Fonones y espectroscopía Raman en semiconductores bidimensionales.

casimir

March 9, 2021

Contents

Abstract

Los materiales bidimensionales (2D) como el grafeno son de gran interés tanto por sus propiedades físicas exclusivas como por sus aplicaciones potenciales. El estudio de la dinámica de la red cristalina (fonones) de estos materiales es un requisito previo para entender su estabilidad estructural y propiedades térmicas, así como sus propiedades de transporte y ópticas.

Este Trabajo de Fin de Grado consiste en la computación de los modos vibracionales de materiales semiconductores 2D y su correlación con los observables relevantes para la interpretación de los experimentos de dispersión de luz.

La redacción del TFG todavía se encuentra en una versión muy preliminar, de echo es más un bloc de notas que lo que espero que sea el TFG final.

1 TODO Introduccion

Las vibraciones reticulares están regidas por las fuerzas que experimentan los átomos cuando se desplazan de su posición de equilibrio. La primera hipótesis es que cada átomo tiene una posicion de equilibrio en el cristal, y consideraremos que estos átomos vibran con una amplitud pequeña alrededor de su posición de equilibrio, de manera que el sólido se encuentra en estados que corresponden a lo que macroscópicamente se conoce como *la region de comportamiento elástico lineal*, donde se verifica la ley de Hooke.

Podremos por tanto aproximar la energia potencial de interaccion por el termino armónico de su desarrolo en serie de potencias del desplazamiento, y la fuerza resultante es por tanto una funcion aproximadamente lineal del desplazamiento

2 TODO Desarrollo del trabajo

2.1 TODO Formular la matriz dinámica para el BN monolayer

En el caso del BN monolayer estamos tratando con un cristal bidimensional del base diatómica, cuya celda unidad viene dada por:

$$\vec{a}_1 = a(1,0); \qquad \vec{a}_2 = a\left(-\frac{1}{2}, \frac{\sqrt{3}}{2}\right) \tag{1}$$

```
import numpy as np
from  numpy  import array, sqrt, sort, vdot, pi, arccos
from numpy.linalg import norm
import pandas as pd
import matplotlib
from matplotlib import pyplot as plt

a=1
a1=np.array([a,0])
a2=np.array([-a/2,sqrt(3)*a/2])
```

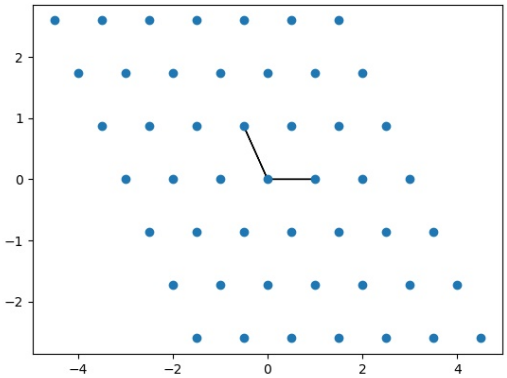
Podemos comprobar que efectivamente se trata de una celdilla hexagonal, pues los dos vectores base forman un angulo `'{:.2f}'.format(arccos(vdot(a1,a2)))` $= 2\pi/3$ rad.

Numeraremos las celdillas unidad con un índice vectorial $\vec{l} = (l_1, l_2)$.

Las posiciones de los nudos son $\vec{R}_{\vec{l}} = l_1\vec{a}_1 + l_2\vec{a}_2$.

Visualizamos una región de la red hexagonal, con los correspondientes nudos (que no átomos), así como la correspondiente celda unidad,

```
def R_l(l1,l2):
 return l1*a1+l2*a2

reddenudos=array([R_l(l1,l2) for l1 in range(-3, 4)
  for l2 in range(-3,4)])

x = reddenudos[:,0]
y = reddenudos[:,1]
plt.plot(x,y,"o")
ax = plt.axes()
ax.arrow(R_l(0,0)[0],R_l(0,0)[1],R_l(1,0)[0],R_l(1,0)[1])
ax.arrow(R_l(0,0)[0],R_l(0,0)[1],R_l(0,1)[0],R_l(0,1)[1])
plt.savefig("Graficas/Reddenudos.jpg")
plt.close()
```



