Notebook-Casimir

April 18, 2021

1 Descripción del BN monolayer

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

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

[1]:
```
reset()
%display latex
var('a', domain='positive')
a_1=a*vector([1,0])
a_2=a*vector([-1/2,sqrt(3)/2])
```

Podemos comprobar que efectivamente se trata de una celdilla hexagonal, puesto que sus dos vectores primitivos forman un ángulo de 120 (o 60) grados.

[2]:
```
arccos(a_1*a_2/(norm(a_1)*norm(a_2)).simplify())
```

[2]: $\frac{2}{3}\,\pi$

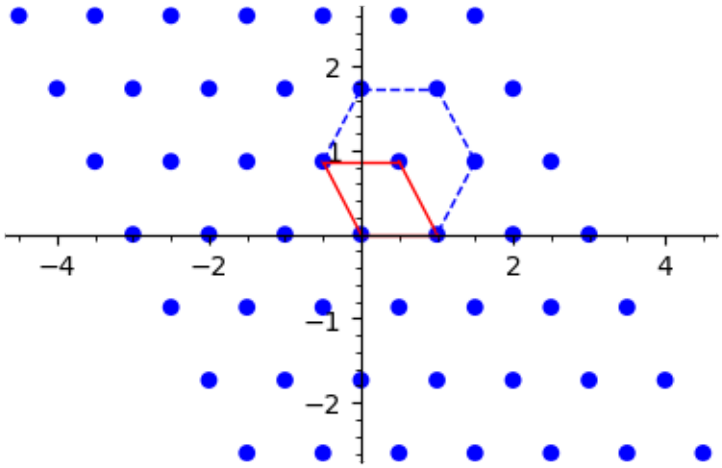
Numeraremos las celdillas con un índice vectorial $\vec{l} = (l_1, l_2)$, aunque habitualmente las encontramos numeradas simplemente con un índice entero, n.

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,

[3]:
```
nudos=points([l_1*a_1/a+l_2*a_2/a for l_1 in range(-3, 4) for l_2 in␣
 ↪range(-3,4)],
             size=40, color="blue", frame=False)

show(nudos+
    line([(0,0),(a_1/a)],color="red")+
     line([(0,0),(a_2/a)],color="red")+
     line([(a_1/a),(a_1/a+a_2/a)],color="red")+
     line([(a_2/a),(a_2/a+a_1/a)],color="red")+
     line([(a_2/a),(a_1/a+2*a_2/a)],linestyle="--")+
     line([(a_1/a+2*a_2/a),(2*a_1/a+2*a_2/a)],linestyle="--")+
```

```
    line([(2*a_1/a+2*a_2/a),(2*a_1/a+a_2/a)],linestyle="--")+
    line([(2*a_1/a+a_2/a),(a_1/a)],linestyle="--"), figsize=4)
```

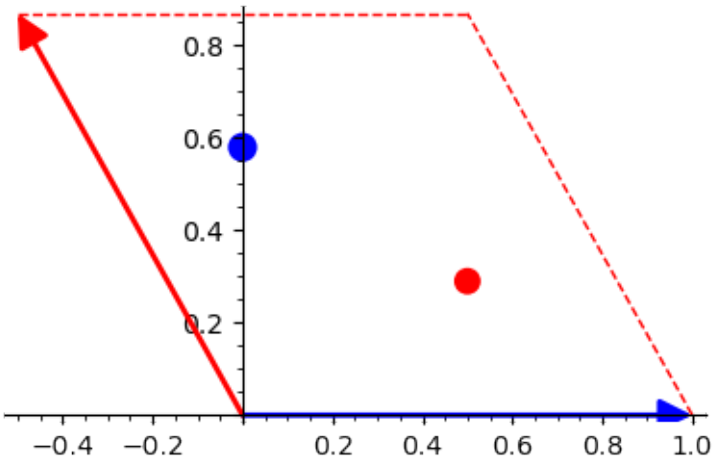


Para calcular los modos de vibración por primeros principios debemos determinar primero las posiciones atómicas de equilibrio en la celda unidad **nota: proporcionadas como datos**

Los átomos estan situados en:

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

[4]:
```
r_B=1/3*a_1+2/3*a_2; r_N=2/3*a_1+1/3*a_2
show(arrow((0,0),(a_1/a),color="blue")+
      arrow((0,0),(a_2/a),color="red")+
      line([(a_1/a),(a_1/a+a_2/a)],linestyle="--",color="red")+
      line([(a_2/a),(a_2/a+a_1/a)],linestyle="--",color="red")+
      point(r_B/a, size=120,color="blue")+
      point(r_N/a, size=100,color="red"), frame=False, figsize=4)
```



1.1 Identificación de los vecinos segun su distáncia

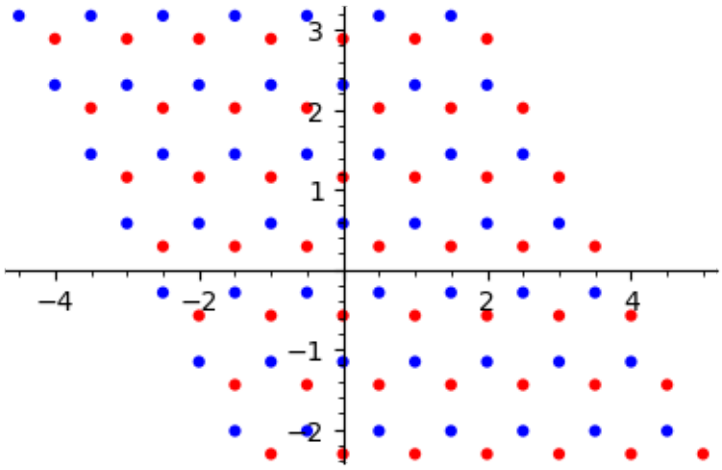
Podemos dibujar los átomos en sus respectivas posiciones de equilibrio:

