
rtot   = 10e3

# symbolic this is the adjustment of the pot, how far it is twisted, 1/2 way is .5
var( "alpha" )

# here are the two values in terms of alpha and rtot
# this is the form for a linear taper pot
rt     = rtot * alpha
rb     = rtot * ( 1- alpha )

#print # blank line


# we use the voltage divider function
print
vout = rb/( rt + rb )
print "V out and its dependence on alpha: ", vout

# comment suppress implied print by having last line a comment.
```

```
Potentiometer example part 1/4
basic setup

LRC() using internal frequency lrc_freq in Hz
LRC() using internal frequency lrc_freq in Hz

V out and its dependence on alpha:  -alpha + 1.00000000000000
```

```
print " " # for blank line
print "Potentiometer example part 2/4"
```

```
print "graph voltage out vs alpha"
print "you can see this is a linear pot."
print

# use the SageMath plot and show functions
pt =  plot( vout, alpha, 0, 1, figsize = 3  )  # pt is plot without load
show( pt )


#
```

```
Potentiometer example part 2/4
graph voltage out vs alpha
you can see this is a linear pot.
```




```
print "Potentiometer example part 3/4"
print "now we will put a load on the output of the pot"
print

print "top leg"
rtl   = LRC()
rtl.add_series_r( rt )
```

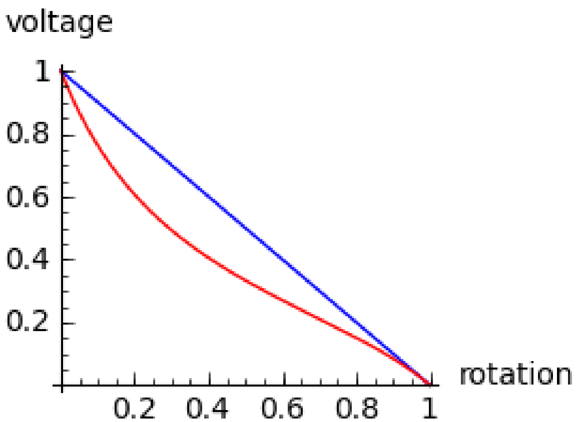
```
print
print "bottom leg"
rbl   = LRC()
rbl.add_series_r( rb )

print
print "here add the load to the bottom leg, 5K"
rbl.add_parallel_r( 5e3 )


# use the voltage divider equation
voutl   = rbl.get_z()/ ( rtl.get_z() + rbl.get_z() )

print
print "Calculated voltage divider output formula:"
show( voutl )
print "now we will plot it..."
print
print

print "Plot of voltage divider output loaded and unloaded"
print "input voltage is 1.0 volts"
pt2 = plot( voutl, alpha, 0, 1, rgbcolor="red", axes_labels=['rotation','voltage']  )
# pt2 is loaded
show( pt +  pt2 )


#
```

```
Potentiometer example part 3/4
now we will put a load on the output of the pot

top leg
LRC() using internal frequency lrc_freq in Hz
LRC.add_series_r() 10000.0000000000*alpha

bottom leg
```

```
LRC() using internal frequency lrc_freq in Hz
LRC.add_series_r() -10000.0000000000*alpha + 10000.0000000000

here add the load to the bottom leg, 5K
LRC.add_parallel_r() 5000.00000000000

Calculated voltage divider output formula:
```

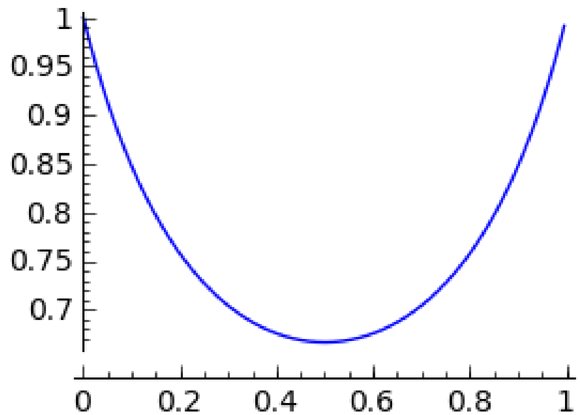
$$\frac{1}{\left(\frac{1}{-10000.0000000000\,\alpha+10000.0000000000}+0.000200000000000000\right)\left(10000.0000000000\,\alpha+\frac{1}{\frac{1}{-10000.0000000000\,\alpha+10000.0000000000}+0.000200000}\right)}$$

```
now we will plot it...


Plot of voltage divider output loaded and unloaded
input voltage is 1.0 volts
```




```
print " " # blank line
print "Potentiometer example part 4/4"

print "show error as a fraction ( 1.00 = no error ) horz. scale is alpha"

show( plot( voutl/vout,  alpha, 0, 1, figsize = 3 ) )
```

Potentiometer example part 4/4
show error as a fraction (1.00 = no error) horz. scale is alpha




```
print "How to adjust a resistor to get a desired value"
print "the value of a resistance can be lowered by putting another"
print "resistor in value"
print ""
print "Suppose we want an 11k resistor starting from a 15k resistor"
print "what do we have to add in parallel?"
print
print "We will do this using a formula, then solve it"

# parallel to get a predefined value

# variable for the two values that will be put in parallel
var( "r1" )
var( "r2" )

rdesired     = 11e3   # the resistance we want 11k

lrc          = LRC()

# put them in parallel
```

```
print
lrc.add_parallel_r( r1 )
lrc.add_parallel_r( r2 )

# set it up as an equation
equ  = ( rdesired == lrc.get_z() )

# show what the equation looks like
print
print "equation is: "
show( equ )

# substitute into the equation the given value
print "substitute for r1"
equ = equ.substitute( r1 = 15e3 )

# lets see the the equation
show( equ )

# solve the equation and print the result.
print "required resistor is: ", equ.solve( r2 )
```

```
How to adjust a resistor to get a desired value
the value of a resistance can be lowered by putting another
resistor in value

Suppose we want an 11k resistor starting from a 15k resistor
what do we have to add in parallel?