Para calcular los modos de vibración por primeros principios debemos determinar primero las posiciones atómicas de equilibrio en la celda unidad **nota: proporcionadas como datos**

Los átomos estan situados en:

$$\vec{R}_B = \frac{1}{3}\vec{a_1} + 2\vec{a_2}$$
$$\vec{R}_N = \frac{2}{3}\vec{a_1} + \frac{1}{3}\vec{a_2} \tag{2}$$

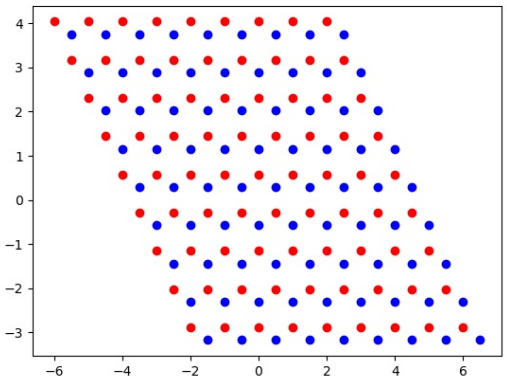
```
R_B=1/3*a1+2/3*a2
R_N=2/3*a1+1/3*a2
```

2.1.1 Cristales bidimensionales con base diatómica

Las posiciones de equilibrio de los átomos de la base respecto de su nudo son $\vec{R}^0_\nu$, con $\nu = 1, 2$, puesto que la base tiene 2 átomos, el 1 hará referencia a los átomos de B y 2 a los de átomos de N (notemos que aunque los átomos fuesen idénticos deberíamos especificar a que átomo de la base nos referimos, puesto que no ocupan posiciones equivalentes).

Las posiciones de equilibrio de los átomos: $\vec{R}_{\nu,\vec{l}} = \vec{R}_{\vec{l}} + \vec{R}^0_\nu$ así como los desplazamientos atómicos: $\vec{u}_{\nu,\vec{l}}$ quedarán por tanto identificados por medio de dos índices. La fuerza que ejerce el átomo $\nu,\vec{l}$ sobre el átomo $\nu',\vec{0}$ tiene aproximádamente la dirección determinada por las posiciones de equilibrio de estos átomos. Esta dirección es la del vector $\vec{R}_{\nu',\nu,\vec{l}} = \vec{R}_{\vec{l}} + \vec{R}^0_{\nu',\nu}$ donde $\vec{R}^0_{\nu',\nu} \equiv \vec{R}^0_\nu - \vec{R}^0_{\nu'}$.

```
#Posiciones de equilibrio de los átomos

def R_nu_l(nu,l1,l2):
  if nu == 1:
    return l1*a1+l2*a2+R_B

  elif nu == 2:
    return l1*a1+l2*a2+R_N

  else:
    print("Error, nu solo puede ser 1 o 2 ")

AtomosB=array([R_nu_l(1,l1,l2) for l1 in range(-4, 5)
               for l2 in range(-4,5)])

AtomosN=array([R_nu_l(2,l1,l2) for l1 in range(-4, 5)
               for l2 in range(-4,5)])

xB = AtomosB[:,0]
yB = AtomosB[:,1]
plt.plot(xB,yB,"o",color="red")

xN = AtomosN[:,0]
yN = AtomosN[:,1]
plt.plot(xN,yN,"o",color="blue")

plt.savefig("Graficas/Reddeatomos.jpg")
plt.close()
```



Las dimensiones del cristal son $L_1 = N_1 a_1$ y $L_2 = N_2 a_2$, donde N_i $(i = 1, 2)$ es el número de celdillas en la dirección $\hat{a}_i$. El cristal tiene tiene $N = N_1 N_2$ celdillas unidad primitivas y $2N$ átomos.

La idea básica es que si la base tiene N_ν átomos entonces debemos plantear y resolver las ecuaciones de movimiento de los N_ν átomos de la base de la celdilla $\vec{0}$, por lo tanto en el caso que estamos estudiando debemos resolver 2 ecuaciones vectoriales de movimiento: una para el átomo de B y la otra para el de N.