[5]:
```
#Posiciones de equilibrio de los átomos
def RB(l_1,l_2):
    return (l_1*a_1+l_2*a_2+r_B)

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

AtomosB=points([RB(l_1,l_2)/a for l_1 in range(-3, 4) for l_2 in␣
 ↪range(-3,4)],size=20,color='blue')
AtomosN=points([RN(l_1,l_2)/a for l_1 in range(-3, 4) for l_2 in␣
 ↪range(-3,4)],size=20,color='red')

show(AtomosB+AtomosN, figsize=4)
```



Determinamos los vecinos de los dos átomos situados en la celdilla $\vec{0}$, para poder clasificarlos segun su distancia a cada uno de estos respectivos átomos.

[6]:
```
#import pandas as pd
var('q_x, q_y'); assume(q_x, q_y, 'real'); q=vector([q_x,q_y])


## Parametros de la red, de la celdilla y del cristal
## Vector R_l (vector de traslación primitivo)
def R_l(l_1,l_2):
    return l_1*a_1+l_2*a_2

## Vector de posición de los átomos del cristal (en equilibrio)
def R_alpha_l(alpha,l_1,l_2):
    if alpha == 1:
        return l_1*a_1+l_2*a_2+r_B

    elif alpha == 2:
        return l_1*a_1+l_2*a_2+r_N

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

## Vector unitario que une uno de los átomo alphaprima con el átomo considerado␣
 ↪alpha, l_1,l_2
def R_hat(alphaprima,alpha,l_1,l_2):
    if (R_alpha_l(alpha,l_1,l_2)-R_alpha_l(alphaprima,0,0)).norm()>0:
        return (R_alpha_l(alpha,l_1,l_2)-R_alpha_l(alphaprima,0,0))/
 ↪(R_alpha_l(alpha,l_1,l_2)\
```

```
 ↪-R_alpha_l(alphaprima,0,0)).norm()
    else:
        return (R_alpha_l(alpha,l_1,l_2)-R_alpha_l(alphaprima,0,0))

# Distancia entre el átomo alphaprime y su vecino alpha situado en la celdilla␣
 ↪l_1,l_2
def distancia(alphaprime,alpha,l_1,l_2):
    return (R_alpha_l(alpha,l_1,l_2)-R_alpha_l(alphaprime,0,0)).norm()/a

def fase(l_1,l_2):
    return exp(I*q*R_l(l_1,l_2))

#Genero una lista con la distancia de cada átomo a los átomos de la celdilla␣
 ↪unidad
#def valores_atomos(l_1, l_2):
#    return [(k, m, i, j, R_hat(k, m, i, j), distancia(k,m,i,j)) 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)]

## Construyo un DataFrame de pandas con la información necesaria para␣
 ↪identificar a los primeros, segundos, ... vecinos, según su distancia a cada␣
 ↪uno de los átomos de la celdilla unidad
#columnas = [r"$\alpha\prime$",r"$\alpha$",r"$l_1$", r"$l_2$", r"$\hat␣
 ↪R$",'Distancia']

#def lista_atomos(l_1, l_2):
#    return pd.DataFrame(valores_atomos(l_1,l_2),columns=columnas).
 ↪sort_values(['Distancia',r"$\alpha\prime$"], ascending=[True,True])

## Mostramos el dataframe como una tabla
#table(lista_atomos(2,2).to_html(index=False))
```

