RF Notebook

March 23, 2016

1 RF Notebook

This file contains RF specifications, calculations and tests.

In [15]:
```
import sys
sys.path.append("/Users/benno/Dropbox/Software/python/")
sys.path.append("C:/Users/bm1u11/Dropbox/Software/python/")
import gammaList as gL
from scipy.constants import mu_0
from numpy import min, max, pi, sqrt, array, real, imag, angle, ones, linspace, logspace, exp,
%matplotlib inline
import matplotlib.pyplot as plt
def piHalf(Q, P, nuc="1H", coil="1H"):
    if coil == "1H":
        V = 15e-3*pi*(13.5e-3/2)**2
        omega = 2*pi*285.4e6

    B1 = sqrt(mu_0*Q*P/(2*omega*V))
    print "B1 is: {} mT".format(B1*1000)

    gamma = gL.gammaList(nuc)
    omega1 = gamma*B1
    tauPiHalf = pi/(2*omega1)
    print "Pi/2 pulse is {} us".format(tauPiHalf*1e6)

piHalf(86, 30, nuc="13C")
```

```
B1 is: 0.648871246856 mT
Pi/2 pulse is 35.9796581287 us
```

In [2]:
```
print "Carbon frequency: {} MHz".format(285.4*gL.gammaList("13C")/gL.gammaList("1H"))
```

```
Carbon frequency: 71.7791932935 MHz
```

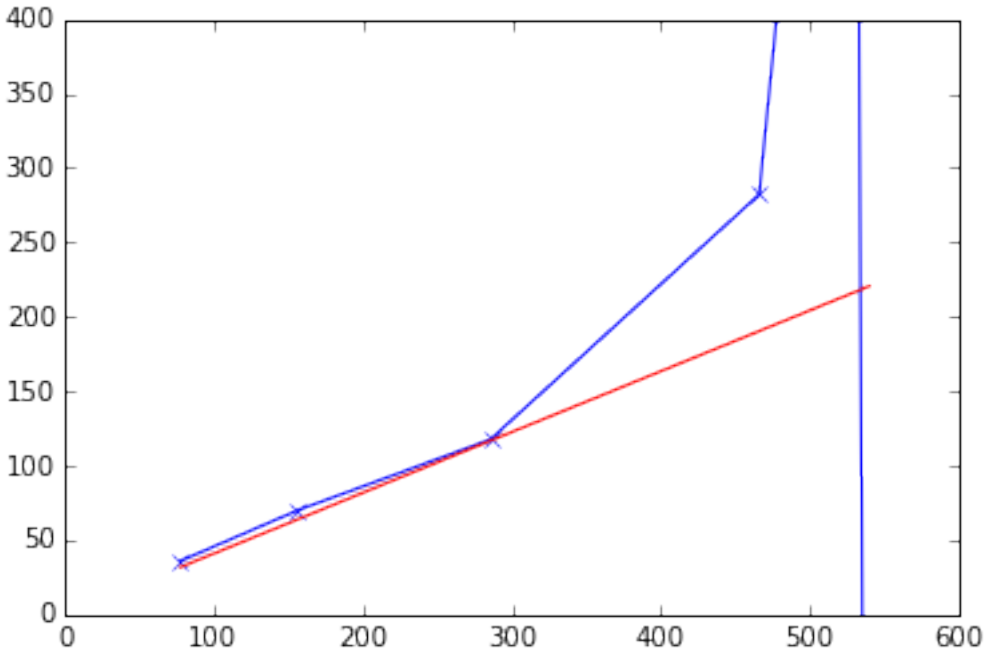
In [3]:
```
wC = 2*pi*71.779e6
wH = 2*pi*285.4e6

print "Reactance Carbon from {0} to {1} Ohm".format(1./(wC*1.5e-12), 1./(wC*45e-12))
print "Reactance Hydrogen from {0} to {1} Ohm".format(1./(wH*1.5e-12), 1./(wH*45e-12))
print "Ratio wC/wH: {0}".format(wC/wH)

print "Reactance coil : {0} Ohm".format(wH*60e-9)

#To bring the values down (especially for Carbon) one needs to add capacitance!
```

```
Reactance Carbon from 1478.1941152 to 49.2731371732 Ohm
Reactance Hydrogen from 371.770481411 to 12.3923493804 Ohm
Ratio wC/wH: 0.251503153469
Reactance coil : 107.5932652 Ohm
```

Coil dimensions: Milled to -1 with 2 mm cutter on 16.2 mm diameter PTFE. X from 0 to 18.5 => h = 18.5, d = 15 mm Coil measured with NVA after calibration, ground leg soldered to cavity support with cavity mounted:

```
In [4]: f = array([77.5, 156.4, 285.6, 466, 530, 540])*1e6
        realZ = array([0.4, 0.8, 2, 15.8, 1400, 1100])
        imagZ = array([35.9, 70, 117.5, 282, 930, -850])

        plt.plot(f/1e6, imagZ, "-x")
        plt.plot(f/1e6, 2*pi*f*65e-9, "-r")
        plt.ylim([0, 400])

        print "The inductance of the coil is 65 nH"
        print "The reactance at 13C res frequency hence is ", wC*65e-9
        print "The resistance is 0.5 Ohm (measured)"
```

```
The inductance of the coil is 65 nH
The reactance at 13C res frequency hence is  29.3150492807
The resistance is 0.5 Ohm (measured)
```

