Anexo: Códigos python

0.1. Programa principal

0.2

```
# -*- coding: utf-8 -*-
"""
Created on Mon Apr 24 14:39:46 2023

@author: pablo
"""

import numpy as np
import math as ma
import numpy.linalg as la
from scipy.linalg import eig,eigh
from scipy.interpolate import make_interp_spline
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D #3D plots
#import time

#start_time = time.time()


# Datos de la red

nAx=4#Num de celdas unidad en el sentido del vector primitivo ax
nAy=1#Idem para ay. Para una cadena monoat mica dimerizada nAy=1
nAz=1 #Num de planos x-y.
ax=np.array([0,1,0]) #Vector ax de la red directa.
```

```
ay=np.array([0,1,0]) #Vector ay de la red directa
#Grafeno: ax=[1.42*1.5,1.42*np.sqrt(3)/2,0]  ay=[1.42*1.5,-1.42*np.sqrt(3)/2,0]
#Red cuadrada: ax=[1,0,0] ay=[0,1,0]
#Cadena monoat mica dimerizada: ax=[1,0,0] ay=[0,0,0]

ZstepA=0 #separaci n entre planos x-y
az=np.array([0,0,ZstepA]) #Vector separaci n entre dos capas x-y
ang=0 # ngulo de rotaci n del plano superior respecto al inferior (en grados)

if nAx==1:
    ax=np.array([0,0,0])

if nAz==1:
    az=np.array([0,0,0])

if nAy==1:
    ay=np.array([0,0,0])

nAu=1 #N mero de t por c.u.
#Grafeno y la cadena monoat. dim: nAu=2
U=np.array([[-0.27,0,0],[0.27,0,0]]) #Matriz del motivo en la c.u.
#Grafeno: [[-1.42/2,0,0],[1.42/2,0,0]]
#Cadena monoat. dim. sin prop. topolog. especiales [[-0.23,0,0],[0.23,0,0]]
#Cadena monoat. dim con prop. topolog. especiales [[-0.27,0,0],[0.27,0,0]]



nAxy=nAx*nAy #num total de c.u. por capa x-y
nAxyz=nAxy*nAz #num total de c.u.
nA=nAxyz*nAu #N mero total de tomos

if nAu==1:                  #Si s lo hubiese un t . por c.u. lo consideramos centrado
    U=np.array([0,0,0])



#Datos de las interacciones at micas

e0=0.0;t=2.8 #Energ a on-site e integral de salto
V=0.5 #Energ a de enlace de VdW
```

```
alpha=3
delta=0.01 #Diferencial de energ a sobre el que calcular la DOS
#Para g ficas precisas (con picos): delta=0.0005
#Para g ficas suaves: delta=0.01


#Ordenaci n y posici n de los  tomos :
R=np.zeros([nA,3])
for iz in range (nAz):
    for iy in range(nAy):
        for ix in range(nAx):
            for iu in range(nAu):
                R[iu+(ix+iy*nAx+iz*nAxy)*nAu]=U[iu]+(ix+1)*ax+(iy+1)*ay+(iz+1)*az
#Los  tomos  se ordenan agrupados por celda unidad en el orden en el que se
#introduce el motivo. Se recogen las posiciones de todos los at. de la c.u.
#Se avanza a la siguiente c.u. en el sentido ax. Al acabar la hilera de c.u.
#en ese sentido se repite para la siguiente hilera segun ay.
#Se reitera para el plano superior del mismo modo.


#Centramos los planos x-y en el origen de coordenadas
d=(ax*(nAx-1)+ay*(nAy-1))/2
for ia in range (nA):
    R[ia]=R[ia]-d


#Rotaci n del plano superior respecto el inferior
theta=(ang*ma.pi)/180
A=np.array([[ma.cos(theta),-ma.sin(theta),0],[ma.sin(theta),ma.cos(theta),0],[0,0,1]])
if nAz>=2:
    for ir in range (nAxy*nAu):
        R[nAxy*nAu+ir]=np.dot(A,R[nAxy*nAu+ir])


#Volvemos a colocarlo todo
for ia in range (nA):
    R[ia]=R[ia]+d
```

```
    #Datos de la red directa y del motivo
if nAxy>=2:
    print('Vectores de la red directa:')
    if nAx>=2:
        print('ax=',ax)
    if nAy>=2:
        print('ay=',ay)
if nAu>=2:
    print()
    print('Motivo de la red:')
    print('U=',U)



#Representaci n de las posiciones at micas