[7]:
```
#Angulo que forma el átomo considerado respecto al eje x

def angulo(alphaprima,alpha,l_1,l_2):
    if R_hat(alphaprima,alpha,l_1,l_2)[1] <0:
        return -acos(R_hat(alphaprima,alpha,l_1,l_2)*vector([1,0]))

    else:
        return acos(R_hat(alphaprima,alpha,l_1,l_2)*vector([1,0]))

#Matriz unitaria de rotación para cambio de ejes coordenados (entorno al eje z)

def U(theta):
```

```
    return matrix([[cos(theta),sin(theta),0], [-sin(theta),␣
 ↪cos(theta),0],[0,0,1]])

#Matriz de fuerza para los primeros vecinos

var('M_B,M_N', domain='positive')
var('omega')

phi1rBN=var('phi1rBN',latex_name='\\phi_{1,r}^{BN}')
phi1tiBN=var('phi1tiBN',latex_name='\\phi_{1,ti}^{BN}')
phi1toBN=var('phi1toBN',latex_name='\\phi_{1,to}^{BN}',domain='real')

phi1rNB=phi1rBN; phi1tiNB=phi1tiBN; phi1toNB=phi1toBN


Phi_10__BN=1/sqrt(M_B*M_N)*Matrix([[phi1rBN,0,0],[0,phi1tiBN,0],[0,0,phi1toBN]])
Phi_10__NB=1/sqrt(M_N*M_B)*Matrix([[phi1rNB,0,0],[0,phi1tiNB,0],[0,0,phi1toNB]])


def Phi_1l__BN(alphaprima,alpha,l_1,l_2):
    return U(-angulo(alphaprima,alpha,l_1,l_2))*Phi_10__BN*\
           U(angulo(alphaprima,alpha,l_1,l_2))

def Phi_1l__NB(alphaprima,alpha,l_1,l_2):
    return U(-angulo(alphaprima,alpha,l_1,l_2))*Phi_10__NB*\
           U(angulo(alphaprima,alpha,l_1,l_2))

def D_1l_BN(alphaprima,alpha,l_1,l_2):
    return Phi_1l__BN(alphaprima,alpha,l_1,l_2)*fase(l_1,l_2)

def D_1l_NB(alphaprima,alpha,l_1,l_2):
    return Phi_1l__NB(alphaprima,alpha,l_1,l_2)*fase(l_1,l_2)

# Matriz de fuerza para los segundos vecinos

phi2rBB=var('phi2rBB',latex_name='\\phi_{2,r}^{BB}')
phi2tiBB=var('phi2tiBB',latex_name='\\phi_{2,ti}^{BB}')
phi2toBB=var('phi2toBB',latex_name='\\phi_{2,to}^{BB}',domain='real')

phi2rNN=var('phi2rNN',latex_name='\\phi_{2,r}^{NN}')
phi2tiNN=var('phi2tiNN',latex_name='\\phi_{2,ti}^{NN}')
phi2toNN=var('phi2toNN',latex_name='\\phi_{2,to}^{NN}',domain='real')

Phi_20__BB=1/M_B*Matrix([[phi2rBB,0,0],[0,phi2tiBB,0],[0,0,phi2toBB]])
Phi_20__NN=1/M_N*Matrix([[phi2rNN,0,0],[0,phi2tiNN,0],[0,0,phi2toNN]])

#A tener en cuenta: cuando consideramos el mismo tipo de átomos
```

```
# (de la misma subred, no porque sean el mismmo elemento)

def Phi_2l__BB(alphaprima,alpha,l_1,l_2):
    return U(-angulo(alphaprima,alpha,l_1,l_2))*Phi_20__BB*\
           U(angulo(alphaprima,alpha,l_1,l_2))

def Phi_2l__NN(alphaprima,alpha,l_1,l_2):
    return U(-angulo(alphaprima,alpha,l_1,l_2))*Phi_20__NN*\
           U(angulo(alphaprima,alpha,l_1,l_2))

def D_2l_BB(alphaprima,alpha,l_1,l_2):
    return Phi_2l__BB(alphaprima,alpha,l_1,l_2)*fase(l_1,l_2)

def D_2l_NN(alphaprima,alpha,l_1,l_2):
    return Phi_2l__NN(alphaprima,alpha,l_1,l_2)*fase(l_1,l_2)


#Matriz de fuerza para los terceros vecinos

phi3rBN=var('phi3rBN',latex_name='\\phi_{3,r}^{BN}')
phi3tiBN=var('phi3tiBN',latex_name='\\phi_{3,ti}^{BN}')
phi3toBN=var('phi3toBN',latex_name='\\phi_{3,to}^{BN}',domain='real')
phi3rNB,phi3tiNB,phi3toNB=phi3rBN,phi3tiBN,phi3toBN
Phi_30__BN=1/sqrt(M_B*M_N)*Matrix([[phi3rBN,0,0],[0,phi3tiBN,0],[0,0,phi3toBN]])
Phi_30__NB=1/sqrt(M_N*M_B)*Matrix([[phi3rNB,0,0],[0,phi3tiNB,0],[0,0,phi3toNB]])

def Phi_3l__BN(alphaprima,alpha,l_1,l_2):
    return U(-angulo(alphaprima,alpha,l_1,l_2))*Phi_30__BN*\
           U(angulo(alphaprima,alpha,l_1,l_2))

def Phi_3l__NB(alphaprima,alpha,l_1,l_2):
    return U(-angulo(alphaprima,alpha,l_1,l_2))*Phi_30__NB*\
           U(angulo(alphaprima,alpha,l_1,l_2))

def D_3l_BN(alphaprima,alpha,l_1,l_2):
    return Phi_3l__BN(alphaprima,alpha,l_1,l_2)*fase(l_1,l_2)

def D_3l_NB(alphaprima,alpha,l_1,l_2):
    return Phi_3l__NB(alphaprima,alpha,l_1,l_2)*fase(l_1,l_2)

#Matriz de fuerza para los cuartos vecinos
phi4rBN=var('phi4rBN',latex_name='\\phi_{4,r}^{BN}')
phi4tiBN=var('phi4tiBN',latex_name='\\phi_{4,ti}^{BN}')
phi4toBN=var('phi4toBN',latex_name='\\phi_{4,to}^{BN}',domain='real')
phi4rNB,phi4tiNB,phi4toNB=phi4rBN,phi4tiBN,phi4toBN

Phi_40__BN=1/sqrt(M_B*M_N)*Matrix([[phi4rBN,0,0],[0,phi4tiBN,0],[0,0,phi4toBN]])
```

```
Phi_40__NB=1/sqrt(M_N*M_B)*Matrix([[phi4rNB,0,0],[0,phi4tiNB,0],[0,0,phi4toNB]])

def Phi_4l__BN(alphaprima,alpha,l_1,l_2):
    return U(-angulo(alphaprima,alpha,l_1,l_2))*Phi_40__BN*\
           U(angulo(alphaprima,alpha,l_1,l_2))

def Phi_4l__NB(alphaprima,alpha,l_1,l_2):
    return U(-angulo(alphaprima,alpha,l_1,l_2))*Phi_40__NB*\
           U(angulo(alphaprima,alpha,l_1,l_2))

def D_4l_BN(alphaprima,alpha,l_1,l_2):
    return Phi_4l__BN(alphaprima,alpha,l_1,l_2)*fase(l_1,l_2)

def D_4l_NB(alphaprima,alpha,l_1,l_2):
    return Phi_4l__NB(alphaprima,alpha,l_1,l_2)*fase(l_1,l_2)

# Finalmente construimos la matriz dinámica "a capas"
# Con la tabla de la celda de codigo anterior comprobamos que para considerar
# hasta los cuartos vecinos es suficiente con l_1,l_2=2

D1BN, D1NB, D2BB, D2NN, D3BN, D3NB, D4BN, D4NB = (matrix(3) for i in range(8))
D01BN, D01NB, D02BB, D02NN, D03BN, D03NB, D04BN, D04NB= (matrix(3) for i in␣
 ↪range(8))
for k in [1,2]:
    for m in [1,2]:
         for i in range(-2,3):
            for j in range(-2,4):
                if (k==1) & bool( distancia(k,m,i,j) == sqrt(3)/3 ):
                    D1BN += D_1l_BN(k,m,i,j)
                    D01BN += Phi_1l__BN(k,m,i,j)

                elif (k==2) & bool( distancia(k,m,i,j) == sqrt(3)/3 ):
                    D1NB += D_1l_NB(k,m,i,j)
                    D01NB += Phi_1l__NB(k,m,i,j)

                elif (k==1) & bool( distancia(k,m,i,j) == 1):
                    D2BB += D_2l_BB(k,m,i,j)
                    D02BB += Phi_2l__BB(k,m,i,j)

                elif (k==2) & bool( distancia(k,m,i,j) == 1):
                    D2NN += D_2l_NN(k,m,i,j)
                    D02NN += Phi_2l__NN(k,m,i,j)

                elif (k==1) & bool( distancia(k,m,i,j) == 2*sqrt(3)/3 ):
                    D3BN += D_3l_BN(k,m,i,j)
                    D03BN += Phi_3l__BN(k,m,i,j)
```

```
                elif (k==2) & bool( distancia(k,m,i,j) == 2*sqrt(3)/3 ):
                    D3NB += D_3l_NB(k,m,i,j)
                    D03NB += Phi_3l__NB(k,m,i,j)

                elif (k==1) & bool( distancia(k,m,i,j) == sqrt(7/3)):
                    D4BN += D_4l_BN(k,m,i,j)
                    D04BN += Phi_4l__NB(k,m,i,j)

                elif (k==2) & bool( distancia(k,m,i,j) == sqrt(7/3)):
                    D4NB += D_4l_NB(k,m,i,j)
                    D04NB += Phi_4l__NB(k,m,i,j)

# Tenemos en cuenta la contribución a la matriz dinámica de los átomos situados␣
 ↪en
# la celdilla 0
D2BB=D2BB-D01BN-D02BB-D03BN
D2NN=D2NN-D01NB-D02NN-D03NB
```