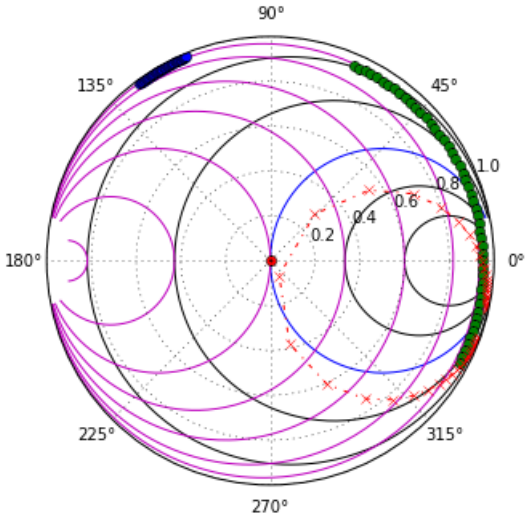


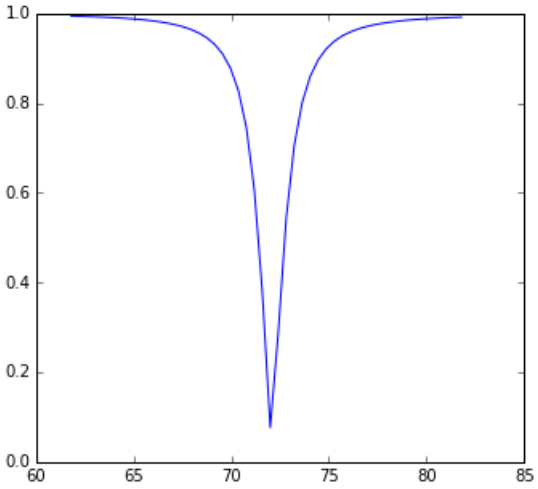

```
In [7]: def impedanceToGamma(impedance, Z0 = 50):
            return (impedance - Z0)/(impedance + Z0)

        def addConstantResistance(plot, Z0 = 50, colour="-k"):
            for r in [5,20,50,100,200]:
```

```
        Z = r*ones(500) + 1j*linspace(-500,500, num=500)
        gamma = impedanceToGamma(Z)
        if r == 50:
            plt.plot(angle(gamma), abs(gamma), "-b")
        else:
            plt.plot(angle(gamma), abs(gamma), colour)

def addConstantReactance(plot, Z0 = 50, colour="-r"):
    for XL in [-200,-100,-50,-20,5,0,5,20,50,100,200]:
        Z = linspace(0.1,500, num=500) + 1j*XL*ones(500)
        gamma = impedanceToGamma(Z)
        plt.plot(angle(gamma), abs(gamma), colour)

def addConstantConductance(plot, Z0 = 50, colour = "-m"):
    for g in [1./1500, 1./500, 1./200, 1./100, 1./50, 1./20, 1./5]:
        B = linspace(-1./5,1./5, num = 500)
        Y = g + 1j*B
        gamma = impedanceToGamma(1/Y)
        plt.plot(angle(gamma), abs(gamma), colour)

def addMatchingAndTuningRange(plot, nuc="13C", colour="b.", Xmin = 50):
    if nuc == "13C":
        Xmax = 500
        Xmin = Xmin

    reactanceRange = linspace(Xmin, Xmax)

    #print reactanceRange
    Z0 = 50

    Zlist = []
    for Xm in reactanceRange:
        for Xt in reactanceRange:
            Z = Z0 + 1j*Xm
            Z = Z*(1j)*Xt/(Z + (1j)*Xt)
            Zlist.append(Z)

    Zlist = array(Zlist)
    #print Zlist
    gammaList = (Zlist - Z0)/(Zlist + Z0)
    #print gammaList
    plt.plot(angle(gammaList), abs(gammaList), colour)



fig = plt.figure(figsize=(12,5), facecolor='white')
axes = plt.subplot(121, polar=True, axisbg='white')
#axes.spines['polar'].set_visible(False)
#plt.yticks([])
#plt.xticks([])


addConstantResistance(plt)
#addConstantReactance(plt, colour = "-gray")
```

```
addConstantConductance(plt)


#addMatchingAndTuningRange(plt, Xmin=5, colour="b.")
#addMatchingAndTuningRange(plt, Xmin=50, colour="r.")

plt.xlim([0,1])
#plt.grid(False)
#plt.polar?

f = 71.8e6
f = linspace(61.8e6, 81.8e6)
omega = 2*pi*f


Z0 = 50
L = 65e-9
# coil
coilImpedance = 0.5+1j*omega*L
gamma_coil = impedanceToGamma(coilImpedance)
#print "Gamma coil:", gamma_coil
plt.plot(angle(gamma_coil), abs(gamma_coil), "o")
plt.plot(0, 0, "ro")

#now add Shunt Capacitor
CT = 68e-12# + 15.2e-12
ZCT = -1j/(omega*CT)
Z = coilImpedance*ZCT/(coilImpedance+ZCT)
#print "Z after Shunt Capacitance", Z
gamma_shunt = impedanceToGamma(Z)
#print "Gamma: ", gamma_shunt
plt.plot(angle(gamma_shunt), abs(gamma_shunt), "o")

#now add Matching Capacitor
CM = 7.2e-12
ZCM = -1j/(omega*CM)
Z = Z + ZCM
#print "Z after Series Capacitance", Z
gamma_series = impedanceToGamma(Z)
#print "Gamma: ", gamma_series
plt.plot(angle(gamma_series), abs(gamma_series), "-.x")

#print "Resonance Frequency:", 1/sqrt(L*CT)/(2*pi)/1e6

plt.subplot(122)

plt.plot(f/1e6, abs(gamma_series))
```