fig = plt.figure(figsize=(6,6)) #Espacio de representaci n en 3d
axe = fig.add_subplot(111, projection='3d') #Ejes
axe.scatter(R[:,0],R[:,1],R[:,2])
axe.set_title("Posiciones at micas")
axe.set_xlabel("X")
axe.set_ylabel("Y")
axe.set_zlabel("Z")
plt.show()
if nAz>=2:
    print(' ngulo de rotaci n=',ang,'  ')

# Hamiltoniano y matriz de solapes
Smat=np.zeros((nA,nA),complex)
Hmat=np.zeros((nA,nA),complex)

for iF in range(nA):
  Smat[iF,iF]=1.0 #Consideramos que cada orbital s lo solapa consigo mismo

for iF in range(nA): #La interacci n de los electrones con cada tomo con su n cleo
    #por la energ a on-site, mientras que su interacci n con otros n cleos decae ex
    for jF in range(nA):
        if iF!=jF:
            if R[iF,2]==R[jF,2]:
                Hmat[iF,jF]=-t*(np.exp(-alpha*np.sqrt(((R[iF,0]-R[jF,0])**2)+((R[iF,1]-
```

```
            if R[iF,2]!=R[jF,2]:
                Hmat[iF,jF]=-V*(np.exp(-alpha*np.sqrt(((R[iF,0]-R[jF,0])**2)+((R[iF,1]-
    Hmat[iF,iF]=e0


# Autovalores y autovectores
Eval,Evec=eig(Hmat,Smat)


ind=np.argsort(np.real(Eval))
Eval=np.real(Eval[ind])

# Autovalores y autovectores
Eval,Evec=eig(Hmat,Smat)


ind=np.argsort(np.real(Eval))
Eval=np.real(Eval[ind])


# Print and plot eigenvalues
#print(Eval)
plt.plot(Eval,'ro')
plt.title("Autovalores python")
plt.xlabel('Autovalores')
plt.ylabel('Energ a')
plt.show()

#C lculo de los vectores k bidimensionales
kvec=np.zeros((nA,2))
incrkx=np.array([0,0])
incrky=np.array([0,0])

if nAx>1:
    if ax[0]==0:
        if  ax[1]==0:
            incrkx=np.array([0,0])
        if  ax[1]!=0:
```

```
            incrkx=np.array([0,(np.pi/(ax[1]*(nAx+1)))])
    if ax[1]==0:
        if  ax[0]!=0:
            incrkx=np.array([(np.pi/(ax[0]*(nAx+1))),0])
    if  ax[0]!=0 and ax[1]!=0:
        normax=la.norm(np.array([ax[0],ax[1]]))
        incrkx=(np.pi/((nAx+1)*(normax*normax)))*np.array([ax[0],ax[1]])

if nAy>1:
    if ay[0]==0:
        if  ay[1]==0:
            incrky=np.array([0,0])
        if  ay[1]!=0:
            incrky=np.array([0,(np.pi/(ay[1]*(nAy+1)))])
    if ay[1]==0:
        if  ay[0]!=0:
            incrky=np.array([(np.pi/(ay[0]*(nAy+1))),0])
    if  ay[0]!=0 and ay[1]!=0:
        normay=la.norm(np.array([ay[0],ay[1]]))
        incrky=(np.pi/((nAy+1)*(normay*normay)))*np.array([ay[0],ay[1]])


for iz in range (nAz):
    for iy in range(nAy):
        for ix in range(nAx):
            for iu in range(nAu):
                kvec[iu+(ix+iy*nAx+iz*nAxy)*nAu]=(ix+1)*incrkx+(iy+1)*incrky


#Cálculo de los autovectores de H
coef=np.zeros(nA) #La fila j será un autovector asociado al k dado por la fila j de k
#for ik in range (nA):


for iz in range (nAz):
    for iy in range(nAy):
        for ix in range(nAx):
            for iu in range(nAu):
```

```
                coef[iu+(ix+iy*nAx+iz*nAxy)*nAu]=np.sqrt(2/(nA+1))*np.sin(np.dot(np.ar
```

```
#    Clculo     de las autoenerg as
E=np.zeros(nA)
for ie in range(nA):
    E[ie]=np.dot(Hmat[ie,:],coef)/coef[ie]

"""
eyk=np.zeros((nA,4))
for ie in range(nA):
    eyk[ie,1]=np.pi*(ie+1)/(nA+1) #Valor de k
    coef=np.zeros(nA) #Autovector asociado a k
    for ic in range(nA):
        coef[ic]=np.sqrt(2/(nA+1))*np.sin((ic+1)*eyk[ie,1])
    eyk[ie,0]=np.dot(Hmat[0,:],coef)/coef[0] #Autoenerg a asociada a k"""
```