1.1.1 Unas simples comprobaciones para comprobar que he definido bien los tensores de fuerza y las matrices dinámicas:

(Comparando la matriz dinámica para los primeros vecinos del boro con las obtenidas ``manualmente'')

[8]:
```
D_1l_BN(1,2,-1,0)+D_1l_BN(1,2,0,0)+D_1l_BN(1,2,0,1)-D1BN
```

[8]:
$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

1.2 Vamos a fijarnos sólo en las vibraciones transversales fuera de plano

Puesto que las vibraciones fuera de plano son, por cómo hemos constuido la matriz dinámica, independientes de las interplanares, podemos estudiar primero las vibraciones fuera de plano

[9]:
```
D1BN_zz=D1BN[2,2]
D1NB_zz=D1NB[2,2]

D2BB_zz=D2BB[2,2]
D2NN_zz=D2NN[2,2]

D3BN_zz=D3BN[2,2]
D3NB_zz=D3NB[2,2]
D4BN_zz=D4BN[2,2]
D4NB_zz=D4NB[2,2]
```

```
#D_zz=Matrix([[D2BB_zz,D1BN_zz+D3BN_zz+D4BN_zz],
 ↪[D1NB_zz+D3NB_zz+D4BN_zz,D2NN_zz]])
D_zz=Matrix([[D2BB_zz,D1BN_zz+D3BN_zz], [D1NB_zz+D3NB_zz,D2NN_zz]])
#valors_propis_D_zz=D_zz.eigenvalues()
D_zz[0,0]
```

[9]:
$$\frac{\phi_{2,to}^{BB} e^{\left(\frac{1}{2} i \sqrt{3} a q_y + \frac{1}{2} i a q_x\right)}}{M_B} + \frac{\phi_{2,to}^{BB} e^{\left(\frac{1}{2} i \sqrt{3} a q_y - \frac{1}{2} i a q_x\right)}}{M_B} + \frac{\phi_{2,to}^{BB} e^{\left(-\frac{1}{2} i \sqrt{3} a q_y + \frac{1}{2} i a q_x\right)}}{M_B} + \frac{\phi_{2,to}^{BB} e^{\left(-\frac{1}{2} i \sqrt{3} a q_y - \frac{1}{2} i a q_x\right)}}{M_B} + \frac{\phi_{2,to}^{BB} e^{(i a q_x)}}{M_B} + \frac{\phi_{2,to}^{BB} e^{(-i a q_x)}}{M_B} - \frac{3\,\phi_{1,to}^{BN}}{\sqrt{M_B M_N}} - \frac{3\,\phi_{3,to}^{BN}}{\sqrt{M_B M_N}} - \frac{6\,\phi_{2,to}^{BB}}{M_B}$$

1.2.1 Para el punto $\Gamma(k_x = 0, k_y = 0)$

```
from periodictable import C, B, N, constants
u=constants.atomic_mass_constant*10**3 #para que este en CGS (y las const. de
 ↪fuerza en dyn)

omega_Gamma_ZO=830 #cm-1
omega_Gamma_ZA=0

D_Gamma_zz=D_zz.subs(q_x=0,q_y=0) #,(M_B,B.mass*u),(M_N,N.mass*u)])
D_Gamma_zz
```

[10]:
$$\begin{pmatrix} -\frac{3\,\phi_{1,to}^{BN}}{\sqrt{M_B M_N}} - \frac{3\,\phi_{3,to}^{BN}}{\sqrt{M_B M_N}} & \frac{3\,\phi_{1,to}^{BN}}{\sqrt{M_B M_N}} + \frac{3\,\phi_{3,to}^{BN}}{\sqrt{M_B M_N}} \\ \frac{3\,\phi_{1,to}^{BN}}{\sqrt{M_B M_N}} + \frac{3\,\phi_{3,to}^{BN}}{\sqrt{M_B M_N}} & -\frac{3\,\phi_{1,to}^{BN}}{\sqrt{M_B M_N}} - \frac{3\,\phi_{3,to}^{BN}}{\sqrt{M_B M_N}} \end{pmatrix}$$

```
D_Gamma_zz.eigenvalues()
```

[11]:
$$\left[-\frac{6\left(\phi_{1,to}^{BN}+\phi_{3,to}^{BN}\right)}{\sqrt{M_B M_N}}, 0\right]$$

```
Eq1=(D_Gamma_zz.eigenvalues()[0]==omega**2).subs(omega=omega_Gamma_ZO)
#timeit('Eq1=solve(det(D_Gamma_zz-omega**2)==0, omega)')#.
 ↪subs(omega=omega_Gamma_ZO)
Eq1
```

[12]:
$$-\frac{6\left(\phi_{1,to}^{BN}+\phi_{3,to}^{BN}\right)}{\sqrt{M_B M_N}} = 688900$$

```
solEq1=solve(Eq1, phi3toBN); solEq1[0]
```

[13]:
$$\phi_{3,to}^{BN} = -\phi_{1,to}^{BN} - \frac{344450}{3}\sqrt{M_B M_N}$$

1.2.2 Para el punto K $(k_x = 4\pi/(3a),\ k_y = 0)$

```
omega_K_ZO=605 #cm-1
omega_K_ZA=322
D_K_zz=D_zz.subs(q_x=4*pi/(3*a),q_y=0)
D_K_zz.eigenvalues()
#solve(det(D_K_zz-omega**2)==0, omega**2)
```

[14]:
$$\left[-\frac{3\left(M_N\phi_{1,to}^{BN}+M_N\phi_{3,to}^{BN}+3\sqrt{M_BM_N}\phi_{2,to}^{NN}\right)}{\sqrt{M_BM_N}M_N},-\frac{3\left(M_B\phi_{1,to}^{BN}+M_B\phi_{3,to}^{BN}+3\sqrt{M_BM_N}\phi_{2,to}^{BB}\right)}{\sqrt{M_BM_N}M_B}\right]$$

Podemos observar que en el baso del BN, a diferencia del caso del grafeno, obtenemos 2 frecuencias distintas en el punto K debido a que en la base tenemos dos átomos distintos.

[15]:
```
sol=[]
Eq2=(D_K_zz.eigenvalues()[0]==omega**2).subs(omega=omega_K_ZO)
Eq3=(D_K_zz.eigenvalues()[1]==omega**2).subs(omega=omega_K_ZA)
sol.append(solve(Eq2.subs(solEq1),phi2toNN)[0])
sol.append(solve(Eq3.subs(solEq1),phi2toBB)[0])
sol
```

[15]:
$$\left[\phi_{2,to}^{NN}=-\frac{21575}{9}\,M_N,\phi_{2,to}^{BB}=\frac{240766}{9}\,M_B\right]$$

1.2.3 Y para el punto $M(q_x=\pi/a,q_y=\pi/(\sqrt{3}a)$

[16]:
```
omega_M_ZO=635 #cm-1
omega_M_ZA=314

D_M_zz=D_zz.subs(q_x=pi/a,q_y=pi/(sqrt(3)*a))
#D_M_zz.eigenvalues()
```

Podemos simplificar un poco la expresión obtenida para los valores propios en el punto M (simplemente reescribiendo el argumento de la raiz cuadrada)

[17]:
```
omegaM1cuadrado=-4*phi2toBB/M_B-4*phi2toNN/M_N-3/
↪sqrt(M_B*M_N)*(phi1toBN+phi3toBN)-sqrt(M_B*M_N*(phi1toBN-3*phi3toBN)^2+(4*(M_N*phi2toBB-M_B
↪(M_B*M_N)
if bool(D_M_zz.eigenvalues()[0]==omegaM1cuadrado):
    show(omegaM1cuadrado)
```

$$-\frac{3\left(\phi_{1,to}^{BN}+\phi_{3,to}^{BN}\right)}{\sqrt{M_BM_N}}-\frac{4\,\phi_{2,to}^{BB}}{M_B}-\frac{4\,\phi_{2,to}^{NN}}{M_N}-\frac{\sqrt{M_BM_N\left(\phi_{1,to}^{BN}-3\,\phi_{3,to}^{BN}\right)^2+16\left(M_N\phi_{2,to}^{BB}-M_B\phi_{2,to}^{NN}\right)^2}}{M_BM_N}$$

[18]:
```
omegaM2cuadrado=-4*phi2toBB/M_B-4*phi2toNN/M_N-3/
↪sqrt(M_B*M_N)*(phi1toBN+phi3toBN)+sqrt(M_B*M_N*(phi1toBN-3*phi3toBN)^2+(4*(M_N*phi2toBB-M_B
↪(M_B*M_N)
if bool(D_M_zz.eigenvalues()[1]==omegaM2cuadrado):
    show(omegaM2cuadrado)
```

$$-\frac{3\left(\phi_{1,to}^{BN}+\phi_{3,to}^{BN}\right)}{\sqrt{M_BM_N}}-\frac{4\,\phi_{2,to}^{BB}}{M_B}-\frac{4\,\phi_{2,to}^{NN}}{M_N}+\frac{\sqrt{M_BM_N\left(\phi_{1,to}^{BN}-3\,\phi_{3,to}^{BN}\right)^2+16\left(M_N\phi_{2,to}^{BB}-M_B\phi_{2,to}^{NN}\right)^2}}{M_BM_N}$$

Podemos comprobar que en el caso que fuesen los átomos identicos obtenemos las mismas expresiones que en Falkowsky

[19]:
```
D_M_zz.subs(M_B=M_N, phi2toBB=phi2toNN).eigenvalues()
```

[19]:

$$\left[-\frac{2\left(\phi_{1,to}^{BN}+4\,\phi_{2,to}^{NN}+3\,\phi_{3,to}^{BN}\right)}{M_N},-\frac{4\left(\phi_{1,to}^{BN}+2\,\phi_{2,to}^{NN}\right)}{M_N}\right]$$

[20]:
```
Eq5=(omegaM1cuadrado==omega_M_ZO**2)
Eq6=(omegaM2cuadrado==omega_M_ZA**2)
#sol.append(solve(((Eq5-Eq6)**2).subs(sol[0], sol[1], solEq1[0],M_N=N.mass,␣
 ↪M_B=B.mass), phi1toBN)[1])
```

[21]:
```
sol1=(phi1toBN==n(solve(((Eq6-Eq5)**2).subs(solEq1), phi1toBN)[0].subs(sol[0],␣
 ↪sol[1]).subs(M_B=B.mass, \
    M_N=N.mass).rhs()))
sol1
```

[21]:
$$\phi_{1,to}^{BN} = \left(-1.36116402436406 \times 10^{6}\right)$$

[22]:
```
sol2=sol[0].subs(M_N=N.mass)
sol2
```

[22]:
$$\phi_{2,to}^{NN} = (-33577.1725)$$

[23]:
```
sol3=sol[1].subs(M_B=B.mass)
sol3
```

[23]:
$$\phi_{2,to}^{BB} = 289213.46955555555$$

[24]:
```
sol4=solEq1[0].subs(M_B=B.mass, M_N=N.mass).subs(sol1)
sol4
```

[24]:
$$\phi_{3,to}^{BN} = (-51717.6151722642)$$

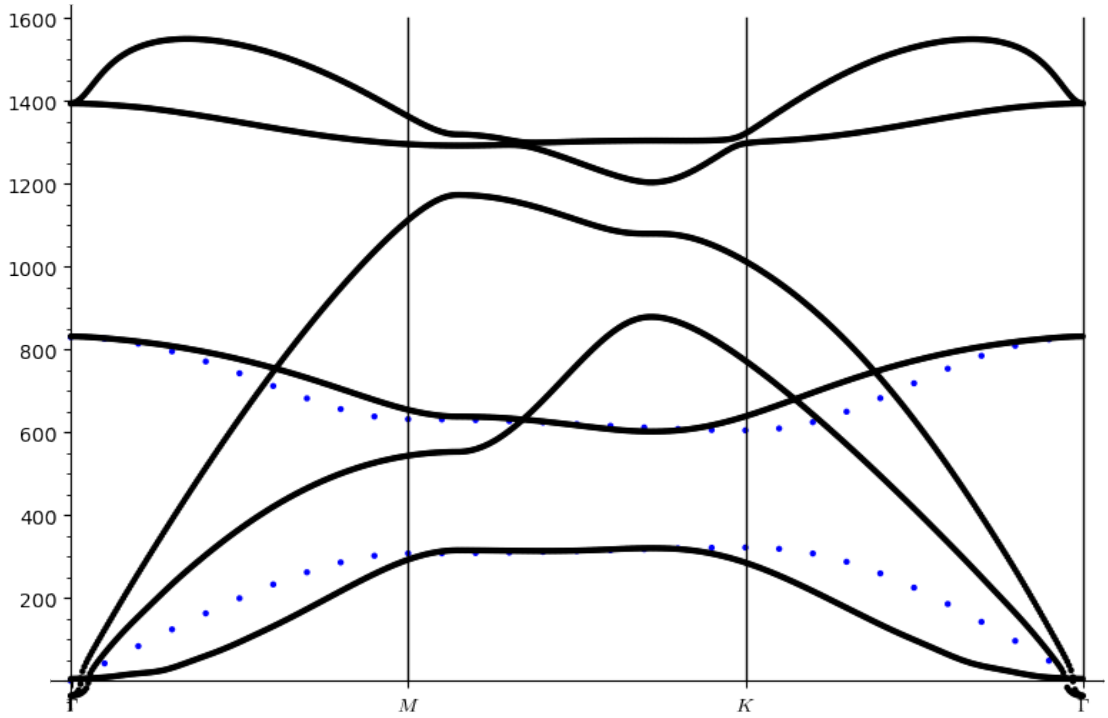
[25]:
```
from numpy import arange
```

[26]:
```
[real_part(n(sqrt(D_zz.subs(sol1, sol2, sol3, sol4, M_B=B.mass, \
        M_N=N.mass, a=1, q_x=n(x), q_y=n(y)).simplify_full().
 ↪eigenvalues()[1]))) for x in [0,4*pi/3] for y in [0]]
```

[26]:
$$[830.000000000000, 605.000000000000]$$

[27]:
```
%%time
from pylab import loadtxt
dades=loadtxt("freq.dat.txt")
show(\
list_plot(
    [real_part(n(sqrt(D_zz.subs(sol1, sol2, sol3, sol4, M_B=B.mass, \
        M_N=N.mass, a=1, q_x=n(x*pi), q_y=n(x*pi/sqrt(3))).simplify_full().
 ↪eigenvalues()[1]))) \
        for x in arange(0,1,0.1)] +\
         [real_part(n(sqrt(D_zz.subs(sol1, sol2, sol3, sol4, M_B=B.mass, \
M_N=N.mass, a=1, q_x=n(pi*(1+x/3)), q_y=n(pi/sqrt(3)*(1-x))).simplify_full().
 ↪eigenvalues()[1]))) \
```

```
        for x in arange(0,1,0.1)]+\
         [real_part(n(sqrt(D_zz.subs(sol1, sol2, sol3, sol4, M_B=B.mass, \
M_N=N.mass, a=1, q_x=n(4*pi/3*(1-x)), q_y=0).simplify_full().
 ↪eigenvalues()[1]))) \
        for x in arange(0,1,0.1)]) + \
list_plot(
    [real_part(n(sqrt(D_zz.subs(sol1, sol2, sol3, sol4, M_B=B.mass, \
        M_N=N.mass, a=1, q_x=n(x*pi), q_y=n(x*pi/sqrt(3))).simplify_full().
 ↪eigenvalues()[0]))) \
        for x in arange(0,1,0.1)]+
          [real_part(n(sqrt(D_zz.subs(sol1, sol2, sol3, sol4, M_B=B.mass, \
M_N=N.mass, a=1,q_x=n(pi*(1+x/3)), q_y=n(pi/sqrt(3)*(1-x))).simplify_full().
 ↪eigenvalues()[0]))) \
        for x in arange(0,1,0.1)]+\
         [real_part(n(sqrt(D_zz.subs(sol1, sol2, sol3, sol4, M_B=B.mass, \
M_N=N.mass, a=1, q_x=n(4*pi/3*(1-x)), q_y=0).simplify_full().
 ↪eigenvalues()[0]))) \
        for x in arange(0,1,0.1)]) \
     +line([(10,0),(10,1600)], color="black")+line([(20,0),(20,1600)],␣
 ↪color="black")\
     +line([(30,0),(30,1600)], color="black", ticks=[[0.05,10,20,30], None], \
        tick_formatter = [[r'$\Gamma$', '$M$', '$K$', r'$\Gamma$'], None])+\
points(zip(dades[:,0]/523*30, dades[:,1]), color="black")\
     ,figsize=8)
```



```
CPU times: user 44 s, sys: 493 ms, total: 44.5 s
Wall time: 41.7 s
```

2 Para la vibraciones interplanares

[28]:
```
D1BN_xy=D1BN.matrix_from_rows_and_columns([0,1],[0,1])
D1NB_xy=D1NB.matrix_from_rows_and_columns([0,1],[0,1])

D2BB_xy=D2BB.matrix_from_rows_and_columns([0,1],[0,1])
D2NN_xy=D2NN.matrix_from_rows_and_columns([0,1],[0,1])

D3BN_xy=D3BN.matrix_from_rows_and_columns([0,1],[0,1])
D3NB_xy=D3NB.matrix_from_rows_and_columns([0,1],[0,1])
#D4BN_xy=D4BN.matrix_from_rows_and_columns([0,1],[0,1])
#D4NB_xy=D4NB.matrix_from_rows_and_columns([0,1],[0,1])
```

[29]:
```
D_xy=block_matrix([[D2BB_xy, D1BN_xy+D3BN_xy],[D1NB_xy+D3NB_xy, D2NN_xy]])
```

2.0.1 Para el punto $\Gamma(k_x = 0, k_y = 0)$

[30]:
```
D_Gamma_xy=D_xy.subs(q_x=0,q_y=0)
D_Gamma_xy.factor()
```

[30]:
$$\begin{pmatrix} -\frac{3\left(\phi_{1,r}^{BN}+\phi_{1,ti}^{BN}+\phi_{3,r}^{BN}+\phi_{3,ti}^{BN}\right)}{2\sqrt{M_B M_N}} & 0 & \frac{3\left(\phi_{1,r}^{BN}+\phi_{1,ti}^{BN}+\phi_{3,r}^{BN}+\phi_{3,ti}^{BN}\right)}{2\sqrt{M_B M_N}} & 0 \\ 0 & -\frac{3\left(\phi_{1,r}^{BN}+\phi_{1,ti}^{BN}+\phi_{3,r}^{BN}+\phi_{3,ti}^{BN}\right)}{2\sqrt{M_B M_N}} & 0 & \frac{3\left(\phi_{1,r}^{BN}+\phi_{1,ti}^{BN}+\phi_{3,r}^{BN}+\phi_{3,ti}^{BN}\right)}{2\sqrt{M_B M_N}} \\ \frac{3\left(\phi_{1,r}^{BN}+\phi_{1,ti}^{BN}+\phi_{3,r}^{BN}+\phi_{3,ti}^{BN}\right)}{2\sqrt{M_B M_N}} & 0 & -\frac{3\left(\phi_{1,r}^{BN}+\phi_{1,ti}^{BN}+\phi_{3,r}^{BN}+\phi_{3,ti}^{BN}\right)}{2\sqrt{M_B M_N}} & 0 \\ 0 & \frac{3\left(\phi_{1,r}^{BN}+\phi_{1,ti}^{BN}+\phi_{3,r}^{BN}+\phi_{3,ti}^{BN}\right)}{2\sqrt{M_B M_N}} & 0 & -\frac{3\left(\phi_{1,r}^{BN}+\phi_{1,ti}^{BN}+\phi_{3,r}^{BN}+\phi_{3,ti}^{BN}\right)}{2\sqrt{M_B M_N}} \end{pmatrix}$$

[31]:
```
D_Gamma_xy.eigenvalues()
```

[31]:
$$\left[-\frac{3\left(\phi_{1,r}^{BN}+\phi_{1,ti}^{BN}+\phi_{3,r}^{BN}+\phi_{3,ti}^{BN}\right)}{\sqrt{M_B M_N}}, -\frac{3\left(\phi_{1,r}^{BN}+\phi_{1,ti}^{BN}+\phi_{3,r}^{BN}+\phi_{3,ti}^{BN}\right)}{\sqrt{M_B M_N}}, 0, 0\right]$$

2.1 Nota: para el punto K y M, obtengo los valores propios suponiendo que en vez de una base diatomica con 2 átomos de diferentes elementos los átomos de la base son iguales

[32]:
```
#omega_K_ZO=605 #cm-1
#omega_K_ZA=322
D_K_xy=D_xy.subs(q_x=4*pi/(3*a),q_y=0,phi2rNN=phi2rBB,␣
↪phi2tiNN=phi2tiBB,M_N=M_B)
D_K_xy.eigenvalues()
#solve(det(D_K_xy-omega**2)==0, omega)
```

[32]:

$$\left[-\frac{3\left(2\,\phi_{1,ti}^{BN}+3\,\phi_{2,r}^{BB}+3\,\phi_{2,ti}^{BB}+2\,\phi_{3,ti}^{BN}\right)}{2\,M_B}, -\frac{3\left(2\,\phi_{1,r}^{BN}+3\,\phi_{2,r}^{BB}+3\,\phi_{2,ti}^{BB}+2\,\phi_{3,r}^{BN}\right)}{2\,M_B}, -\frac{3\left(\phi_{1,r}^{BN}+\phi_{1,ti}^{BN}+3\,\phi_{2,r}^{BB}+3\,\phi_{2,ti}^{BB}+\phi_{3,r}^{BN}+\phi_{3,ti}^{BN}\right)}{2\,M_B}, -\frac{3\left(\phi_{1,r}^{BN}+\phi_{1,ti}^{BN}+3\right.}{}\right.$$

2.1.1 Y para el punto $M(q_x = \pi/a, q_y = \pi/(\sqrt{3}a)$

[33]:
```
#omega_M_ZO=635 #cm-1
#omega_M_ZA=314
D_M_xy=D_xy.subs(q_x=pi/a,q_y=pi/(sqrt(3)*a),phi2rNN=phi2rBB,␣
↪phi2tiNN=phi2tiBB,M_N=M_B)
```

[34]:
```
D_M_xy.eigenvalues()
```

[34]:
$$\left[-\frac{3\,\phi_{1,r}^{BN}+\phi_{1,ti}^{BN}+2\,\phi_{2,r}^{BB}+6\,\phi_{2,ti}^{BB}}{M_B}, -\frac{\phi_{1,r}^{BN}+3\,\phi_{1,ti}^{BN}+6\,\phi_{2,r}^{BB}+2\,\phi_{2,ti}^{BB}}{M_B}, -\frac{2\,\phi_{1,r}^{BN}+6\,\phi_{2,r}^{BB}+2\,\phi_{2,ti}^{BB}+3\,\phi_{3,r}^{BN}+3\,\phi_{3,ti}^{BN}}{M_B}, -\frac{2\,\phi_{1,ti}^{BN}+2\,\phi_{2,r}^{BB}+6\,\phi_{2,ti}^{BB}+3\,\phi_{3,r}^{BN}+3}{M_B}\right.$$

[ ]:
```

```