La fuerza que ejerce el átomo $\nu, \vec{l}$ sobre el átomo $\nu', \vec{0}$ se puede expresar de manera aproximada como:

$$F_{\nu',\vec{0},\nu,\vec{l}} = \alpha_{\nu',\nu,\vec{l}} \left(\hat{R}_{\nu',\nu,\vec{l}} \otimes \hat{R}_{\nu',\nu,\vec{l}} \right) \cdot \left(\vec{u}_{\nu,\vec{l}} - \vec{u}_{\nu',\vec{0}} \right)$$

donde $\hat{R}_{\nu',\nu,\vec{l}}$ es el vector unitario en la dirección $\vec{R}_{\nu',\nu,\vec{l}}$

La ecuación de movimiento del átomo $\nu', \vec{0}$ es por lo tanto:

$$m_{\nu'} \ddot{\vec{u}}_{\nu',\vec{0}} = \sum_{\nu,\vec{l}} \alpha_{\nu',\nu,\vec{l}} \left(\hat{R}_{\nu',\nu,\vec{l}} \otimes \hat{R}_{\nu',\nu,\vec{l}} \right) \cdot \left(\vec{u}_{\nu,\vec{l}} - \vec{u}_{\nu',\vec{0}} \right)$$

Buscaremos soluciones de la forma:

$$\vec{u}_{\nu,\vec{l}} = \vec{A}_\nu e^{i(\vec{q}\cdot\vec{R}_{\vec{l}} - \omega t)}$$

donde $\vec{A}_\nu$ es el *vector de polarización* que determina la amplitud y dirección de vibración de los átomos de tipo ν . Notemos que aunque el cristal sea bi-dimensional, los átomos de este pueden vibrar en las 3 direcciones espaciales.

Es importante apreciar que se necesitan tantas amplitudes de vibración como átomos tenga la base porque estos no ocupan posiciones equivalentes y describen vibraciones distintas. Se deben cumplir así $N_\nu = 2$ ecuaciones vectoriales del tipo

$$\boxed{-m_{\nu'}\omega^2 \vec{A}_{\nu'} = \sum_{\nu,\vec{l}} \alpha_{\nu',\nu,\vec{l}} \left(\hat{R}_{\nu',\nu,\vec{l}} \otimes \hat{R}_{\nu',\nu,\vec{l}} \right) \cdot \left(\vec{A}_\nu e^{i\vec{q}\cdot\vec{R}_{\vec{l}}} - \vec{A}_{\nu'} \right)} \tag{3}$$

Como se trata de un sistema de ecuaciones lineales homogéneas, se debe cumplir la correspondiente ecuación secular, es decir, que el determinante de la matriz de dimensión $3N_\nu \otimes 3N_\nu$ $(3 \cdot 2 \otimes 3 \cdot 2)$ de los coeficientes $A_{\nu',i}$ en la ecuación 3 sea nula. Esta ecuación tiene $3N_\nu = 6$ soluciones que describen las 6 ramas de la relación de dispersión, es decir, las 6 frecuencias características de los 6 modos normales de vibración de vector de onda $\vec{q}$. Se cumple que el número total de modos normales de vibración coincide con el triple del número total de átomos, es decir, *el número total de modos normales de vibración coincide con el de grados de libertad de movimiento de los átomos*

Debemos notar que tal y como esta escrita la ecuación 3, sólo estamos considerando vibraciones dentro del plano del cristal (puesto que estamos considerando $\hat{R}_{\nu',\nu,\vec{l}}$ como vectores de 2 dimensiones . Como los átomos pueden moverse en la tres dimensiones debemos añadir una tercera componente a la matriz $\hat{R}_{\nu\prime\nu\vec{l}} \otimes \hat{R}_{\nu\prime\nu\vec{l}}$, y dado que podemos considerar que las vibraciones perpendiculares al plano están completamente desacopladas de las interplanares, simplemente añadimos la unidad a la diagonal de la matriz, siendo nulos los otros elementos añadidos.

Así que tratamos por un lado las vibraciones en el plano por y por otro las perpendiculares al plano del cristal, ya que como hemos comentado se trata de vibraciones completamente desacopladas.