0.2. Primer programa auxiliar: Grafeno con bordes zig-zag o armchair

0.2

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jun 29 12:53:34 2023

@author: pablo
"""

import numpy as np
import math as ma
import numpy.linalg as la
from scipy.linalg import eig,eigh
from scipy.interpolate import make_interp_spline
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D #3D plots
```

```
#GRAFENO
nAx=nAy=10
ax=np.array([1.42*1.5,1.42*np.sqrt(3)/2])
ay=np.array([1.42*1.5,-1.42*np.sqrt(3)/2])
nAu=2
U=np.array([[-1.42/2,0],[1.42/2,0]])
e0=0.0 #Energa on-site
alpha=3.6
t=3.3*np.exp(alpha*1.42)  #Integral de salto
V=t #Correcci n a la energa de enlace de VdW frente al enlace covalente
delta=0.15
DELTA=0.02



#Variables auxiliares
nAxy=nAx*nAy #N mero total de c.u. por capa x-y
nA=nAxy*nAu #N mero total de  tomos



#Ordenaci n y posici n de los  tomos :
R=np.zeros([nA,2])
for iy in range(nAy):
    for ix in range(nAx):
            for iu in range(nAu):
                R[iu+(ix+iy*nAx)*nAu]=U[iu]+ix*ax+iy*ay

#Escala de los ejes en la representaci n gr fica:
#Calculamos el valor m ximo y m nimo que alcanza la posici n de los  tomos  en
#cada eje para definir el espacio de representaci n gr fica de la red
Rxmin=np.min(R[:,0])
Rxmax=np.max(R[:,0])
Rymin=np.min(R[:,1])
Rymax=np.max(R[:,1])

if Rxmin<Rymin:
    Rmin=Rxmin
```

```
else:
    Rmin=Rymin

if Rxmax<Rymax:
    Rmax=Rymax
else:
    Rmax=Rxmax



## GRAFENO ZIGZAG PURO
TITULO="GRAFENO bordes zigzag"

x1=[(0.8*nAx-1)*ax[0],(0.8*nAx-1)*ax[0]]
y1=[Rymin,Rymax]

x2=np.array([0,0])
y2=[Rymin,Rymax]

x3=[Rxmin,Rxmax/2]
y3=[(nAx-1.5)*ax[1],(nAx-1.5)*ax[1]]

x4=[Rxmin,Rxmax/2]
y4=[(nAy-1.5)*ay[1],(nAy-1.5)*ay[1]]

#Vista area

plt.figure(figsize=(6,6))
plt.plot(R[:,0],R[:,1],'b.')
plt.plot(x1, y1,'r', linewidth=1)
plt.plot(x2, y2,'r', linewidth=1)
plt.plot(x3, y3,'r', linewidth=1)
plt.plot(x4, y4,'r', linewidth=1)
plt.suptitle(TITULO)
plt.xlim(Rmin-2,Rmax+2)
plt.ylim(Rmin-2,Rmax+2)
#plt.title("Vista area para "+str(ang)+r"$^o$")
plt.xlabel('X')
plt.ylabel('Y')
```

```
#plt.savefig(TITULO+"_aut_"+str(ang)+".png")
plt.show()



R1=np.array([0,0])
for i in range (nA):
    if R[i,0]<((0.8*nAx-1)*ax[0]):
        R1=np.vstack([R1,R[i,:]])
R1 = np.delete(R1, np.s_[0], axis=0)



R2=np.array([0,0])
for i in range (np.shape(R1)[0]):
    if R1[i,0]>(0):
        R2=np.vstack([R2,R1[i,:]])
R2 = np.delete(R2, np.s_[0], axis=0)

R3=np.array([0,0])
for i in range (np.shape(R2)[0]):
    if R2[i,1]<((nAx-1.5)*ax[1]):
        R3=np.vstack([R3,R2[i,:]])
R3 = np.delete(R3, np.s_[0], axis=0)

R4=np.array([0,0])
for i in range (np.shape(R3)[0]):
    if R3[i,1]>((nAy-1.5)*ay[1]):
        R4=np.vstack([R4,R3[i,:]])
R4 = np.delete(R4, np.s_[0], axis=0)

Rzig=R4
nAz=np.shape(Rzig)[0]



plt.figure(figsize=(6,6))
plt.plot(Rzig[:,0],Rzig[:,1],'b.')
plt.plot(x1, y1,'r', linewidth=1)
plt.plot(x2, y2,'r', linewidth=1)
```

```
plt.plot(x3, y3,'r', linewidth=1)
plt.plot(x4, y4,'r', linewidth=1)
plt.suptitle(TITULO)
plt.xlim(Rmin-2,Rmax+2)
plt.ylim(Rmin-2,Rmax+2)
#plt.title("Vista area para "+str(ang)+r"$^o$")
plt.ylabel('Y')
#plt.savefig(TITULO+"_aut_"+str(ang)+".png")
plt.show()
```

```
#GRAFENO ARMCHAIR PURO
TITULO="GRAFENO bordes armchair"

#Ecuacion de la recta y=mx+n con m=(ax+2*U[0,:])[1]/(ax+2*U[0,:])[0]
#que pasa por el punto x=(nAx-1)*ax[0], y=(nAx-1)*ax[1]
m1=(ax+2*U[0,:])[1]/(ax+2*U[0,:])[0]
n1=(nAx-1)*ax[1]-m1*(nAx-1)*ax[0]
xa1=[Rxmin,Rxmax]
ya1=[m1*(Rxmin)+n1,m1*(Rxmax)+n1]

#Ecuacion de la recta y=mx+n con m=(ax+2*U[0,:])[1]/(ax+2*U[0,:])[0]
#que pasa por el punto x=(nAx-1)*ay[0], y=(nAx-1)*ay[1]
m2=(ax+2*U[0,:])[1]/(ax+2*U[0,:])[0]
n2=(nAx-1)*ay[1]-m2*(nAx-1)*ay[0]
xa2=[Rxmin,Rxmax]
ya2=[m2*(Rxmin)+n2,m2*(Rxmax)+n2]

