# APPENDIX B

Calle J. Stenberg

Python Code

[47]:
```
import kwant
import numpy as np
import tinyarray
import scipy.linalg as la
import scipy.sparse.linalg as sla
import scipy.io
from matplotlib import pyplot
from cmath import exp
from math import pi
```

[48]:
```
def make_system(a=1, r1=10, r2=25):
    lat = kwant.lattice.square(a, norbs=1)
    sys = kwant.Builder()
    h = 1.054571817e-34
    e = 1.602e-19
    m = 0.026 * 9.1094e-31
    a = 1e-9
    g = 10
    u_b = 5.788e-5
    alpha = 16e-3
    #onsite term
    t = h**2 / ( 2 * m * a**2 ) / e
    #flux hopping phaseshift term
    gamma = h / (2 * m )
    #onsite flux term
    delta = e * a**2 / (2 * m)
    #soc hopping flux term
    sigma = alpha * e**2 * a**2 / h / e
    #soc onsite flux term
    psi = alpha * e * a**2 / h
    #zeeman splitting term
    E_z = g * u_b / 2
    V_barr = 131e-3

    sigma_0 = tinyarray.array([[1,0], [0,1]])
    sigma_x = tinyarray.array([[0, 1], [1, 0]])
    sigma_y = tinyarray.array([[0, -1j], [1j, 0]])
```

```
    sigma_z = tinyarray.array([[1,0], [0,-1]])

    def ring(pos):
        (x, y) = pos
        rsq = x ** 2 + y ** 2
        return (r1 ** 2 < rsq < r2 ** 2)

    def barr(site):
        (x, y) = site.pos
        if x == 0:
            return 2*V_barr
        else:
            return 0

    def potential(site, pot = 0.0087):
        (x, y) = site.pos
        if   x == 0:
            return 0
        if x < 0:
            return pot
        elif  x > 0:
            return -pot

    def onsite(site, B ):
        (x, y) = site.pos
        return ( 4 * t + delta * B**2 * x**2  + potential(site) + barr(site)) *␣
↪sigma_0 +  E_z * B * sigma_z  + psi * B * x * sigma_x

    def hopy(site1, site2, B ):
        (x, y) = site1.pos
        return -t * sigma_0 + 1j * alpha * sigma_x / 2 + 1j * gamma * B * x *␣
↪sigma_0

    #onsite elems
    sys[lat.shape(ring, (0, r1+1))] = onsite
    #hoppings in x-directions
    sys[kwant.builder.HoppingKind((1, 0), lat, lat)] = \
        -t * sigma_0 - 1j * alpha * sigma_y / 2
    # hoppings in y-directions
    sys[kwant.builder.HoppingKind((0, 1), lat, lat)] = \
        hopy
    return sys
```

[49]:
```
def make_system_detuning(a=1, r1=10, r2=25):
    lat = kwant.lattice.square(a, norbs=1)
    syst = kwant.Builder()
    h = 1.054571817e-34
```

```
e = 1.602e-19
m = 0.026 * 9.1094e-31
a = 1e-9
g = 10
u_b = 5.788e-5
alpha = 16e-3
#onsite term
t = h**2 / ( 2 * m * a**2 ) / e
#flux hopping phaseshift term
gamma = h / (2 * m )
#onsite flux term
delta = e * a**2 / (2 * m)
#soc hopping flux term
sigma = alpha * e**2 * a**2 / h / e
#soc onsite flux term
psi = alpha * e * a**2 / h
#zeeman splitting term
E_z = g * u_b / 2
B = 0.02
V_barr = 131e-3

sigma_0 = tinyarray.array([[1,0], [0,1]])
sigma_x = tinyarray.array([[0, 1], [1, 0]])
sigma_y = tinyarray.array([[0, -1j], [1j, 0]])
sigma_z = tinyarray.array([[1,0], [0,-1]])

def ring(pos):
    (x, y) = pos
    rsq = x ** 2 + y ** 2
    return (r1 ** 2 < rsq < r2 ** 2)

def barr(site):
    (x, y) = site.pos
    if x == 0:
        return 2*V_barr
    else:
        return 0

def potential(site, pot):
    (x, y) = site.pos
    if   x == 0:
        return 0
    if x < 0:
        return pot
    elif  x > 0:
        return -pot
```

```
    def onsite(site, pot ):
        (x, y) = site.pos
        return ( 4 * t +  delta * B**2 * x**2  + potential(site, pot) +␣
↪barr(site)) * sigma_0 +  E_z * B * sigma_z  +  psi * B * x * sigma_x

    def hopy(site1, site2 ):
        (x, y) = site1.pos
        return -t * sigma_0 +  1j * alpha * sigma_x / 2 + 1j * gamma * B * x *␣
↪sigma_0

    #onsite elems
    syst[lat.shape(ring, (0, r1+1))] = onsite
    #hoppings in x-directions
    syst[kwant.builder.HoppingKind((1, 0), lat, lat)] = \
        -t * sigma_0 - 1j * alpha * sigma_y / 2
    # hoppings in y-directions
    syst[kwant.builder.HoppingKind((0, 1), lat, lat)] = \
        hopy
    return syst
```