Debemos determinar cuales son las posiciones de equilibrio de los átomos más cercanos a los átomos de la celda $\vec{0}$: para ello genero un array con los datos que voy a necesitar $(\nu, \nu', \hat{R}_{\nu\prime,nu,\vec{l}}, \ldots$ ordenandolos según su distancia a los 2 átomos de la celda $l = \vec{0}$ hasta los cuartos vecinos (usando para ello un DataFrame de pandas, que facilita mucho la manipulación de los datos)

```
def propiedades_atomos(l1, l2):

   return [(k, m, i, j, R_nu(m,i,j),
    (R_nu(m,i,j)-R_nu(k,0,0))/norm(R_nu(m,i,j)-R_nu(k,0,0)), norm(R_nu(m,i,j)-R_nu(k,0,0)))
    for k in [1,2] for m in [1,2] for i in range(-l1,l1+1) for j in range(-l2,l2+1)]

columnas = [r"$\nu$",r"$\nu\prime$",r"$l_1$", r"$l_2$", r"$\vec R_{\nu,\vec l}$",
            r"$\hat R_{\nu\prime,\nu,\vec l}$",'Distancia' ]

def Atomos(l1, l2):
   return pd.DataFrame(propiedades_atomos(l1, l2),columns=columnas).sort_values(
    ['Distancia',r"$\nu\prime$"], ascending=[True, True])

Atomos(2,2).head(38).to_latex(escape=False,float_format="{:0.4f}".format,index=False)
```

ν	$\nu\prime$	l_1	l_2	$\vec{R}_{\nu,\vec{l}}$	$\hat{R}_{\nu\prime,\nu,\vec{l}}$	Distancia
1	1	0	0	[0.0, 0.5773502691896257]	[nan, nan]	0.0000
2	2	0	0	[0.5, 0.28867513459481287]	[nan, nan]	0.0000
2	1	0	-1	[0.5, -0.28867513459481287]	[0.0, -1.0]	0.5774
2	1	0	0	[0.0, 0.5773502691896257]	[-0.8660254037844387, 0.5]	0.5774
2	1	1	0	[1.0, 0.5773502691896257]	[0.8660254037844387, 0.5]	0.5774
1	2	-1	0	[-0.5, 0.28867513459481287]	[-0.8660254037844387, -0.5]	0.5774
1	2	0	0	[0.5, 0.28867513459481287]	[0.8660254037844387, -0.5]	0.5774
1	2	0	1	[0.0, 1.1547005383792515]	[0.0, 1.0]	0.5774
1	1	-1	-1	[-0.5, -0.28867513459481287]	[-0.5000000000000001, -0.8660254037844387]	1.0000
1	1	0	-1	[0.5, -0.28867513459481287]	[0.5000000000000001, -0.8660254037844387]	1.0000
1	1	0	1	[-0.5, 1.4433756729740643]	[-0.5000000000000001, 0.8660254037844387]	1.0000
1	1	1	1	[0.5, 1.4433756729740643]	[0.5000000000000001, 0.8660254037844387]	1.0000
2	2	-1	-1	[0.0, -0.5773502691896257]	[-0.5000000000000001, -0.8660254037844387]	1.0000
2	2	0	-1	[1.0, -0.5773502691896257]	[0.5000000000000001, -0.8660254037844387]	1.0000
2	2	0	1	[0.0, 1.1547005383792515]	[-0.5000000000000001, 0.8660254037844387]	1.0000
2	2	1	1	[1.0, 1.1547005383792515]	[0.5000000000000001, 0.8660254037844387]	1.0000
1	1	-1	0	[-1.0, 0.5773502691896257]	[-1.0, 0.0]	1.0000
1	1	1	0	[1.0, 0.5773502691896257]	[1.0, 0.0]	1.0000
2	2	-1	0	[-0.5, 0.28867513459481287]	[-1.0, 0.0]	1.0000
2	2	1	0	[1.5, 0.28867513459481287]	[1.0, 0.0]	1.0000
2	1	-1	-1	[-0.5, -0.28867513459481287]	[-0.8660254037844387, -0.5]	1.1547
2	1	1	-1	[1.5, -0.28867513459481287]	[0.8660254037844387, -0.5]	1.1547
2	1	1	1	[0.5, 1.4433756729740643]	[0.0, 1.0]	1.1547
1	2	-1	-1	[0.0, -0.5773502691896257]	[0.0, -1.0]	1.1547
1	2	-1	1	[-1.0, 1.1547005383792515]	[-0.8660254037844387, 0.5]	1.1547
1	2	1	1	[1.0, 1.1547005383792515]	[0.8660254037844387, 0.5]	1.1547
2	1	-1	-2	[0.0, -1.1547005383792515]	[-0.3273268353539886, -0.944911182523068]	1.5275
2	1	0	-2	[1.0, -1.1547005383792515]	[0.3273268353539886, -0.944911182523068]	1.5275
2	1	0	1	[-0.5, 1.4433756729740643]	[-0.6546536707079772, 0.7559289460184545]	1.5275
2	1	2	1	[1.5, 1.4433756729740643]	[0.6546536707079772, 0.7559289460184545]	1.5275
1	2	-2	-1	[-1.0, -0.5773502691896257]	[-0.6546536707079772, -0.7559289460184545]	1.5275
1	2	0	-1	[1.0, -0.5773502691896257]	[0.6546536707079772, -0.7559289460184545]	1.5275
1	2	0	2	[-0.5, 2.02072594216369]	[-0.3273268353539886, 0.9449111825230679]	1.5275
1	2	1	2	[0.5, 2.02072594216369]	[0.3273268353539886, 0.9449111825230679]	1.5275
2	1	-1	0	[-1.0, 0.5773502691896257]	[-0.9819805060619656, 0.1889822365046136]	1.5275
2	1	2	0	[2.0, 0.5773502691896257]	[0.9819805060619656, 0.1889822365046136]	1.5275
1	2	-2	0	[-1.5, 0.28867513459481287]	[-0.9819805060619656, -0.1889822365046136]	1.5275
1	2	1	0	[1.5, 0.28867513459481287]	[0.9819805060619656, -0.1889822365046136]	1.5275

2.1.2 Fijemonos, por ahora, sólo en los primeros vecinos:

Paso a usar sympy en vez de numpy (si eso ya cambiare la parte anterior a este apartado más adelante, las modificaciones son mínimas) para intentar determinar la matrix dinámica (y resolver la ecuación secular) simbólicamente

```
from sympy import *
a=Symbol('a', real=True, positive=True)
q_x=Symbol('q_x', real=True);  q_y=Symbol('q_y', real=True)
q=Matrix([q_x,q_y])
a_1=Matrix([a,0]); a_2=Matrix([-a/2,sqrt(3)*a/2])
R_B=1/3*a_1+2/3*a_2; R_N=2/3*a_1+1/3*a_2

def R_l(l_1,l_2):
  return l_1*a_1+l_2*a_2

def R_nu_l(nu,l_1,l_2):
  if nu == 1:
    return l_1*a_1+l_2*a_2+R_B

  elif nu == 2:
    return l_1*a_1+l_2*a_2+R_N

  else:
    print("Error, nu solo puede ser 1 o 2 ")

def R_hat(nuprima,nu,l_1,l_2):
  return (R_nu_l(nu,l_1,l_2)-R_nu_l(nuprima,0,0))/(R_nu_l(nu,l_1,l_2)-R_nu_l(nuprima,0,0)).norm()

def propiedades_atomos(l_1, l_2):
  return [(k, m, i, j, R_hat(k,m,i,j), (R_nu_l(m,i,j)-R_nu_l(k,0,0)).norm())
     for k in [1,2] for m in [1,2] for i in range(-l_1,l_1+1) for j in range(-l_2,l_2+1)]

columnas = [r"$\nu\prime$",r"$\nu",r"$l_1$", r"$l_2$",r"$\hat R_{\nu\prime,\nu,\vec l}$"
            ,'Distancia']

def Atomos(l_1, l_2):
  return pd.DataFrame(propiedades_atomos(l_1, l_2),columns=columnas).sort_values(
    ['Distancia',r"$\nu\prime$"], ascending=[True, True])

PrimerosVecinosBoro= Atomos(1,1)[(Atomos(1,1)['Distancia']/a<0.9) &
                          (Atomos(1,1)['Distancia']/a>0) & (Atomos(1,1)[r"$\nu\prime$"]==1)]

PrimerosVecinosNitrogeno= Atomos(1,1)[(Atomos(1,1)['Distancia']/a<0.9) &
                          (Atomos(1,1)['Distancia']/a>0) & (Atomos(1,1)[r"$\nu\prime$"]==2)]
```

```
PrimerosVecinosBoro.to_latex(escape=False,index=False)
```