#Ecuacion de la recta y=mx+n con m=(ay+2*U[0,:])[1]/(ay+2*U[0,:])[0]
#que pasa por el punto x=(nAx-1)*ax[0], y=(nAx-1)*ax[1]
m3=(ay+2*U[0,:])[1]/(ay+2*U[0,:])[0]
n3=(nAx-1)*ax[1]-m3*(nAx-1)*ax[0]
xa3=[Rxmin,Rxmax]
ya3=[m3*(Rxmin)+n3,m3*(Rxmax)+n3]

#Ecuacion de la recta y=mx+n con m=(ay+2*U[0,:])[1]/(ay+2*U[0,:])[0]
```

```
#que pasa por el punto x=(nAx-1)*ay[0], y=(nAx-1)*ay[1]
m4=(ay+2*U[0,:])[1]/(ay+2*U[0,:])[0]
n4=(nAx-1)*ay[1]-m4*(nAx-1)*ay[0]
xa4=[Rxmin,Rxmax]
ya4=[m4*(Rxmin)+n4,m4*(Rxmax)+n4]

#xa4=[Rxmin,Rxmax/2]
#ya4=[(nAy-1.5)*ay[1],(nAy-1.5)*ay[1]]

#Vista area

plt.figure(figsize=(6,6))
plt.plot(R[:,0],R[:,1],'b.')
plt.plot(xa1, ya1,'r', linewidth=1)
plt.plot(xa2, ya2,'r', linewidth=1)
plt.plot(xa3, ya3,'r', linewidth=1)
plt.plot(xa4, ya4,'r', linewidth=1)
plt.suptitle(TITULO)
plt.xlim(Rmin-2,Rmax+2)
plt.ylim(Rmin-2,Rmax+2)
#plt.title("Vista area para "+str(ang)+r"$^o$")
plt.xlabel('X')
plt.ylabel('Y')
#plt.savefig(TITULO+"_aut_"+str(ang)+".png")
plt.show()



#Puntos por debajo de la recta y=m1x+n1
Ra1=np.array([0,0])
for i in range (nA):
    if R[i,1]<m1*R[i,0]+n1:
        Ra1=np.vstack([Ra1,R[i,:]])
Ra1 = np.delete(Ra1, np.s_[0], axis=0)



#Puntos por encima de la recta y=m2x+n2
Ra2=np.array([0,0])
```

```
for i in range (np.shape(Ra1)[0]):
    if Ra1[i,1]>m2*Ra1[i,0]+n2:
        Ra2=np.vstack([Ra2,Ra1[i,:]])
Ra2 = np.delete(Ra2, np.s_[0], axis=0)


#Puntos por debajo de la recta y=m3x+n3
Ra3=np.array([0,0])
for i in range (np.shape(Ra2)[0]):
    if Ra2[i,1]<m3*Ra2[i,0]+n3:
        Ra3=np.vstack([Ra3,Ra2[i,:]])
Ra3 = np.delete(Ra3, np.s_[0], axis=0)


#Puntos por debajo de la recta y=m4x+n4
Ra4=np.array([0,0])
for i in range (np.shape(Ra3)[0]):
    if Ra3[i,1]>m4*Ra3[i,0]+n4:
        Ra4=np.vstack([Ra4,Ra3[i,:]])
Ra4 = np.delete(Ra4, np.s_[0], axis=0)

Rarm=Ra4
nAa=np.shape(Rarm)[0]


plt.figure(figsize=(6,6))
plt.plot(Rarm[:,0],Rarm[:,1],'b.')
plt.plot(xa1, ya1,'r', linewidth=1)
plt.plot(xa2, ya2,'r', linewidth=1)
plt.plot(xa3, ya3,'r', linewidth=1)
plt.plot(xa4, ya4,'r', linewidth=1)
plt.suptitle(TITULO)
plt.xlim(Rmin-2,Rmax+2)
plt.ylim(Rmin-2,Rmax+2)
#plt.title("Vista area para "+str(ang)+r"$^o$")
plt.ylabel('Y')
#plt.savefig(TITULO+"_aut_"+str(ang)+".png")
plt.show()
```

```
print('N mero de tomos en grafeno zigzag:')
print(nAz)
print('N mero de tomos en grafeno armchair:')
print(nAa)


## HAMILTONIANO Y AUTOENERGIAS


# Hamiltoniano y matriz de solapes
Sz=np.zeros((nAz,nAz),complex)
Hz=np.zeros((nAz,nAz),complex)

Sa=np.zeros((nAa,nAa),complex)
Ha=np.zeros((nAa,nAa),complex)



for iF in range(nAz):
  Sz[iF,iF]=1.0 #Consideramos que cada orbital s lo solapa consigo mismo

for iF in range(nAa):
  Sa[iF,iF]=1.0



for iF in range(nAz): #La interacci n de los electrones con cada tomo con
    #su n cleo viene prefijada por la energ a on-site, mientras que su
    #interacci n con otros n cleos decae exponencialmente con la distancia
    for jF in range(nAz):
        Hz[iF,jF]=-t*(np.exp(-alpha*np.sqrt(((Rzig[iF,0]-Rzig[jF,0])**2)+((Rzig[iF,1]-
    Hz[iF,iF]=e0

for iF in range(nAa): #La interacci n de los electrones con cada tomo con
    #su n cleo viene prefijada por la energ a on-site, mientras que su
```