We will do this using a formula, then solve it
LRC() using internal frequency lrc_freq in Hz

LRC.add_parallel_r() r1
LRC.add_parallel_r() r2

equation is:
```

$$11000.0000000000 = \frac{1}{\frac{1}{r_1} + \frac{1}{r_2}}$$

substitute for r1

$$11000.0000000000 = \frac{1}{\frac{1}{r_2} + 0.0000666666666666667}$$

required resistor is: [
r2 == 41250
]

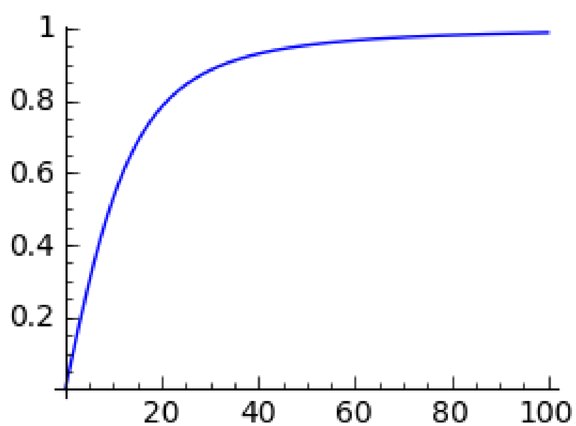
```
print "Lets try LRC for a RC voltage divider ( a high pass filter ) "
print "how long can wer print a stirng when rap Lets try LRC for a RC voltage divider
( a high pass filter ) "


lrc1 = LRC()

print
lrc1.add_series_c( 1e-6 )  # 1 micro farad


print "use voltage divider equation, lower leg 10k resistor "
vout = ( 10e3 /(  lrc1.get_z() + 10e3 ) )
show( vout )

print "linear plot"
show( plot( abs( vout ), lrc.get_freq(), .01, 1e2, figsize = 3 ) )

var( "zz" )
```

```
print "log log plot"
show( plot( log ( abs ( vout( e^zz) )), log( zz ), log( .01 ), log( 1e2 ), figsize = 3
) )


#
```

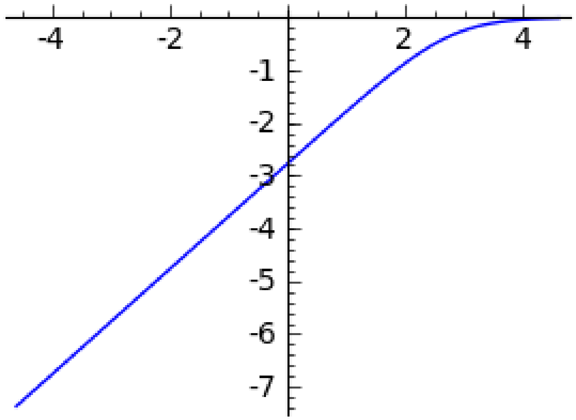
```
Lets try LRC for a RC voltage divider ( a high pass filter )
how long can wer print a stirng when rap Lets try LRC for a RC
voltage divider ( a high pass filter )
LRC() using internal frequency lrc_freq in Hz

LRC.add_series_c() 1.00000000000000e-6
use voltage divider equation, lower leg 10k resistor
```

$$\frac{10000.0000000000}{-\frac{500000.000000000i}{\pi \mathrm{lrc}_{\mathrm{freq}}} + 10000.0000000000}$$

```
linear plot
```




```
log log plot
__main__:1: DeprecationWarning: Substitution using function-call
syntax and unnamed arguments is deprecated and will be removed from
a future release of Sage; you can use named arguments instead, like
EXPR(x=..., y=...)
```




```
print "Use LRC for resonate circuits"

print "first we will look into an ideal LC series resonate circuit - no resistance"
print

# lets take some values of components that are reasonably available

l       = 1e-3        # this is 3 mili henris inductor, note we are using scientific
notation for the value
c       = 1e-6        # this is a 1 micro farad capacitor




# estimate of the resonate frequency standard equation
#
res     = 1/(sqrt(l*c))


# these are sagemath variables which we will use to work with symbolic properties
# around the resonance, you will see how they are used later
var ( "omega" )    # omega will be used for the angular frequency of the AC in the
circuit
```

```
# make a component for the computation.
imp1    = LRC( omega, use_angular = True  )



# now put in the LC values
imp1.add_series_c( c )
imp1.add_series_l( l )


# get the computed value of impedance, it will be a complex value and will of
# course depend on omega -- that is the answer will look more or like a formula
print "get_z()", imp1.get_z()
print # blank line

print "get_z()", imp1.get_z()
print # blank line

# now do a plot of the function, we can see the impedance
# goes to 0 at the resonate frequency
imp1.plot( res/4, res * 4  )

#
```

```
Use LRC for resonate circuits
first we will look into an ideal LC series resonate circuit - no
resistance

LRC() using frequency:  omega  angular in radians/sec
LRC.add_series_c() 1.00000000000000e-6
LRC.add_series_l() 0.00100000000000000
get_z() 0.00100000000000000*I*omega - 1.00000000000000e6*I/omega

get_z() 0.00100000000000000*I*omega - 1.00000000000000e6*I/omega


LRC Plot, abs( z ) vs frequency
```

$$\left| 0.00100000000000000 i\,\omega - \frac{1.00000000000000 \times 10^{6} i}{\omega} \right|$$