$\nu\prime$	ν	l_1	l_2	$\hat{R}_{\nu\prime,\nu,\vec{l}}$	Distancia
1	2	-1	0	[-0.866025403784439, -0.288675134594813*sqrt(3)]	0.577350269189626*a
1	2	0	0	[0.866025403784439, -0.288675134594813*sqrt(3)]	0.577350269189626*a
1	2	0	1	[0, 0.577350269189626*sqrt(3)]	0.577350269189626*a

```
alpha=Symbol('alpha')
A1x, A1y, A1z, A2x, A2y, A2z = symbols("A1x, A1y, A1z, A2x, A2y, A2z")
M1=(R_hat(1,2,-1,0)*R_hat(1,2,-1,0).T*Matrix([A2x*exp(-I*q.dot(a_1))-A1x,
    A2y*exp(-I*q.dot(a_1))-A1y])+
    (R_hat(1,2,0,0)*R_hat(1,2,0,0).T*Matrix([A2x-A1x, A2y-A1y]))+
      (R_hat(1,2,0,1)*R_hat(1,2,0,1).T*Matrix([A2x*exp(I*q.dot(a_2))-A1x,
        A2y*exp(I*q.dot(a_2))-A1y])))

PrimerosVecinosNitrogeno.to_latex(escape=False,index=False)
```

$\nu\prime$	ν	l_1	l_2	$\hat{R}_{\nu\prime,\nu,\vec{l}}$	Distancia
2	1	0	0	[-0.866025403784439, 0.288675134594813*sqrt(3)]	0.577350269189626*a
2	1	1	0	[0.866025403784439, 0.288675134594813*sqrt(3)]	0.577350269189626*a
2	1	0	-1	[0, -0.577350269189626*sqrt(3)]	0.577350269189626*a

```
M2=(R_hat(2,1,0,-1)*R_hat(2,1,0,-1).T*Matrix([A2x*exp(-I*q.dot(a_2))-A1x,
      A2y*exp(-I*q.dot(a_2))-A1y])+(R_hat(2,1,0,0)*R_hat(2,1,0,0).T*
       Matrix([A2x-A1x, A2y-A1y]))+  (R_hat(2,1,1,0)*R_hat(2,1,1,0).T*
        Matrix([A2x*exp(I*q.dot(a_1))-A1x, A2y-A1y])))

M=M1.row_insert(2,M2)
print_latex(M) #.rewrite(cos).trigsimp()
```

$$\begin{bmatrix} -1.5A1x + 0.75A2x + 0.75A2xe^{-iaq_x} - 0.25\sqrt{3}\left(-A1y + A2y\right) + 0.25\sqrt{3}\left(-A1y + A2ye^{-iaq_x}\right) \\ -1.5A1y + 1.0A2ye^{i\left(-\frac{aq_x}{2}+\frac{\sqrt{3}aq_y}{2}\right)} + 0.25A2y + 0.25A2ye^{-iaq_x} - 0.25\sqrt{3}\left(-A1x + A2x\right) + 0.25\sqrt{3}\left(-A1x + A2xe^{-iaq_x}\right) \\ -1.5A1x + 0.75A2xe^{iaq_x} + 0.75A2x \\ -1.5A1y + 0.5A2y + 1.0A2ye^{-i\left(-\frac{aq_x}{2}+\frac{\sqrt{3}aq_y}{2}\right)} - 0.25\sqrt{3}\left(-A1x + A2x\right) + 0.25\sqrt{3}\left(-A1x + A2xe^{iaq_x}\right) \end{bmatrix} \tag{4}$$

Por otra parte, tenemos que considerar las vibraciones perpendiculares al plano del cristal, notemos que en este caso tenemos que resolver dos ecuaciones escalares del tipo:

$$-m_\nu\omega^2 A_\nu = \alpha \sum_{\nu\prime,\vec{l}} \left(A_{\nu\prime}e^{i\vec{q}\cdot\vec{R}_{\vec{l}}} - A_\nu\right) \tag{5}$$

- Para $\nu\prime = 1$

```
M_1_z=Matrix([A2z*exp(-I*q.dot(a_1))-A1z+ A2z-A1z+A2z*exp(I*q.dot(a_2))-A1z])
```

- Para $\nu\prime = 2$

```
M_2_z=Matrix([A1z-A2z+ A1z*exp(I*q.dot(a_1))-A2z+A1z*exp(-I*q.dot(a_2))-A2z])

Mprova1=M.row_insert(2,M_1_z)
Mprova2=Mprova1.row_insert(5,M_1_z)
print_latex(Mprova2.expand()) #.rewrite(cos).trigsimp()
```

$$\begin{bmatrix} -1.5A1x + 0.75A2x + 0.75A2xe^{-iaq_x} - 0.25\sqrt{3}A2y + 0.25\sqrt{3}A2ye^{-iaq_x} \\ -1.5A1y - 0.25\sqrt{3}A2x + 0.25\sqrt{3}A2xe^{-iaq_x} + 0.25A2y + 0.25A2ye^{-iaq_x} + 1.0A2ye^{-\frac{iaq_x}{2}}e^{\frac{\sqrt{3}iaq_y}{2}} \\ -3A1z + A2z + A2ze^{-iaq_x} + A2ze^{-\frac{iaq_x}{2}}e^{\frac{\sqrt{3}iaq_y}{2}} \\ -1.5A1x + 0.75A2xe^{iaq_x} + 0.75A2x \\ -1.5A1y + 0.25\sqrt{3}A2xe^{iaq_x} - 0.25\sqrt{3}A2x + 1.0A2ye^{\frac{iaq_x}{2}}e^{-\frac{\sqrt{3}iaq_y}{2}} + 0.5A2y \\ -3A1z + A2z + A2ze^{-iaq_x} + A2ze^{-\frac{iaq_x}{2}}e^{\frac{\sqrt{3}iaq_y}{2}} \end{bmatrix} \tag{6}$$

A falta de multiplicar cada fila por la correspondiente m_ν (3 de arriba por m_1 y 3 de abajo por m_2) y constante de fuerza α esta vector (es el resultado de multiplicar la denominada matriz dinámica por el vector polarzación $\vec{A}$, de 6 componentes

```
A=Matrix([A1x,A1y,A1z,A2x,A2y,A2z])
print_latex(A)
```

$$\vec{A} = \begin{bmatrix} A1x \\ A1y \\ A1z \\ A2x \\ A2y \\ A2z \end{bmatrix} \tag{7}$$

- Pasamos ya a calcular la matriz dinámica considerndo sólo los primeros vecinos ...

```
print_latex(Matrix([Mprova2[i].coeff(j).expand().rewrite(cos).trigsimp() for i in range(0,6) for j
 in [A1x, A1y, A1z, A2x, A2y, A2z]]).reshape(6,6))
```

$$\begin{bmatrix} -1.5 & 0 & 0 & 0.75 + 0.75e^{-iaq_x} & 0 & 0 \\ 0 & -1.5 & 0 & 0 & 1.0e^{i\left(-\frac{aq_x}{2}+\frac{\sqrt{3}aq_y}{2}\right)} + 0.25 + 0.25e^{-iaq_x} & 0 \\ 0 & 0 & -3 & 0 & 0 & e^{i\left(-\frac{aq_x}{2}+\frac{\sqrt{3}aq_y}{2}\right)} + 1 + e^{-iaq_x} \\ -1.5 & 0 & 0 & 0.75e^{iaq_x} + 0.75 & 0 & 0 \\ 0 & -1.5 & 0 & 0 & 0.5 + 1.0e^{-i\left(-\frac{aq_x}{2}+\frac{\sqrt{3}aq_y}{2}\right)} & 0 \\ 0 & 0 & -3 & 0 & 0 & e^{i\left(-\frac{aq_x}{2}+\frac{\sqrt{3}aq_y}{2}\right)} + 1 + e^{-iaq_x} \end{bmatrix} \tag{8}$$

Una vez calculada bién la matriz dinámica para los primeros vecinos (aún no lo está), la idea es programarla directamente mediante un script, no sólo para los primeros vecinos, sino hasa los cuartos vecinos (en vez de mirar yo la distancia, el programa creará una nueva matriz cuando cambie el valor de la distancia ...)