```
    #interacci n con otros n cleos decae exponencialmente con la distancia
    for jF in range(nAa):
        Ha[iF,jF]=-t*(np.exp(-alpha*np.sqrt(((Rarm[iF,0]-Rarm[jF,0])**2)+((Rarm[iF,1]-R
    Ha[iF,iF]=e0



# Autovalores y autovectores
Evalz,Evecz=eig(Hz,Sz)

Evala,Eveca=eig(Ha,Sa)



indz=np.argsort(np.real(Evalz))
Evalz=np.real(Evalz[indz])



inda=np.argsort(np.real(Evala))
Evala=np.real(Evala[inda])


#Gr fica de autovalores
plt.figure(figsize=(6,6))
plt.plot(Evalz,'b.',markersize=1, label= "zigzag")
plt.plot(Evala,'r.',markersize=1,label= "armchair")
#plt.suptitle(TITULO)
#plt.title("Autovalores para "+str(ang)+r"$^o$")
plt.xlabel('Autovalores')
plt.ylabel('Energ a')
plt.legend()
#plt.savefig(TITULO+"_aut_"+str(ang)+".png")
plt.show()


#DOSz suave para el grafeno zigzag
def DOSz(E): #funcion DOS
```

```
    dosz=0
    for js in range(nAz):
        dosz=dosz+(1/np.pi)*delta/(((E-Evalz[js])**2)+delta**2)
    return dosz

#DOS suave para el grafeno armchair
def DOSa(E): #funcion DOS
    dosa=0
    for js in range(nAa):
        dosa=dosa+(1/np.pi)*delta/(((E-Evala[js])**2)+delta**2)
    return dosa


#DOS con picos para el grafeno zigzag
def DOSpz(E): #funcion DOSp
    dospz=0
    for js in range(nAz):
        dospz=dospz+(1/np.pi)*DELTA/(((E-Evalz[js])**2)+DELTA**2)
    return dospz



#DOS con picos para el grafeno armchai
def DOSpa(E): #funcion DOS
    dospa=0
    for js in range(nAa):
        dospa=dospa+(1/np.pi)*DELTA/(((E-Evala[js])**2)+DELTA**2)
    return dospa



#Grfica DOS
ejeenergia=np.linspace(min(min(Evalz),min(Evala))-(2*delta), max(max(Evalz),max(Evala)-
plt.plot(ejeenergia,[DOSz(i) for i in ejeenergia], 'b', linewidth=1, label= "zigzag") #
plt.plot(ejeenergia, [DOSa(i) for i in ejeenergia], 'r' , linewidth=1, label= "armchai
#plt.suptitle(TITULO)
#plt.title("Densidad de estados "+str(ang)+r"$^o$")
```

```
plt.xlabel('E')
plt.ylabel('DOS(E)')
plt.axhline(0, color="black")# Establecer el color de los ejes.
plt.axvline(0, color="black")
#plt.xlim(min(Eval)-(2*delta), max(Eval)+(2*delta))# Limitar los valores de los ejes.
plt.legend()
#plt.savefig(TITULO+"_DOS_"+str(ang)+".png")# Guardar grfico como imgen PNG.
plt.show() # Mostrarlo.
```

```
#Grfica DOS
ejeenergia=np.linspace(min(min(Evalz),min(Evala))-(2*DELTA), max(max(Evalz),max(Evala)-
plt.plot(ejeenergia,[DOSpz(i) for i in ejeenergia], 'b', linewidth=1, label= "zigzag")
plt.plot(ejeenergia, [DOSpa(i) for i in ejeenergia], 'r', linewidth=1, label= "armcha
#plt.suptitle(TITULO)
#plt.title("Densidad de estados "+str(ang)+r"$^o$")
plt.xlabel('E')
plt.ylabel('DOS(E)')
plt.axhline(0, color="black")# Establecer el color de los ejes.
plt.axvline(0, color="black")
#plt.xlim(min(Eval)-(2*delta), max(Eval)+(2*delta))# Limitar los valores de los ejes.
plt.legend()
#plt.savefig(TITULO+"_DOS_"+str(ang)+".png")# Guardar grfico como imgen PNG.
plt.show() # Mostrarlo.
```

0.3. Segundo programa auxiliar: Rotación de láminas de grafeno con bordes armchair

0.2

```
# -*- coding: utf-8 -*-
"""
Created on Fri Jun 30 14:21:11 2023
```

```
@author: pablo
"""


import numpy as np
import math as ma
import numpy.linalg as la
from scipy.linalg import eig,eigh
from scipy.interpolate import make_interp_spline
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D #3D plots


#GRAFENO CON STACKING A-B

ang=4 # ngulo de rotaci n del plano superior respecto al inferior (en grados)

nAx=nAy=24
ax=np.array([1.42*1.5,1.42*np.sqrt(3)/2])
ay=np.array([1.42*1.5,-1.42*np.sqrt(3)/2])
az=np.array([1.42/2,1.42*np.sqrt(3)/2,3.35])
nAu=2
U=np.array([[-1.42/2,0],[1.42/2,0]])
e0=0.0 #Energa on-site
alpha=3.6
t=3.3*np.exp(alpha*1.42) #Integral de salto
V=t #Correcci n a la energa de enlace de VdW frente al enlace covalente
delta=0.2
DELTA=0.08



#Variables auxiliares
nAxy=nAx*nAy #N mero total de c.u. por capa x-y
nA=nAxy*nAu #N mero total de tomos
```

```
#Ordenaci n y posici n de los  tomos :
R0=np.zeros([nA,2])
for iy in range(nAy):
    for ix in range(nAx):
            for iu in range(nAu):
                R0[iu+(ix+iy*nAx)*nAu]=U[iu]+ix*ax+iy*ay

#Escala de los ejes en la representaci n gr fica:
#Calculamos el valor m ximo y m nimo que alcanza la posici n de los  tomos  en
#cada eje para definir el espacio de representaci n gr fica de la red
Rxmin=np.min(R0[:,0])
Rxmax=np.max(R0[:,0])
Rymin=np.min(R0[:,1])
Rymax=np.max(R0[:,1])

if Rxmin<Rymin:
    Rmin=Rxmin
else:
    Rmin=Rymin

if Rxmax<Rymax:
    Rmax=Rymax
else:
    Rmax=Rxmax




#GRAFENO ARMCHAIR PURO
TITULO="GRAFENO bordes armchair"

#Ecuaci n de la recta y=mx+n con m=(ax+2*U[0,:])[1]/(ax+2*U[0,:])[0]
#que pasa por el punto x=(nAx-1)*ax[0], y=(nAx-1)*ax[1]
m1=(ax+2*U[0,:])[1]/(ax+2*U[0,:])[0]
n1=(nAx-1)*ax[1]-m1*(nAx-1)*ax[0]
xa1=[Rxmin,Rxmax]
ya1=[m1*(Rxmin)+n1,m1*(Rxmax)+n1]
```

```
#Ecuaci n de la recta y=mx+n con m=(ax+2*U[0,:])[1]/(ax+2*U[0,:])[0]
#que pasa por el punto x=(nAx-1)*ay[0], y=(nAx-1)*ay[1]
m2=(ax+2*U[0,:])[1]/(ax+2*U[0,:])[0]
n2=(nAx-1)*ay[1]-m2*(nAx-1)*ay[0]
xa2=[Rxmin,Rxmax]
ya2=[m2*(Rxmin)+n2,m2*(Rxmax)+n2]

#Ecuaci n de la recta y=mx+n con m=(ay+2*U[0,:])[1]/(ay+2*U[0,:])[0]
#que pasa por el punto x=(nAx-1)*ax[0], y=(nAx-1)*ax[1]
m3=(ay+2*U[0,:])[1]/(ay+2*U[0,:])[0]
n3=(nAx-1)*ax[1]-m3*(nAx-1)*ax[0]
xa3=[Rxmin,Rxmax]
ya3=[m3*(Rxmin)+n3,m3*(Rxmax)+n3]

#Ecuaci n de la recta y=mx+n con m=(ay+2*U[0,:])[1]/(ay+2*U[0,:])[0]
#que pasa por el punto x=(nAx-1)*ay[0], y=(nAx-1)*ay[1]
m4=(ay+2*U[0,:])[1]/(ay+2*U[0,:])[0]
n4=(nAx-1)*ay[1]-m4*(nAx-1)*ay[0]
xa4=[Rxmin,Rxmax]
ya4=[m4*(Rxmin)+n4,m4*(Rxmax)+n4]

#xa4=[Rxmin,Rxmax/2]
#ya4=[(nAy-1.5)*ay[1],(nAy-1.5)*ay[1]]

#Vista a rea

plt.figure(figsize=(6,6))
plt.plot(R0[:,0],R0[:,1],'b.')
plt.plot(xa1, ya1,'r', linewidth=1)
plt.plot(xa2, ya2,'r', linewidth=1)
plt.plot(xa3, ya3,'r', linewidth=1)
plt.plot(xa4, ya4,'r', linewidth=1)
plt.suptitle(TITULO)
plt.xlim(Rmin-2,Rmax+2)
plt.ylim(Rmin-2,Rmax+2)
#plt.title("Vista a rea para "+str(ang)+r"$^o$")
plt.xlabel('X')
plt.ylabel('Y')
```

```
#plt.savefig(TITULO+"_aut_"+str(ang)+".png")
plt.show()


#Puntos por debajo de la recta y=m1x+n1
Ra1=np.array([0,0])
for i in range(nA):
    if R0[i,1]<m1*R0[i,0]+n1:
        Ra1=np.vstack([Ra1,R0[i,:]])
Ra1 = np.delete(Ra1, np.s_[0], axis=0)



#Puntos por encima de la recta y=m2x+n2
Ra2=np.array([0,0])
for i in range(np.shape(Ra1)[0]):
    if Ra1[i,1]>m2*Ra1[i,0]+n2:
        Ra2=np.vstack([Ra2,Ra1[i,:]])
Ra2 = np.delete(Ra2, np.s_[0], axis=0)


#Puntos por debajo de la recta y=m3x+n3
Ra3=np.array([0,0])
for i in range(np.shape(Ra2)[0]):
    if Ra2[i,1]<m3*Ra2[i,0]+n3:
        Ra3=np.vstack([Ra3,Ra2[i,:]])
Ra3 = np.delete(Ra3, np.s_[0], axis=0)