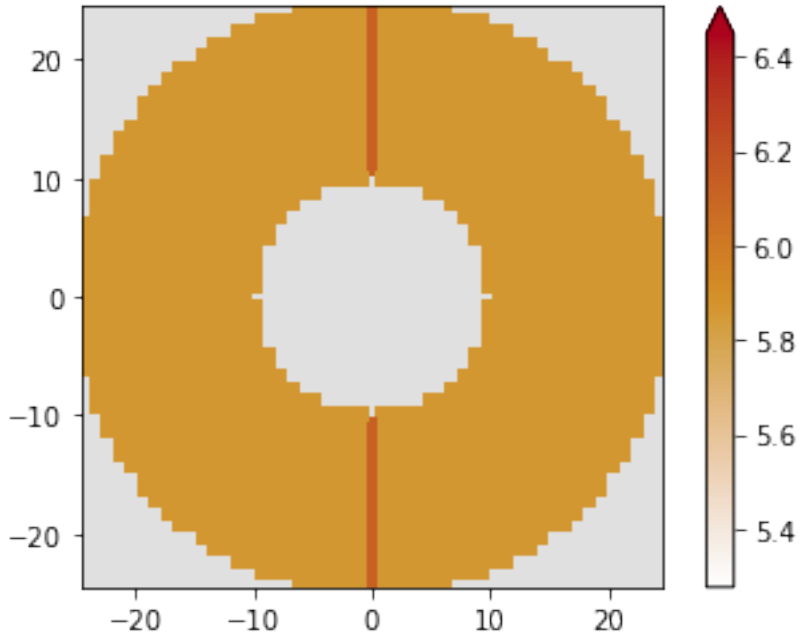
[50]:
```
def make_system_spinless(a=1, r1=10, r2=25):
    lat = kwant.lattice.square(a, norbs=1)
    sys = kwant.Builder()
    h = 1.054571817e-34
    e = 1.602e-19
    m = 0.026 * 9.1094e-31
    a = 1e-9
    g = 10
    u_b = 5.788e-5
    alpha = 16e-3
    #onsite term
    t = h**2 / ( 2 * m * a**2 ) / e
    #flux hopping phaseshift term
    gamma = h / (2 * m )
    #onsite flux term
    delta = e * a**2 / (2 * m)
    #soc hopping flux term
    sigma = alpha * e**2 * a**2 / h / e
    #soc onsite flux term
    psi = alpha * e * a**2 / h
    #zeeman splitting term
    E_z = g * u_b / 2
    V_barr = 131e-3

    sigma_0 = tinyarray.array([[1,0], [0,1]])
    sigma_x = tinyarray.array([[0, 1], [1, 0]])
    sigma_y = tinyarray.array([[0, -1j], [1j, 0]])
```

```
    sigma_z = tinyarray.array([[1,0], [0,-1]])

    def ring(pos):
        (x, y) = pos
        rsq = x ** 2 + y ** 2
        return (r1 ** 2 < rsq < r2 ** 2)

    def circle(pos):
        x, y = pos
        return x**2 + y**2 < r2**2
    def barr(site):
        (x, y) = site.pos
        if x == 0:
            return 2*V_barr
        else:
            return 0

    def potential(site, pot = 0):
        (x, y) = site.pos
        if   x == 0:
            return 0
        if x < 0:
            return pot
        elif  x > 0:
            return -pot

    def onsite(site ):
        (x, y) = site.pos
        return ( 4 * t + potential(site) + barr(site) )

    def hopy(site1, site2 ):
        (x, y) = site1.pos
        return -t

    #onsite elems
    sys[lat.shape(ring, (0, r1+1))] = onsite
    #hoppings in x-directions
    sys[kwant.builder.HoppingKind((1, 0), lat, lat)] = \
        -t
    # hoppings in y-directions
    sys[kwant.builder.HoppingKind((0, 1), lat, lat)] = \
        hopy
    return sys
```

[51]:
```
sys = make_system_spinless()
sys = sys.finalized()
```

```
def energy_onsite(i):
    return sys.hamiltonian(i, i)

kwant.plotter.map(sys, energy_onsite)
```

```
<ipython-input-51-6417ee82ec22>:7: RuntimeWarning: The plotted data contains
1.67% of values overflowing upper limit 5.86215
  kwant.plotter.map(sys, energy_onsite)
```



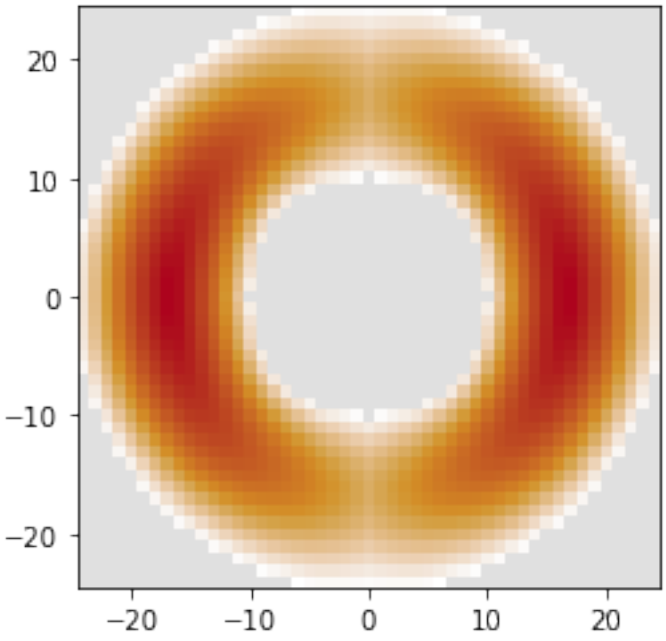

[52]:
```
def sorted_eigs(ev):
    evals, evecs = ev
    evals, evecs = map(np.array, zip(*sorted(zip(evals, evecs.transpose()))))
    return evals, evecs.transpose()
```

[53]:
```
def plot_detuning(syst, Potentials):
    energies = []
    for p in Potentials:
        # Obtain the Hamiltonian as a sparse matrix
        ham_mat = syst.hamiltonian_submatrix(params=dict(pot=p), sparse=True)
        ev = sla.eigsh(ham_mat.tocsc(), k=10, sigma=0,
                       return_eigenvectors=False)
        energies.append(ev)
    print(len(energies))
    print(len(Potentials))
    pyplot.figure()
    pyplot.plot(Potentials, energies)
```

```
    pyplot.xlabel("Delta [V]")
    pyplot.ylabel("energy [eV]")
    pyplot.show()
    return energies, Potentials
```

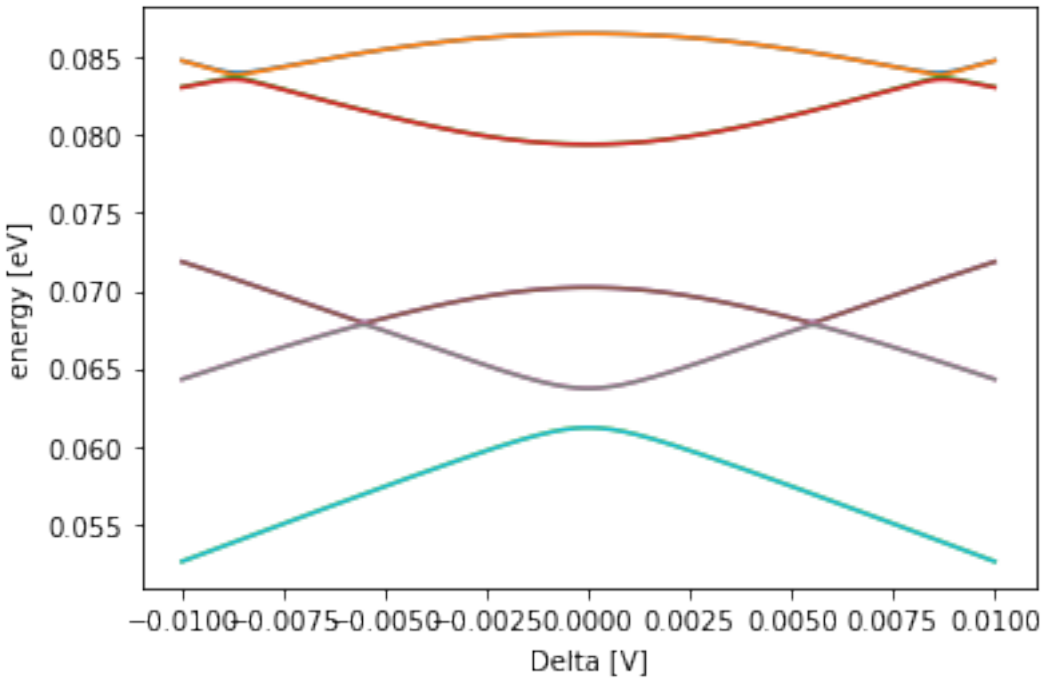
[54]:
```
def plot_spectrum(sys, Bfields):
    energies = []
    for B in Bfields:
        # Obtain the Hamiltonian as a sparse matrix
        ham_mat = sys.hamiltonian_submatrix(params=dict(B=B), sparse=True)
        ev = sla.eigsh(ham_mat.tocsc(), k=10, sigma=0,
                       return_eigenvectors=False)
        energies.append(ev)

    pyplot.figure()
    pyplot.plot(Bfields, energies)
    pyplot.xlabel("B field [T]")
    pyplot.ylabel("energy [eV]")
    pyplot.show()
    return energies, Bfields
```

[55]:
```
sys_barr = make_system_spinless()
sys_barr = sys_barr.finalized()

def plot_wave_function(syst, n):
    ham_mat = syst.hamiltonian_submatrix(sparse=True)
    evals, evecs = sorted_eigs(sla.eigsh(ham_mat.tocsc(), k=10, sigma=0))
    kwant.plotter.map(syst, abs(evecs[:, n]) ,
                  colorbar=False, oversampling=1)

plot_wave_function(sys_barr, 0)
```




[56]:
```
syst = make_system_detuning()
syst = syst.finalized()
ev , pots = plot_detuning(syst, [iP * 0.0001 for iP in range(-100,101)])
```

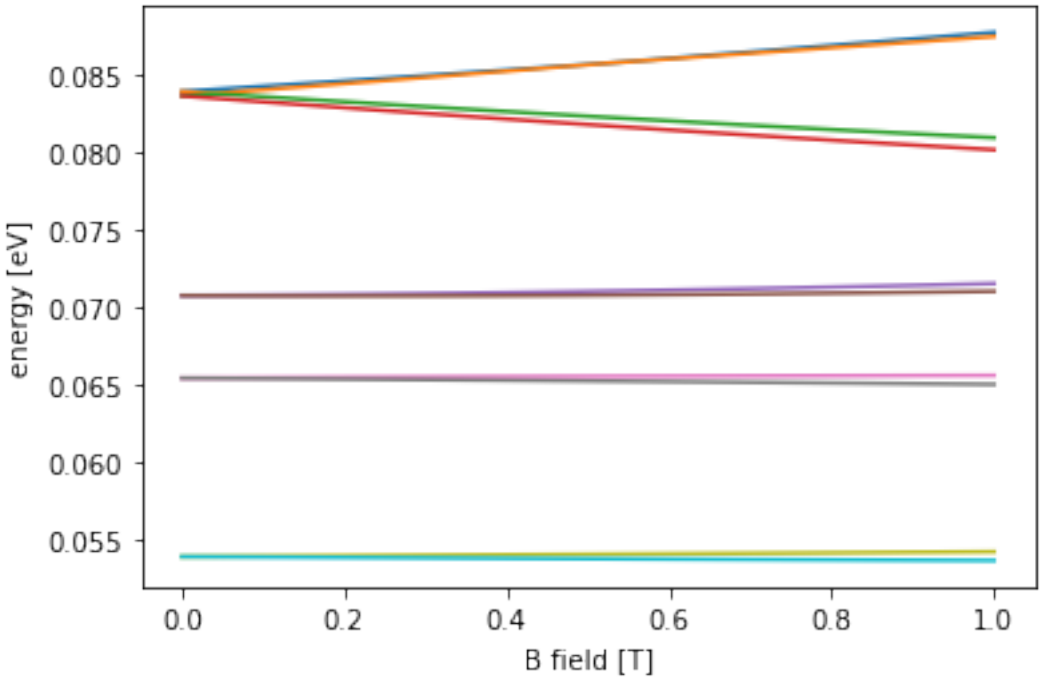
```
201
201
```



[57]:
```
#prints the Delta for closest distance between top two states
close = []
for j in range(len(pots)):
    tmp = abs(ev[j][0] - ev[j][3])
    if not close:
        close.append(tmp)
        close.append(j)
    else:
        if(  tmp < close[0] ):
            close[0] = tmp
            close[1] = j

print(abs(pots[close[1]]))
```

```
0.008700000000000001
```

[58]:
```
sys = make_system()
sys = sys.finalized()
en, b_n = plot_spectrum(sys, [iB * 0.01 for iB in range(101)] )
```




[59]:
```
def L_calc():
    h = 1.054571817e-34
```

```
    e = 1.602e-19
    a = 1e-9
    omega = 1/(2)
    r1 = 10
    r2 = 25

    sigma_0 = tinyarray.array([[1,0], [0,1]])

    sub = kwant.lattice.square(1, norbs=1)
    l_sys = kwant.Builder()
    def ring(pos):
            (x, y) = pos
            rsq = x ** 2 + y ** 2
            return (r1 ** 2 < rsq < r2 ** 2)
    def circle(pos):
            x, y = pos
            return x**2 + y**2 < r2**2
    def hopy(site1, site2):
            (x, y) = site1.pos
            return -1j * omega * x * sigma_0
    def hopx(site1, site2 ):
            (x, y) = site1.pos
            return 1j * omega * y * sigma_0

    l_sys[sub.shape(ring, (0, r1+1))] = 0 * sigma_0
    l_sys[kwant.builder.HoppingKind((1, 0), sub, sub)] = hopx
    l_sys[kwant.builder.HoppingKind((0, 1), sub, sub)] = hopy

    l_sys = l_sys.finalized()
    L = l_sys.hamiltonian_submatrix()

    L_dagger = L.conj()
    L_dagger = L_dagger.transpose()
    #Checking hermicity
    print(np.allclose(L,L_dagger))

    Lvals, Lvecs = la.eigh(L)

    return Lvals, Lvecs
```

[60]:
```
Lvals, Lvecs = L_calc()
```

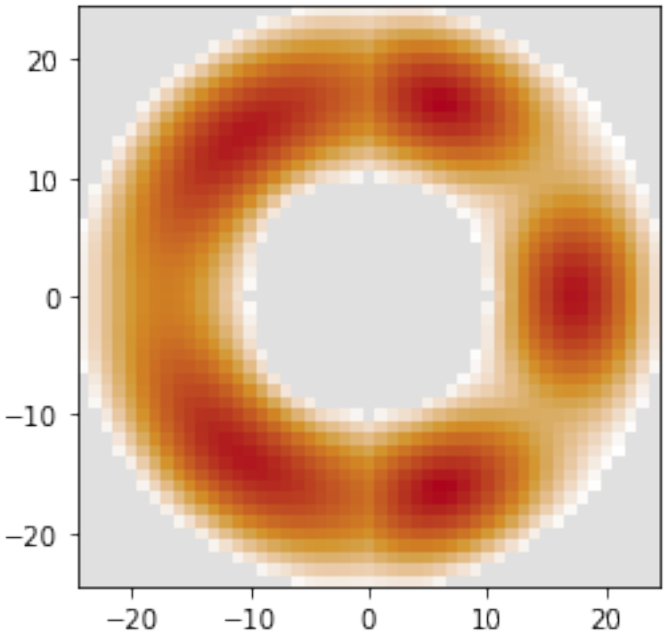
```
True
```

[61]:
```
#calculating the angular momentum of the 6th eigenstate at the (2,3) crossing␣
 ↪for a small B-field
sys_l = make_system()
```

```
sys_l = sys_l.finalized()
b = 0.02 #tesla
h_mat = sys_l.hamiltonian_submatrix(params=dict(B=b))
lva, evecs = la.eigh(h_mat)

eig = evecs[:,6]
L_sum = 0
for j in range(0, len(Lvecs)):
    L_sum += Lvals[j] * abs( np.dot(eig.conj(),Lvecs[:,j]))**2
print(L_sum)
```

```
1.631882758898755
```

[63]:
```
def plot_wave_function(syst, n, B=0.02):
    # Calculate the wave functions in the system.
    ham_mat = sys_l.hamiltonian_submatrix(sparse=True, params=dict(B=B))
    evals, evecs = sorted_eigs(sla.eigsh(ham_mat.tocsc(), k=30, sigma=0))
    kwant.plotter.map(sys_l, abs(evecs[1::2, n])+abs(evecs[::2,n]),
                      colorbar=False, oversampling=1)

plot_wave_function(sys_l, 6, B=0.02)
```




[ ]:
```

```