Out[7]: [<matplotlib.lines.Line2D at 0x106cc4a90>]






```
In [11]: print "Coil: ", min(abs(gamma_coil))
         print "Gamma Matched and Tuned: ", impedanceToGamma(42 + 1j*90)
         print "Matched and Tuned: ", abs(impedanceToGamma(42+1j*90))
         print "Matched and Tuned: ", abs(impedanceToGamma(10-1j*120))
```

```
Coil:  0.984187652971
Gamma Matched and Tuned:  (0.444578604202+0.543347017629j)
Matched and Tuned:  0.70205136342
Matched and Tuned:  0.942809041582
```

```
In [30]: def powerEfficiency(alpha,L,Gamma):
             """Power Efficiency as given by Kodibagkar, Conradi"""
             return exp(2*alpha*L)*(1 - abs(Gamma)**2)/(exp(4*alpha*L) - abs(Gamma)**2)
```

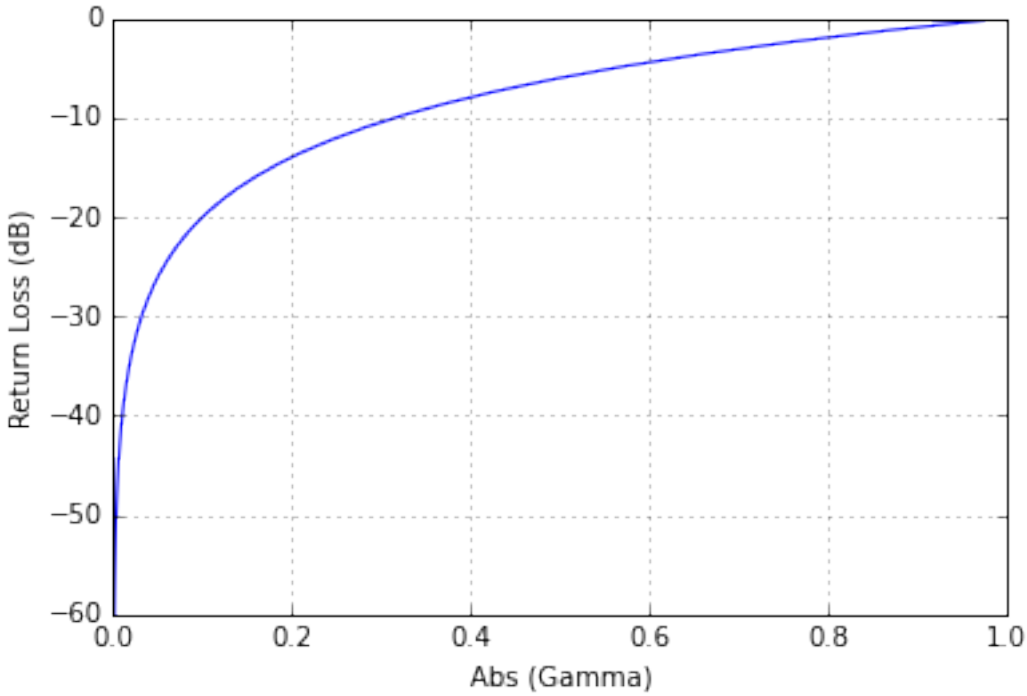
```
In [31]: powerEfficiency(0.1, 1, 0.95)
```

```
Out[31]: 0.20207327029946784
```

```
In [94]: cableLength = 1.46 # cable length in m
         #cable Type: AN50085, 45dB / 100m => 1m = 0.45 dB
         10**(-0.045)
```

```
Out[94]: 0.9015711376059569
```

```
In [24]: absGamma = logspace(-3, 0)
         rL = 20*log10(absGamma)
         plt.plot(absGamma, rL)
         plt.xlabel("Abs (Gamma)")
         plt.ylabel("Return Loss (dB)")
         plt.grid()
         plt.savefig("returnLossVsGamma.pdf")
```



In [27]:
```
#alpha may be extracted from the following: for 1H with nothing attached to the semirigid abs(
#For 13C with nothing attached: abs(Gamma) =  0.96
#This should be exp(-alpha*2*L)
```

In [ ]: