LH-Hands-on-mode-converter

February 10, 2016

1 Hands-on LH1: the TE_{10}-TE_{30} Mode Converter

1.1 Introduction

The Tore Supra Lower Hybrid Launchers are equiped by TE_{10}-TE_{30} mode converters, a waveguide structure which convert the RF power from a propagation mode to another, in order to split the power by three in the poloidal direction. The electric field topology in this device is illustrated in the following figure.

During this hands-on, students are initiated to RF measurements and analysis. Before measuring the RF performances of a mode converter, the students are brought to calibrate the RF measurement apparatus, a (Vectorial) Network Analyser. This device measures the scattering parameters (or S-parameters) between two ports, generally referred as ports 1 and 2. The scattering parameters are defined in terms of incident and reflected waves at ports, and are widely used at microwave frequencies where it becomes difficult to measure voltages and currents directly.

Before starting, let's import the necessary Python libraries:

```
In [1]: # This line configures matplotlib to show figures embedded in the notebook,
        # and also import the numpy library
        %pylab
        %matplotlib inline
```

```
Using matplotlib backend: TkAgg
Populating the interactive namespace from numpy and matplotlib
```

1.2 Scattering parameters measurement

The following figure illustrates the measurement setup, and the adopted port indexing convention.

Below we import the measurements data generated by the network analyser. These data consist in an ASCII file (tab separated file), with the following usual header : ! Network Analyzer HP8720ES.07.62: Oct 06, 1999 Serial No. US39172126 ! Hewlett-Packard ! 10 Feb 2011 10:37:32 ! Frequency S11 S21 S12 S22 3600000000 -8.5072 -37.511 -5.4372 105.59 -5.4692 105.56 -6.405 48.978 3601000000 -8.5247 -40.428 -5.4388 102.22 -5.4899 102.31 -6.4298 42.946 3602000000 -8.5488 -43.451 -5.4724 98.785 -5.4869 98.979 -6.4618 37.193 where we have, by column number:

1. The frequency in Hertz,
2. The amplitude of the S_{11} parameter in decibel (dB),
3. The phase of the S_{11} parameter in degree,
4. The amplitude of the S_{21} parameter in decibel (dB),
5. The phase of the S_{21} parameter in degree,

and etc for S_{12} and S_{22}.

Below we import the first measurement data, performed between the port indexed "0" of the mode converter (the input, corresponding to the port 1 of the network analyser) and the port indexed "1" of the mode converter (corresponding to port 2 of the network analyser).

```
In [2]: %cd C:\Users\JH218595\Documents\Notebooks\TP Master Fusion
```

```
C:\Users\JH218595\Documents\Notebooks\TP Master Fusion
```

```
In [3]: CDM1 = np.loadtxt('./LH1_Mode-Converter_data/CDM01', skiprows=4)
        f_1 = CDM1[:,0]
```

The measurement contains N frequency points, where N is:

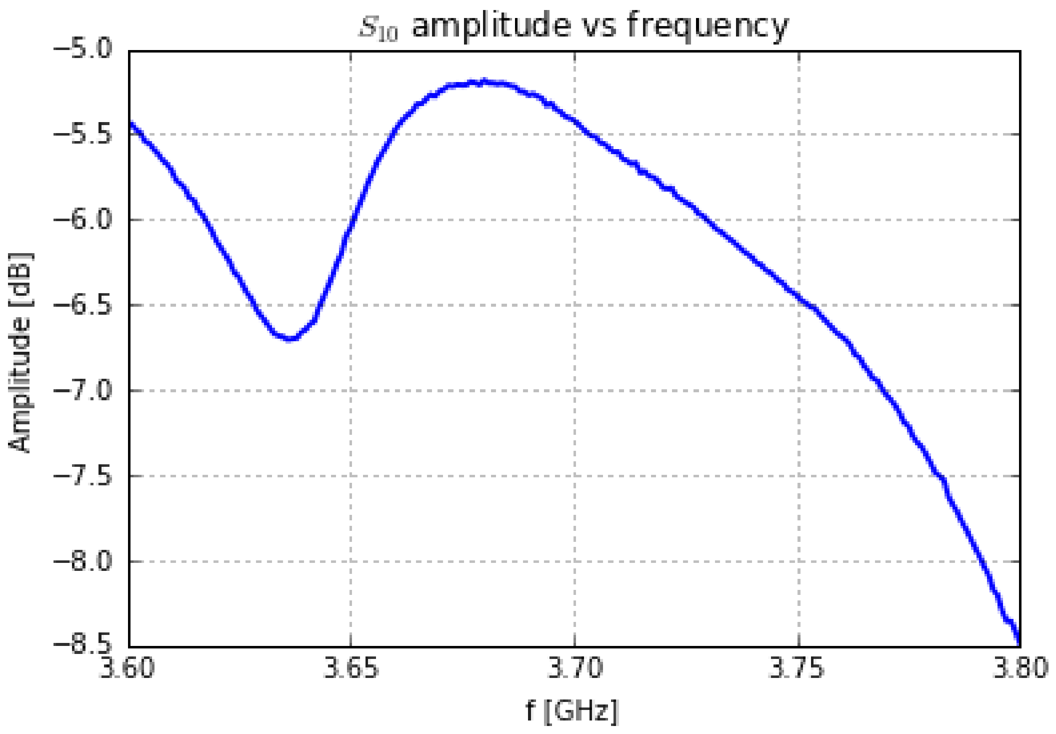
```
In [4]: len(f_1)
```

```
Out[4]: 201
```

Let's have a look to these data, for example by plotting the amplitude of the S_{10} parameter, that is the ratio of the power coming from port 0 to port 1 (which corresponds to S_{21} in the network analyser file).

```
In [5]: S10_dB = CDM1[:,3]
        plot(f_1/1e9,S10_dB, lw=2)
        xlabel('f [GHz]')
        ylabel('Amplitude [dB]')
        grid('on')
        title('$S_{10}$ amplitude vs frequency')
```

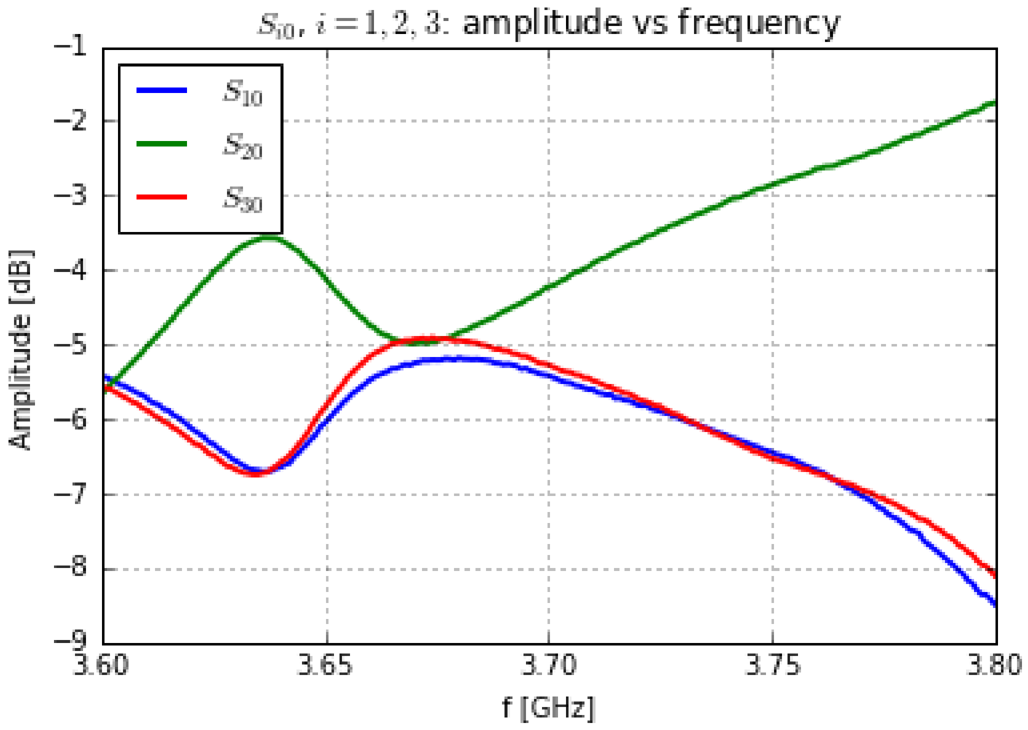
```
Out[5]: <matplotlib.text.Text at 0x733e7f0>
```



OK. Let's do the same for the second and third measurements performed, that is for the power transferred from port 0 to ports 2 and 3.

```
In [6]: CDM2 = loadtxt('LH1_Mode-Converter_data/CDM02', skiprows=4)
        CDM3 = loadtxt('LH1_Mode-Converter_data/CDM03', skiprows=4)

        f_2 = CDM2[:,0]
        f_3 = CDM3[:,0]

        S20_dB = CDM2[:,3]
        S30_dB = CDM3[:,3]
        S00_dB = CDM1[:,1]

        plot(f_1/1e9, S10_dB, f_2/1e9, S20_dB, f_3/1e9, S30_dB, lw=2)
        xlabel('f [GHz]')
        ylabel('Amplitude [dB]')
        grid('on')
        title('$S_{i0}$, $i=1,2,3$: amplitude vs frequency')
        legend(('$S_{10}$','$S_{20}$', '$S_{30}$'),loc='best')
```

```
Out[6]: <matplotlib.legend.Legend at 0x760c198>
```



Nice. Now, let's stop and think. The purpose of the mode converter is the transfer the power from the fundamental mode of rectangular waveguides, namely the TE_{10}, into a higher order mode, the TE_{30}. Once these mode conversion achieved, thin metallic wall septum are located in zero E-field regions, which allows to split the power into three independant waveguides.

Dividing the power by 3, is equivalent in decibels to:

```
In [7]: 10*log10(1.0/3.0)
```

```
Out[7]: -4.7712125471966242
```

Thus, ideally, the three transmission scattering parameters should be equal to -4.77 dB at the operational frequency, 3.7 GHz in our case. Clearly from the previous figure we see that it is not the case. The power splitting is unbalanced, and more power is directed to port 2 than to ports 1 and 3 at 3.7 GHz. In conclusion of this first serie of measurements: this mode converter is not working properly. The big deal is: “why ?”.

Before continuying, it may be usefull to define a function that convert a (dB,degree) number into a natural complex number:

```
In [8]: def dBdegree_2_natural(ampl_dB, phase_deg):
            amp = 10**(ampl_dB/20)
            phase_rad = np.pi/180*phase_deg
            return amp*np.exp(1j*phase_rad)
```

Thus, we can convert the (dB,degree) data into natural (real,imaginary) numbers

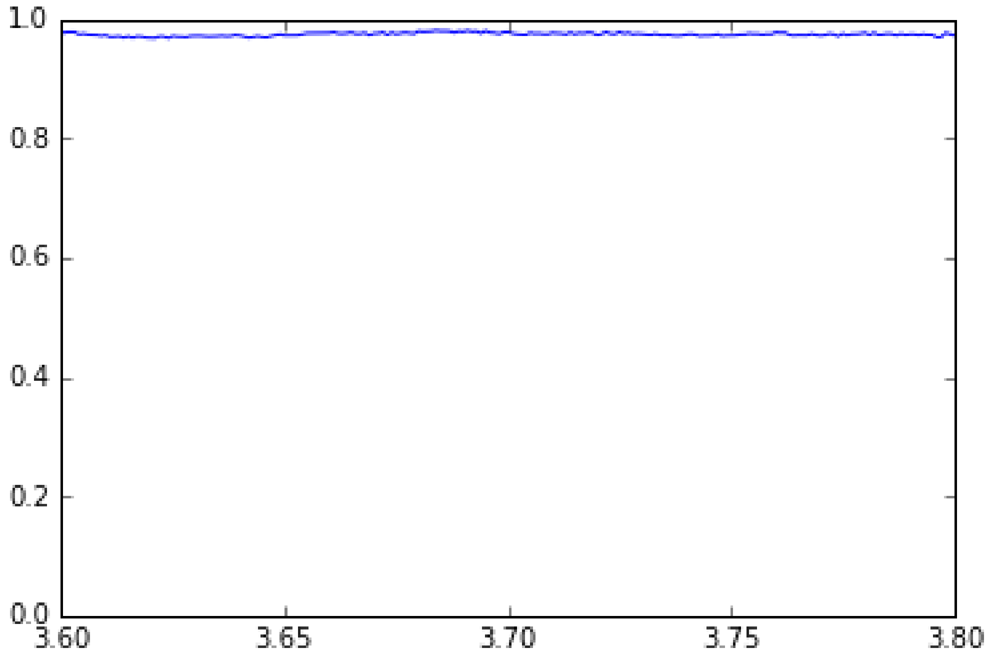
```
In [9]: S00 = dBdegree_2_natural(CDM1[:,1],CDM1[:,2])
        S10 = dBdegree_2_natural(CDM1[:,3],CDM1[:,4])
        S20 = dBdegree_2_natural(CDM2[:,3],CDM2[:,4])
        S30 = dBdegree_2_natural(CDM3[:,3],CDM3[:,4])
```

Let’s check the power conservation. If the calibration has been correctly performed and if the conduction losses are negligible, one has:

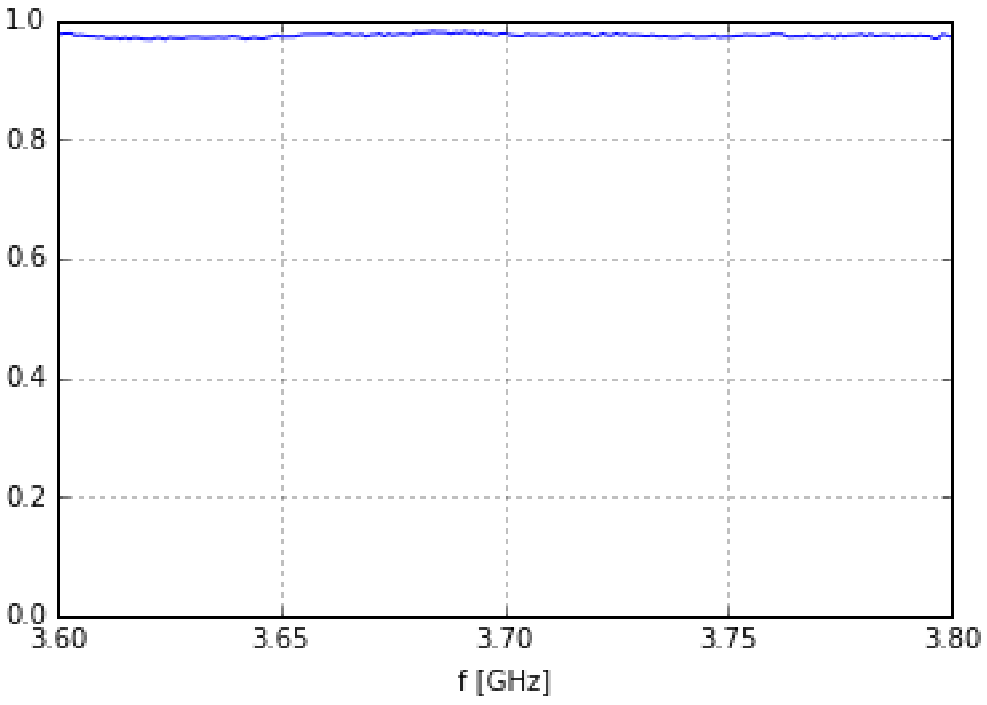
$$\sum_{i=0\ldots3} |S_{j0}|^2 = 1$$

Let’s try:

```
In [10]: # Check the power conservation in dB : reflected+incident = all the power
         plot(f_1/1e9, 10**(S00_dB/10)+10**(S10_dB/10)+10**(S20_dB/10)+10**(S30_dB/10))
         ylim([0, 1])

Out[10]: (0, 1)
```

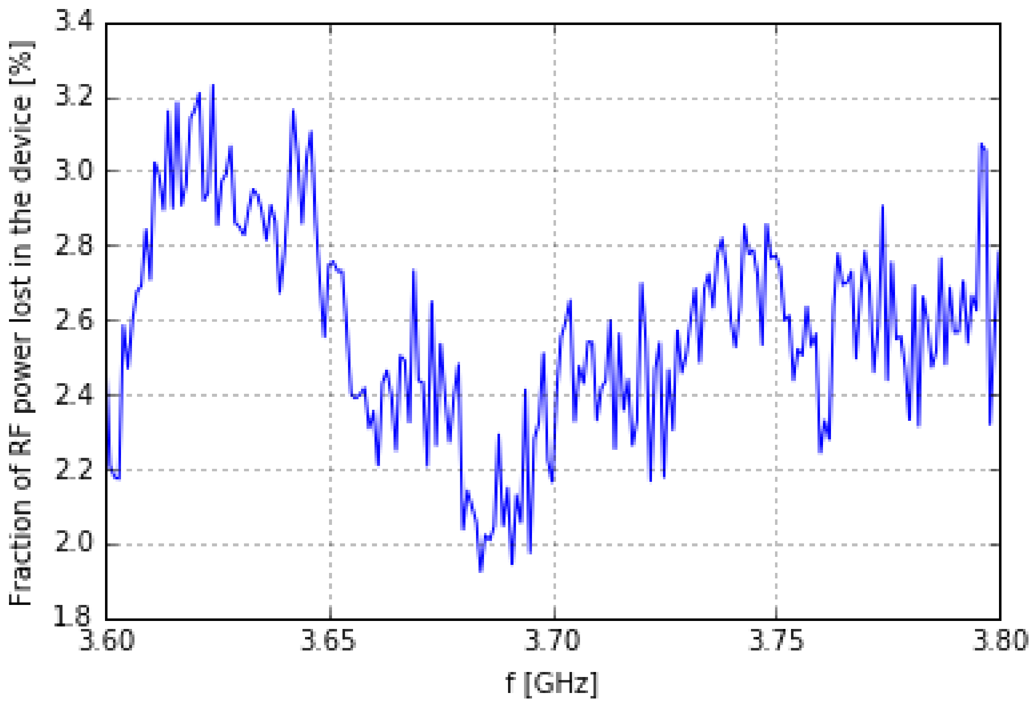


```
In [11]: plot(f_1/1e9, sum([abs(S00)**2,abs(S10)**2,abs(S20)**2,abs(S30)**2],axis=0))
         xlabel('f [GHz]')
         grid('on')
         ylim([0, 1])

Out[11]: (0, 1)
```



We are close to 1. The difference is the conduction losses, but also the intrinsic measurement error. Let's see how much power is lost in the device in terms of percent :

```
In [12]: plot(f_1/1e9, 100*(1-sum([abs(S00)**2,abs(S10)**2,abs(S20)**2,abs(S30)**2],axis=0)))
         xlabel('f [GHz]')
         ylabel('Fraction of RF power lost in the device [%]')
         grid('on')
```



1.3 Electric field measurements

In the previous section we figured out that the mode converter was not working properly as a 3-way splitter; indeed the power splitting is unbalanced. It's now time to understand why, since the guy who performed the RF modeling of the stucture is very good and is sure that his design is correct. In order to dig into this problem, we propose to probe the electric field after the mode converter but before the thin septum splitter: the objective is to "see" what is the electric field topology after the mode conversion (is it as we expect it should be?). But by the way, how should it be?

If the mode converter has performed well, the totality of the electromagnetic field should behave as a TE_{30} mode.

```
In [13]: # Electric field measurement
         # Measurement Hands-On Group #1
         # Feb 2013
         # columns correspond to hole rows 1,2,3
         ampl_dB = -1.0*np.array([           # use a numpy array in order to avoid the caveat to multip
                     [31.3, 32.0, 31.2],
                     [30.6, 31.2, 30.2],
                     [33.2, 33.3, 32.5],
                     [42.8, 42.0, 41.7],
                     [40.1, 41.3, 40.6],
                     [32.7, 33.5, 32.7],
                     [31.2, 31.4, 30.9],
                     [32.5, 33.0, 32.8],
                     [39.1, 39.9, 40.9],
                     [46.6, 44.0, 42.0],
                     [34.8, 33.2, 33.2],
                     [32.4, 30.6, 30.6],
                     [33.3, 30.9, 32.0]])

       phase_deg = np.array(
                   [[-111.7,  64.9, -119.2],
                    [-111.9,  67.2, -119.9],
                    [-110.7,  67.1, -119.8],
                    [-99.1,   75.7, -119.7],
                    [55.90, -127.4,   60.1],
                    [60.4,  -123.3,   59.2],
                    [64.3,  -117.4,   62.8],
                    [61.3,  -119.7,   63.3],
                    [50.9,  -118.3,   68.1],
                    [-95.8,   70.5, -120.9],
                    [-107.9,  74.1, -116.0],
                    [-112.0,  71.2, -112.6],
                    [-116.6,  72.4, -116.0]])
```

Let's check if we have the same number of points for amplitude and phase:

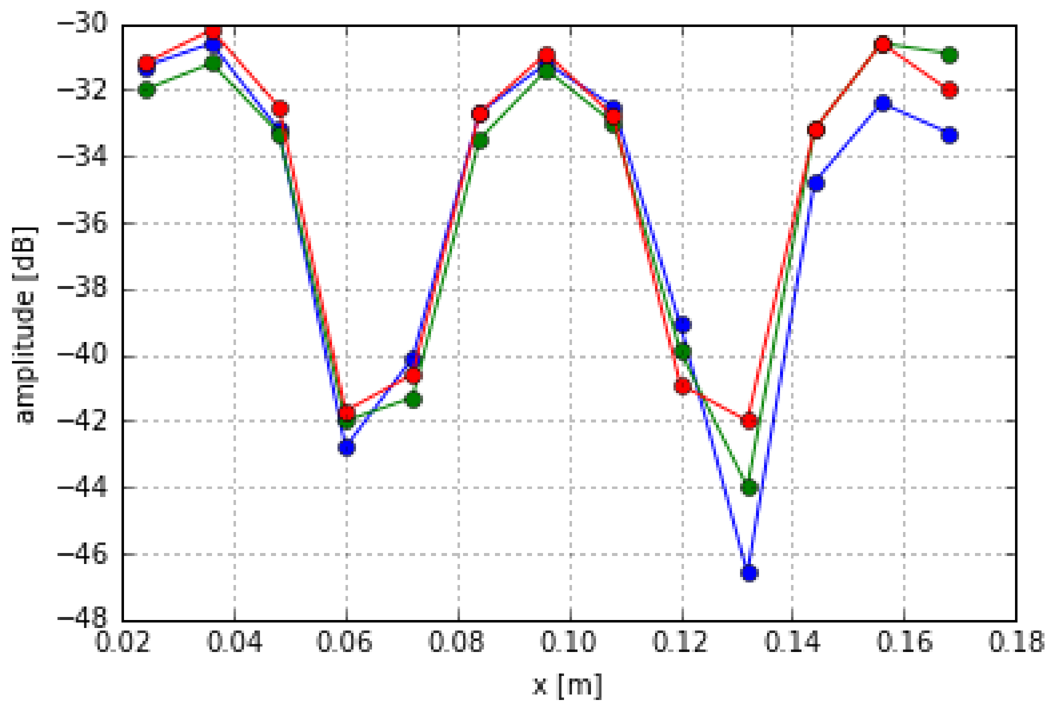
```
In [14]: shape(ampl_dB) == shape(phase_deg)
```

```
Out[14]: True
```

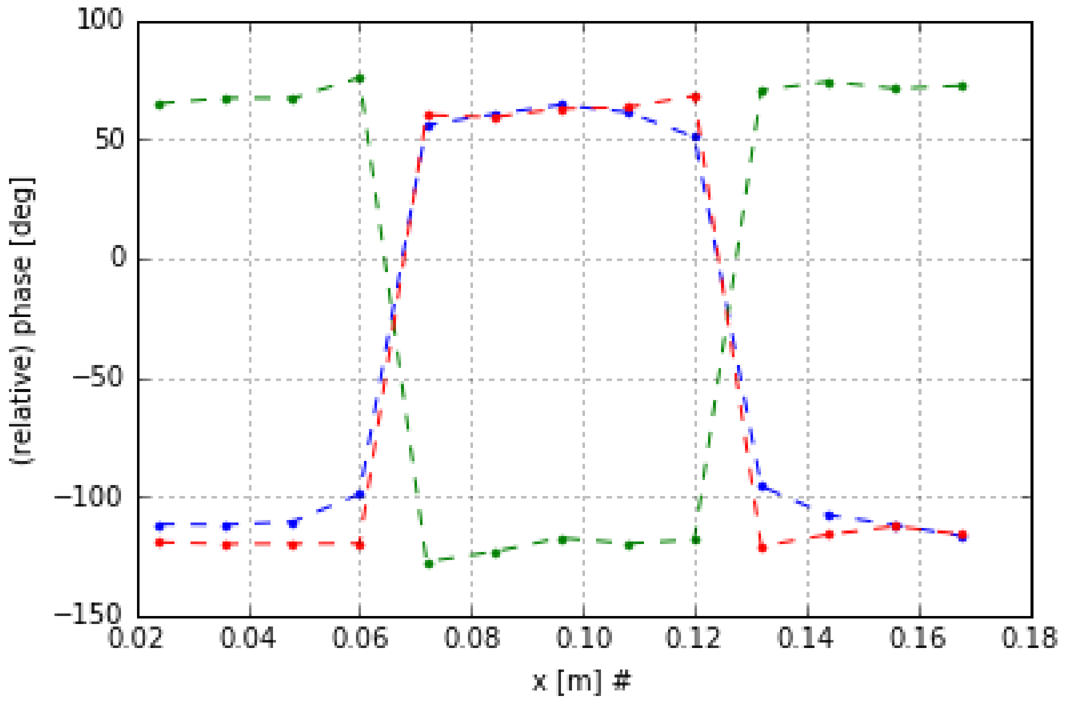
What does it look like?

```
In [15]: f = 3.7e9 # fréquence de la mesure
         a = 192e-3 # largeur grand coté du guide surdimensionné
         b = 40e-3

         ## On ne prend que N mesures aux N abscisses de mesures
         # définition des emplacements des N abscisses :
         x_mes = 24e-3 + arange(0, 12*12e-3, step=12e-3) # Modele @3.7GHz d'après S.Berio
         x = linspace(0, a, 100) # theorical values

         ## Mesures tirées de la thèse de S.Berio p.30
         # definition des emplacements des 3 séries de "mesures"
         z_mes = [0, 28*1e-3, 100*1e-3]
```

```
In [16]: plot(x_mes, ampl_dB, '-o')
         xlabel('x [m]')
         ylabel('amplitude [dB]')
         grid()
```



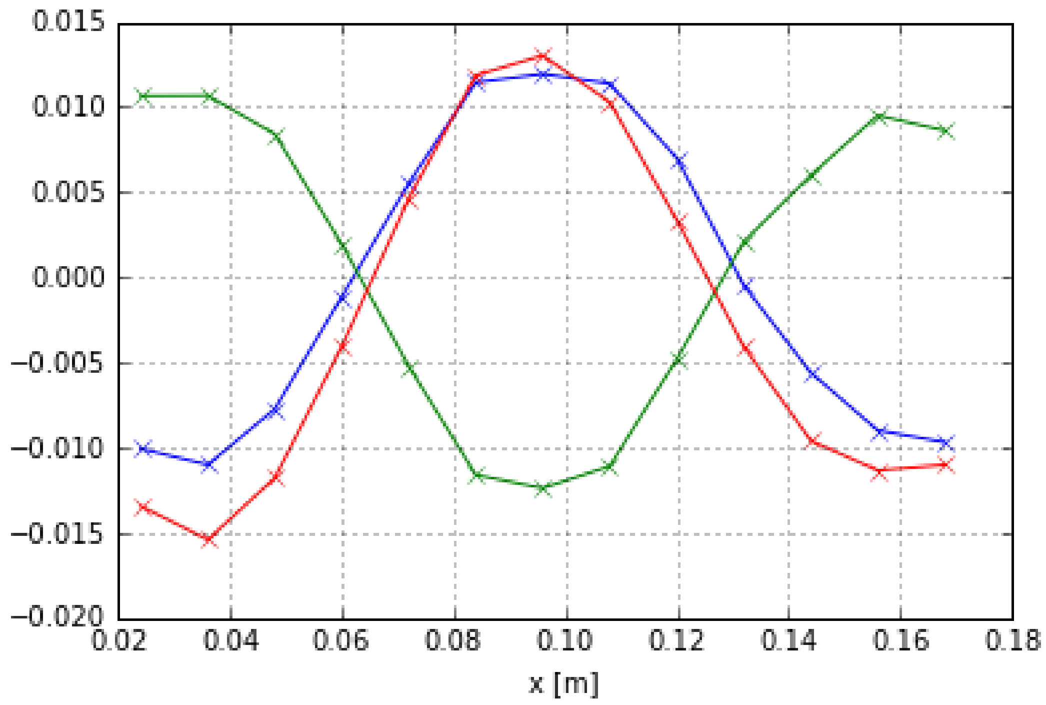
```
In [17]: plot(x_mes, phase_deg, '--.')
         xlabel('x [m] #')
         ylabel('(relative) phase [deg]')
         grid()
```



In natural values

```
In [18]: measures = dBdegree_2_natural(ampl_dB, phase_deg)
```

```
In [19]: plot(x_mes, real(measures),'-x')
         xlabel('x [m]')
         grid('on')
```



From the previous amplitude figures, one can remark that the measureed amplitude is not ideal: the maxima and the minima seem not have exactly the same values. Thus is seems that the mode after the mode converter is not a pure TE$_{30}$, but probably a mixture of various modes. The question is : what is that mixture of modes? Our objective is to deduce that from the Efield probe data.

1.4 Electromagnetic model

Let's define first some usefull functions:

The following function calculates the electric field at a point (x, z) of the waveguide of shape (a, b) for N modes:

$$E_y(x,z) = \sum_{n=1}^{N} a_n \sin\left(\frac{n\pi}{a}x\right) e^{-j\beta_n z}$$

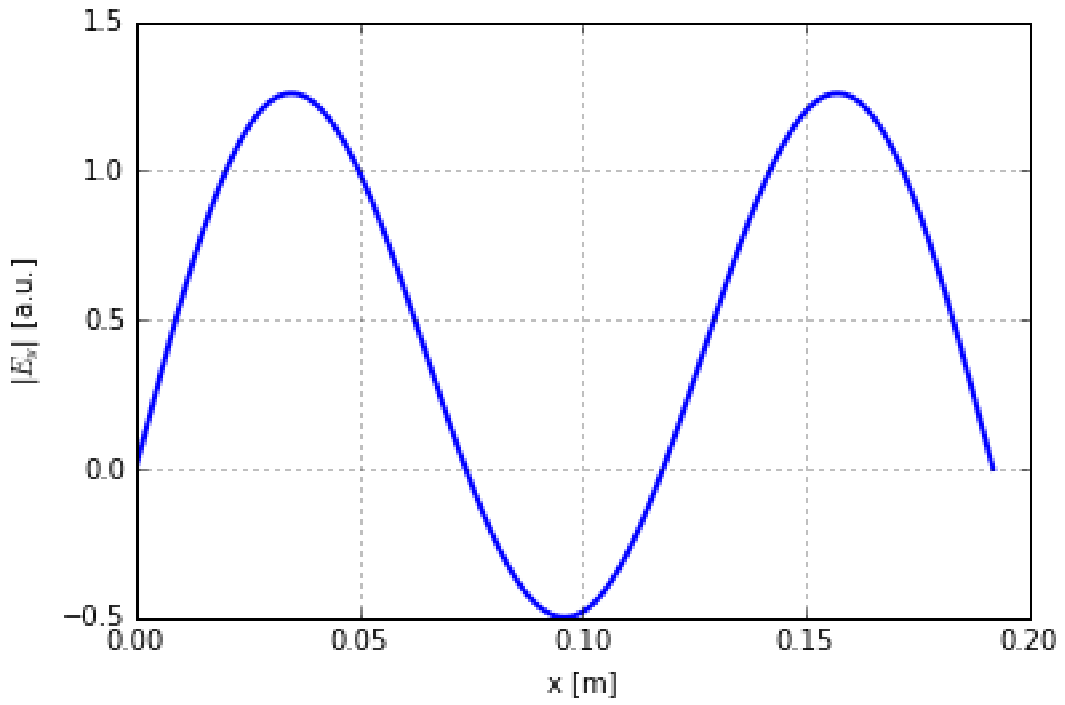
where

$$\beta_n = \sqrt{k_0^2 - \left(\frac{n\pi}{a}\right)^2}$$

```
In [20]: from scipy.constants import c

         def Ey(a_n, x, z, wg_a=192e-3, f=3.7e9):
```

```
        '''
        Evaluates the electric field at the (x,z) location
        x and z should be scalars
        '''

        k0 = 2*pi*f/c

        sin_n = np.zeros_like(a_n, dtype='complex')
        beta_n = np.zeros_like(a_n, dtype='complex')
        exp_n = np.zeros_like(a_n, dtype='complex')
        exp_n2 = np.zeros_like(a_n, dtype='complex')

        N_modes = len(a_n)

        for n in np.arange(N_modes):
            # Guided wavenumber
            # use a negative imaginary part for the square root
            # in order to insure the convergence of the exponential
            if k0**2 - ((n+1)*pi/wg_a)**2 >= 0:
                beta_n[n] = np.sqrt(k0**2 - ((n+1)*pi/wg_a)**2)
            else:
                beta_n[n] = -1j*np.sqrt(((n+1)*pi/wg_a)**2 - k0**2)

            exp_n[n] = np.exp(-1j*beta_n[n]*z)

            sin_n[n] = np.sin((n+1)*pi/wg_a*x)

        # sum of the modes
        Ey = np.sum(a_n*sin_n*exp_n)

        return Ey
```

```
In [21]: u_n = np.array([0.5, 0, 1,])
         x_test = linspace(0, 192e-3, 201)
         z_test = zeros_like(x_test)
         E_test = zeros_like(x_test, dtype='complex')
         for idx in range(len(x_test)):
             E_test[idx] = Ey(u_n, x_test[idx], z_test[idx])

         plot(x_test, real(E_test), lw=2)
         xlabel('x [m]')
         ylabel('|$E_y$| [a.u.]')
         grid()
```



1.5 Least Square solving with Python scipy routines

```
In [22]: print(x_mes)
```

```
[ 0.024  0.036  0.048  0.06   0.072  0.084  0.096  0.108  0.12   0.132
  0.144  0.156  0.168]
```

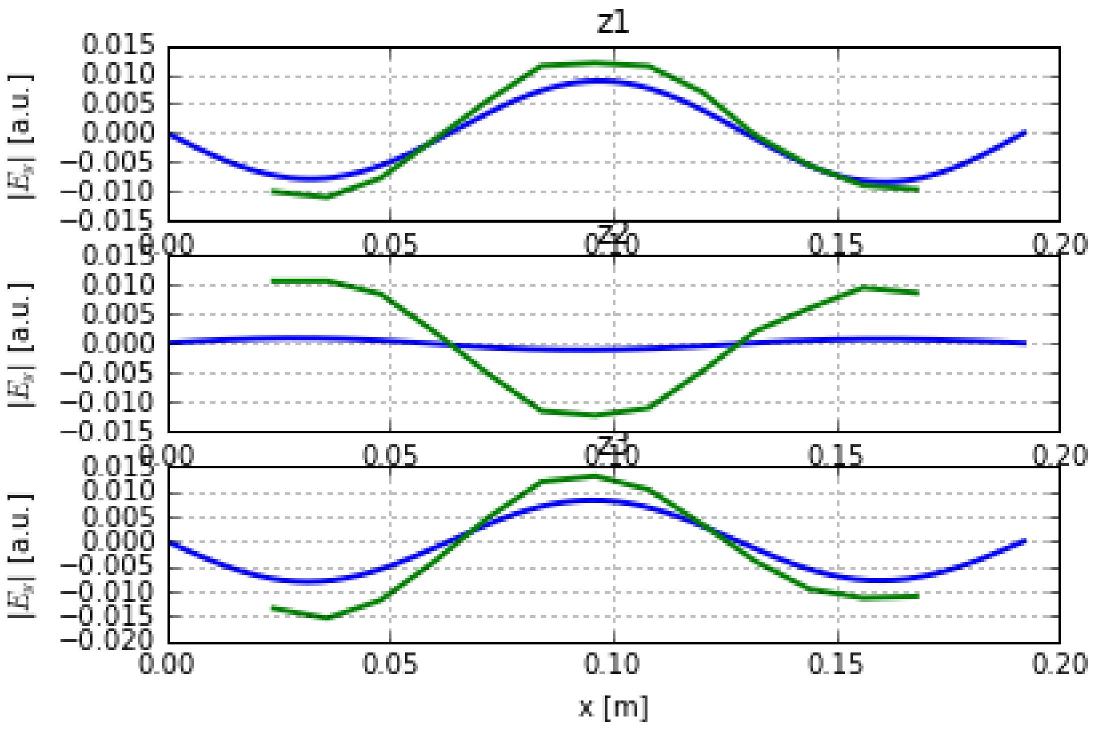
```
In [23]: print(z_mes)
```

```
[0, 0.028, 0.1]
```

```
In [24]: Emeas = dBdegree_2_natural(ampl_dB, phase_deg)
         print(Emeas)
```

```
[[-0.01006710-0.0252975j   0.01065541+0.02274686j -0.01343677-0.02404227j]
 [-0.01100765-0.02738239j  0.01067307+0.02539022j -0.01540474-0.02678967j]
 [-0.00773319-0.02046529j  0.00841566+0.01992265j -0.01178511-0.02057796j]
 [-0.00114575-0.00715318j  0.00196198+0.00769717j -0.00407387-0.00714226j]
 [ 0.00554221+0.00818582j -0.00522947-0.00683986j  0.00465216+0.00809035j]
 [ 0.01144658+0.02014963j -0.01160354-0.0176647j   0.01186605+0.01990549j]
 [ 0.01194396+0.02481772j -0.01238644-0.02389586j  0.01303195+0.02535742j]
 [ 0.01138789+0.02080041j -0.01109194-0.01944624j  0.01029330+0.02046596j]
 [ 0.00699530+0.00860771j -0.00479578-0.00890673j  0.00336275+0.0083651j ]
 [-0.00047268-0.00465341j  0.00210618+0.00594767j -0.00407920-0.00681585j]
 [-0.00559297-0.01731617j  0.00599357+0.02104061j -0.00959052-0.01966347j]
 [-0.00898619-0.02224159j  0.00951073+0.0279376j  -0.01134136-0.02724587j]
 [-0.00968377-0.01933804j  0.00862062+0.02717564j -0.01101139-0.02257669j]]
```

```
In [25]: # Let's reshape x_mes and z_mes vectors
         # in order to get position vectors with the same length
         # x -> [x1 ... x13 x1 ... x13 x1 ... x13]
         # z -> [z1 ... z1  z2 ... z2  z3 ...  z3]
         XX = tile(x_mes, len(z_mes))
         ZZ = repeat(z_mes, len(x_mes))

         # and the same for the measurements :
         # Emes -> [E(x1,z1) ... E(x13,z1) E(x1,z2) ... E(x13,z2) E(x1,z3) ... E(x13,z3)]
         EEmeas = reshape(Emeas, 39, order='F') # order='F' is important to get the correct ordering

In [26]: def optim_fun(a, x, z):
             Emodel = zeros_like(x, dtype='complex')

             for idx in range(len(x)):
                 Emodel[idx] = Ey(a, x[idx], z[idx])

             y = EEmeas - Emodel
             return y.real**2 + y.imag**2


In [27]: from scipy.optimize import leastsq
         a0 = np.array([1,0,1,0])
         sol=leastsq(optim_fun, a0, args=(XX, ZZ))
         a_sol = sol[0]
         print(abs(a_sol)/norm(sol[0])*100)

[  5.65498943   0.28415125  99.7811993    3.41359338]

In [28]: x_test = linspace(0, a, 201)

         def subplote(x_test, z, a, Emes):
             E_test = zeros_like(x_test, dtype='complex')

             for idx in range(len(x_test)):
                 E_test[idx] = Ey(a, x_test[idx], z)

             plot(x_test, real(E_test)/2, lw=2)
             plot(x_mes, real(Emes), lw=2)
             xlabel('x [m]')
             ylabel('|$E_y$| [a.u.]')
             grid()

         subplot(311)
         subplote(x_test, z_mes[0], a_sol, Emeas[:,0])
         title('z1')
         subplot(312)
         subplote(x_test, z_mes[1], a_sol, Emeas[:,1])
         title('z2')
         subplot(313)
         subplote(x_test, z_mes[2], a_sol, Emeas[:,2])
         title('z3')

Out[28]: <matplotlib.text.Text at 0x97059b0>
```



1.6 Least Square Equation Solving

We can also directly solve the problem, by defining

$$\phi_{ij} = \sin\left(\frac{j\pi}{a}x_i\right)e^{-j\beta_j z_i}$$

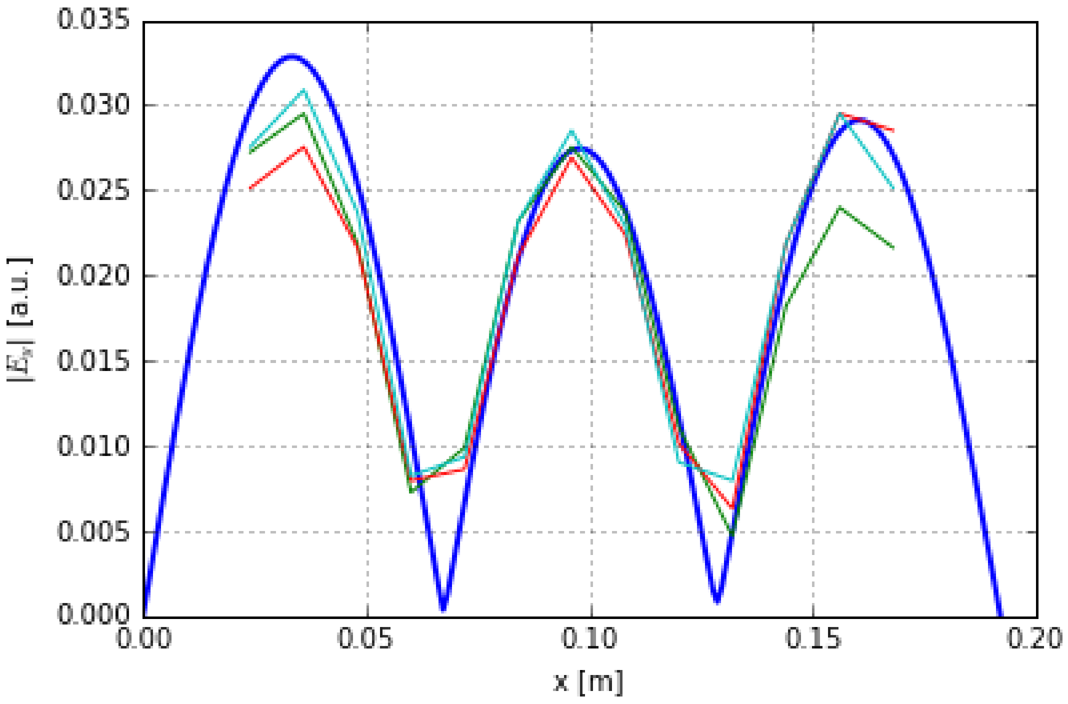
Then

$$\vec{a} = \left(\phi^T\phi\right)^{-1}\phi^T\vec{E}_{meas}$$

```
In [29]: def phi(n, x, z, wg_a=192e-3, f=3.7e9):
             k0 = 2*pi*f/c

             if k0**2 - (n*pi/wg_a)**2 >= 0:
                 beta_n = np.sqrt(k0**2 - (n*pi/wg_a)**2)
             else:
                 beta_n = -1j*np.sqrt((n*pi/wg_a)**2 - k0**2)

             return  np.sin(n*pi/wg_a*x) * np.exp(-1j*beta_n*z)

In [30]: MAT = np.array([phi(1, XX, ZZ), phi(2, XX, ZZ), phi(3, XX, ZZ), phi(4, XX, ZZ)]).T

In [31]: a_sol = np.linalg.inv(np.dot(MAT.T, MAT)).dot(MAT.T).dot(EEmeas)
         print(abs(a_sol)/norm(a_sol)*100)
```

```
[  8.09702917   9.86914378  99.1277604    3.27494091]
```

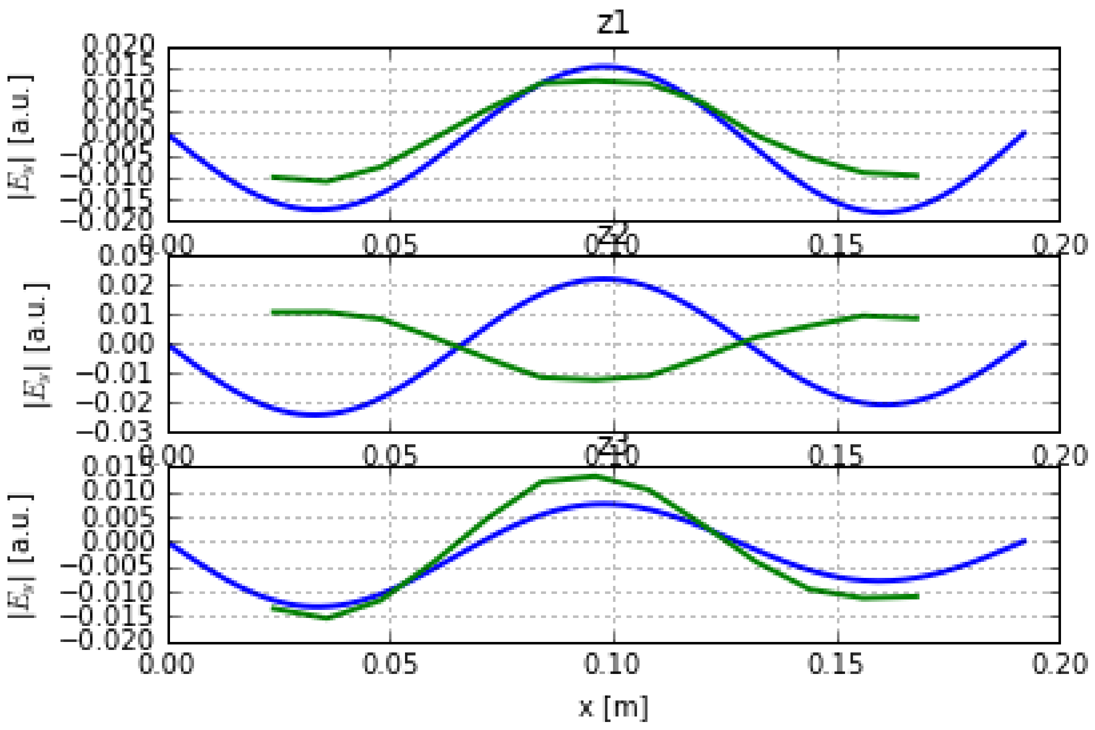
```
In [32]: x_test = linspace(0, a, 201)
         z_test = zeros_like(x_test)
         E_test = zeros_like(x_test, dtype='complex')


         for idx in range(len(x_test)):
             E_test[idx] = Ey(a_sol, x_test[idx], z_test[idx])


         plot(x_test, abs(E_test)/2, lw=2)
         plot(x_mes, 10**(ampl_dB/20))
         xlabel('x [m]')
         ylabel('|$E_y$| [a.u.]')
         grid()
```



```
In [33]: x_test = linspace(0, a, 201)

         def subplote(x_test, z, a, Emes):
             E_test = zeros_like(x_test, dtype='complex')

             for idx in range(len(x_test)):
                 E_test[idx] = Ey(a, x_test[idx], z)

             plot(x_test, real(E_test)/2, lw=2)
             plot(x_mes, real(Emes), lw=2)
             xlabel('x [m]')
             ylabel('|$E_y$| [a.u.]')
             grid()
```

```
        subplot(311)
        subplote(x_test, z_mes[0], a_sol, Emeas[:,0])
        title('z1')
        subplot(312)
        subplote(x_test, z_mes[1], a_sol, Emeas[:,1])
        title('z2')
        subplot(313)
        subplote(x_test, z_mes[2], a_sol, Emeas[:,2])
        title('z3')
```

```
Out[33]: <matplotlib.text.Text at 0x9902d68>
```



1.7 2D plot view

```
In [34]: x = linspace(0, a, 201)
         z = linspace(0, 10e-2, 301)

         XX, ZZ = meshgrid(x,z)
         XX2 = reshape(XX, len(x)*len(z))
         ZZ2 = reshape(ZZ, len(x)*len(z))
         E_2D = zeros_like(XX2, dtype='complex')
         for idx in range(len(XX2)):
             E_2D[idx] = Ey(a_sol, XX2[idx], ZZ2[idx])
         print(shape(E_2D))
         E_2D = reshape(E_2D, (len(z), len(x))    )
         print(shape(E_2D))
```

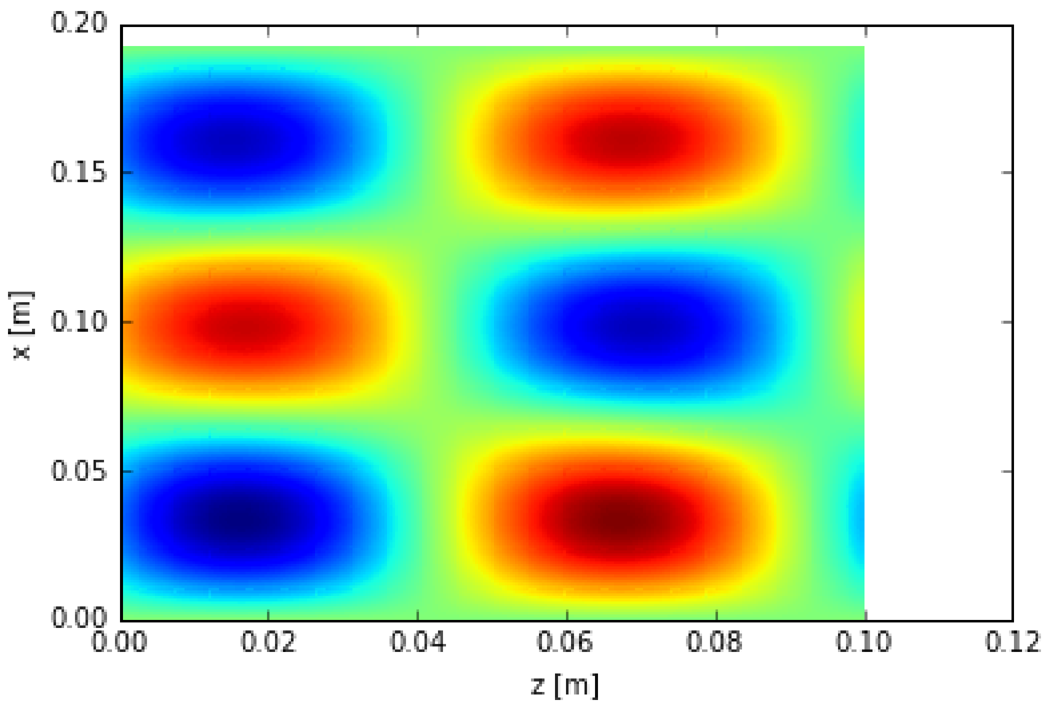
```
(60501,)
(301, 201)
```

```
In [35]: shape(XX)
```

```
Out[35]: (301, 201)
```

```
In [36]: pcolor(ZZ, XX, real(E_2D))
         xlabel('z [m]')
         ylabel('x [m]')
```

```
Out[36]: <matplotlib.text.Text at 0x982e4a8>
```



1.7.1 CSS Styling

```
In [37]: from IPython.core.display import HTML
         def css_styling():
             styles = open("../styles/custom.css", "r").read()
             return HTML(styles)
         css_styling()
```

```
Out[37]: <IPython.core.display.HTML object>
```

```
In [ ]:
```