#Puntos por debajo de la recta y=m4x+n4
Ra4=np.array([0,0])
for i in range(np.shape(Ra3)[0]):
    if Ra3[i,1]>m4*Ra3[i,0]+n4:
        Ra4=np.vstack([Ra4,Ra3[i,:]])
Ra4 = np.delete(Ra4, np.s_[0], axis=0)

Rarm=Ra4
nAa=np.shape(Rarm)[0]
```

```
plt.figure(figsize=(6,6))
plt.plot(Rarm[:,0],Rarm[:,1],'b.')
plt.plot(xa1, ya1,'r', linewidth=1)
plt.plot(xa2, ya2,'r', linewidth=1)
plt.plot(xa3, ya3,'r', linewidth=1)
plt.plot(xa4, ya4,'r', linewidth=1)
plt.suptitle(TITULO)
plt.xlim(Rmin-2,Rmax+2)
plt.ylim(Rmin-2,Rmax+2)
#plt.title("Vista a rea para "+str(ang)+r"$^o$")
plt.xlabel('X')
plt.ylabel('Y')
#plt.savefig(TITULO+"_aut_"+str(ang)+".png")
plt.show()



#CONSTRUCCION DE UNA BICAPA

R=Rarm #Reciclamos la construcci n anterior
R=np.hstack([R,np.zeros((nAa,1))]) #A adimos una tercera dimensi n
R=np.concatenate((R,R)) #Generamos una copia de cada punto
for iF in range(nAa):
    R[iF+nAa,:]=R[iF+nAa,:]+az

nA=np.shape(R)[0]



# Hamiltoniano y matriz de solapes para l minas sin rotar
S00=np.zeros((nA,nA),complex)
H00=np.zeros((nA,nA),complex)

for iF in range(nA):
  S00[iF,iF]=1.0 #Consideramos que cada orbital s lo solapa consigo mismo

for iF in range(nA): #La interacci n de los electrones con cada tomo con
```

```
    #su n cleo viene prefijada por la energ a on-site, mientras que su
    #interacci n con otros n cleos decae exponencialmente con la distancia
    for jF in range(nA):
        if iF!=jF:
            H00[iF,jF]=-t*(np.exp(-alpha*np.sqrt(((R[iF,0]-R[jF,0])**2)+((R[iF,1]-R[jF
    H00[iF,iF]=e0


# Autovalores y autovectores para l minas sin rotar
E00,Evec00=eig(H00,S00)
ond=np.argsort(np.real(E00))
E00=np.real(E00[ond])




#ROTACION DE LA LAMINA SUPERIOR

#Centramos los planos x-y en el origen de coordenadas
#Las cuatro restas que acotan cada l mina pasan por
# x=(nAx-1)*ax[0], y=(nAx-1)*ax[1] y por x=(nAx-1)*ay[0], y=(nAx-1)*ay[1]




d=((nAx-1)/2)*(ax+ay)
d=np.array([d[0],d[1],0])
for ia in range (nA):
    R[ia]=R[ia]-d


#Rotaci n del plano superior respecto el inferior
theta=(ang*ma.pi)/180
A=np.array([[ma.cos(theta),-ma.sin(theta),0],[ma.sin(theta),ma.cos(theta),0],[0,0,1]])
for ir in range (nAa):
    R[nAa+ir]=np.dot(A,R[nAa+ir])
```

```
#Inversi n de la traslaci n anterior
for ia in range (nA):
    R[ia]=R[ia]+d


##Representaci n de las posiciones at micas
fig = plt.figure(figsize=(6,6)) #Espacio de representaci n en 3d
axe = fig.add_subplot(111, projection='3d') #Ejes
axe.scatter(R[:,0],R[:,1],R[:,2],s=5)
plt.suptitle(TITULO)
plt.xlim(Rmin+-2,Rmax+2)
plt.ylim(Rmin-2,Rmax+2)
#axe.set_zlim(Rmin,Rmax) #Descripci n gr fica m s realista pero confusa
axe.set_title("Posiciones at micas para "+str(ang)+r"$^o$")
axe.set_xlabel("X")
axe.set_ylabel("Y")
axe.set_zlabel("Z")
#plt.savefig(TITULO+"_red_"+str(ang)+".png")
plt.show()


# Hamiltoniano y matriz de solapes
Smat=np.zeros((nA,nA),complex)
Hmat=np.zeros((nA,nA),complex)

for iF in range(nA):
  Smat[iF,iF]=1.0 #Consideramos que cada orbital s lo solapa consigo mismo

for iF in range(nA): #La interacci n de los electrones con cada tomo con
    #su n cleo viene prefijada por la energ a on-site, mientras que su
    #interacci n con otros n cleos decae exponencialmente con la distancia
    for jF in range(nA):
        if iF!=jF:
            Hmat[iF,jF]=-t*(np.exp(-alpha*np.sqrt(((R[iF,0]-R[jF,0])**2)+((R[iF,1]-R[jF
    Hmat[iF,iF]=e0
```

```
# Autovalores y autovectores
Eval,Evec=eig(Hmat,Smat)


ind=np.argsort(np.real(Eval))
Eval=np.real(Eval[ind])


#Gr fica de autovalores
plt.figure(figsize=(6,6))
plt.plot(E00,'b.',markersize=1, label= "0   ")
plt.plot(Eval,'r.',markersize=1,label= str(ang)+r"$^o$")
plt.suptitle(TITULO)
plt.title("Autovalores para "+str(ang)+r"$^o$")
plt.xlabel('Autovalores')
plt.ylabel('Energ a')
plt.legend()
#   plt   .savefig(TITULO+"_aut_"+str(ang)+".png")
plt.show()


#DOS suave
def DOS(E): #funcion DOS
    dos=0
    for js in range(nA):
        dos=dos+(1/np.pi)*delta/(((E-Eval[js])**2)+delta**2)
    return dos

#DOS para el caso de 0
def DOS00(E): #funcion DOS
    dos=0
    for js in range(nA):
        dos=dos+(1/np.pi)*delta/(((E-E00[js])**2)+delta**2)
    return dos


#DOS con picos o DOSp
```

```
def DOSp(E): #funcion DOSp
    dosp=0
    for js in range(nA):
        dosp=dosp+(1/np.pi)*DELTA/(((E-Eval[js])**2)+DELTA**2)
    return dosp



#DOSp para el caso de 0
def DOSp00(E): #funcion DOS
    dosp00=0
    for js in range(nA):
        dosp00=dosp00+(1/np.pi)*DELTA/(((E-E00[js])**2)+DELTA**2)
    return dosp00



#Grafica DOS
ejeenergia=np.linspace(min(min(Eval),min(E00))-(2*delta), max(max(Eval),max(E00)+(2*de
plt.plot(ejeenergia,[DOS(i) for i in ejeenergia], 'r', linewidth=1, label= str(ang)+r"$
plt.plot(ejeenergia, [DOS00(i) for i in ejeenergia], 'b', linewidth=1, label= '0 ')
plt.suptitle(TITULO)
plt.title("Densidad de estados "+str(ang)+r"$^o$")
plt.xlabel('E')
plt.ylabel('DOS(E)')
plt.axhline(0, color="black")# Establecer el color de los ejes.
plt.axvline(0, color="black")
plt.xlim(min(Eval)-(2*delta), max(Eval)+(2*delta))# Limitar los valores de los ejes.
plt.legend()
plt.savefig(TITULO+"_DOS_"+str(ang)+".png")# Guardar grfico como imgen PNG.
plt.show() # Mostrarlo.



#Grafica DOSp
```

```
eje00=np.linspace(min(min(Eval),min(E00))-(2*DELTA), max(max(Eval),max(E00))+(2*DELTA)
plt.plot(eje00, [DOSp(i) for i in eje00], 'r', linewidth=1, label= str(ang)+r"$^o$") #
plt.plot(eje00, [DOSp00(i) for i in ejeenergia], 'b' , linewidth=1, label= '0   ' )
plt.suptitle(TITULO)
plt.title("Densidad de estados con picos "+str(ang)+r"$^o$")
plt.xlabel('E')
plt.ylabel('DOSp(E)')
plt.axhline(0, color="black")# Establecer el color de los ejes.
plt.axvline(0, color="black")
plt.xlim(min(min(Eval),min(E00))-(2*DELTA), max(max(Eval),max(E00))+(2*DELTA))# Limita
plt.legend()
plt.savefig(TITULO+"_DOSp_"+str(ang)+".png")# Guardar gr fico como im gen PNG.
plt.show() # Mostrarlo


#Diferencias de DOS entre el caso de 0   y el  ngulo  escogido
plt.plot(ejeenergia,[DOS(i)-DOS00(i) for i in ejeenergia], linewidth=1 )
plt.suptitle(TITULO)
plt.title("Diferencias de DOS entre el caso de 0   y "+str(ang)+r"$^o$")
plt.xlabel('E')
plt.ylabel('DOS(E)-DOS00(E)')
plt.axhline(0, color="black")# Establecer el color de los ejes.
plt.axvline(0, color="black")
plt.xlim(min(Eval)-(2*delta), max(Eval)+(2*delta))# Limitar los valores de los ejes.
plt.savefig(TITULO+"_DOS_"+str(ang)+".png")# Guardar gr fico como im gen PNG.
plt.show() # Mostrarlo.


#Diferencias de DOSp entre el caso de 0   y el  ngulo  escogido
plt.plot(eje00,[DOSp(i)-DOSp00(i) for i in eje00], linewidth=1 )
plt.suptitle(TITULO)
plt.title("Diferencias de DOSp entre el caso de 0   y "+str(ang)+r"$^o$")
plt.xlabel('E')
plt.ylabel('DOSp(E)-DOSp00(E)')
plt.axhline(0, color="black")# Establecer el color de los ejes.
plt.axvline(0, color="black")
plt.xlim(min(Eval)-(2*delta), max(Eval)+(2*delta))# Limitar los valores de los ejes.
plt.savefig(TITULO+"_DOS_"+str(ang)+".png")# Guardar gr fico como im gen PNG.
```

```
plt.show() # Mostrarlo.
```