Einführung in die EPR Spektroskopie

Übungsblatt 3

(12.11.2021)

Besprechung 09.11.2021

Please use the links given below to access the instructions and iPython notebooks for the questions. For your convenience, the text of the questions provided below. Note, however, that some of the instructions are related specifically to the code and to the output it generates, and will only make sense if you work on the iPython notebooks at the given links.

1 Lorentzian lineshape

`https://colab.research.google.com/drive/1qhxGlX70CTpGfBUjzF7Sp71xd-8PXLlK?usp=sharing`

The absorption lineshape in magnetic resonance can often be described by the following Lorentzian function:

$$L(x; c, \gamma) = \frac{1}{\pi\gamma}\left[\frac{\gamma^2}{\gamma^2 + (x-c)^2}\right], \tag{1}$$

where the parameters c and γ determine the position of the center and the width of the spectral line, respectively.

In this exercise, you will get familiar with some of the properties of the Lorentzian function.

a) Plotting the spectral line

Let us start by plotting the function for different values of its two parameters c and γ.

To be able to plot using Python, I will need to **import** the package `matplotlib.pyplot`. It will also be useful to have the package `numpy`, which is very convenient for numerical calculations.

```
import numpy as np
import matplotlib.pyplot as plt
```

I will now define a Python function for calculating the value of the Lorenzian at a given point x, and for given parameters c (`center`) and γ (`width`).

```
def Lorentz(x,center,width):
  return 1/(np.pi * width) * (width**2 /(width**2 + (x - center)**2))
```

Now I prepare a numpy array that contains the points on the x axis for which I will calculate the Lorentzian function.

```
x_points = np.linspace(-5,5,100)
print(x_points)
```

```
[-5.         -4.8989899  -4.7979798  -4.6969697  -4.5959596  -4.49494949
 -4.39393939 -4.29292929 -4.19191919 -4.09090909 -3.98989899 -3.88888889
 -3.78787879 -3.68686869 -3.58585859 -3.48484848 -3.38383838 -3.28282828
 -3.18181818 -3.08080808 -2.97979798 -2.87878788 -2.77777778 -2.67676768
 -2.57575758 -2.47474747 -2.37373737 -2.27272727 -2.17171717 -2.07070707
 -1.96969697 -1.86868687 -1.76767677 -1.66666667 -1.56565657 -1.46464646
 -1.36363636 -1.26262626 -1.16161616 -1.06060606 -0.95959596 -0.85858586
 -0.75757576 -0.65656566 -0.55555556 -0.45454545 -0.35353535 -0.25252525
 -0.15151515 -0.05050505  0.05050505  0.15151515  0.25252525  0.35353535
  0.45454545  0.55555556  0.65656566  0.75757576  0.85858586  0.95959596
  1.06060606  1.16161616  1.26262626  1.36363636  1.46464646  1.56565657
  1.66666667  1.76767677  1.86868687  1.96969697  2.07070707  2.17171717
  2.27272727  2.37373737  2.47474747  2.57575758  2.67676768  2.77777778
  2.87878788  2.97979798  3.08080808  3.18181818  3.28282828  3.38383838
  3.48484848  3.58585859  3.68686869  3.78787879  3.88888889  3.98989899
  4.09090909  4.19191919  4.29292929  4.39393939  4.49494949  4.5959596
  4.6969697   4.7979798   4.8989899   5.        ]
```

This is an array of 100 numbers starting from -5 and going to 5.

(I print this array so that you can see the numbers that it contains. Feel free to change the starting and ending values, as well as the number of points, and observe how the numbers in the array change.)

Now I will call the Lorentzian (Python) function that I defined in the beginning to calculate the values of the Lorentzian (mathematical) function at all 100 points contained in the array `x_points`.

Note how I also specify the center and the width of the Lorentzian as the second and third arguments that I provide to the function call.

```
c = -1
gamma = 1
y_points = Lorentz(x_points, c, gamma)
```

```
print(y_points)
```

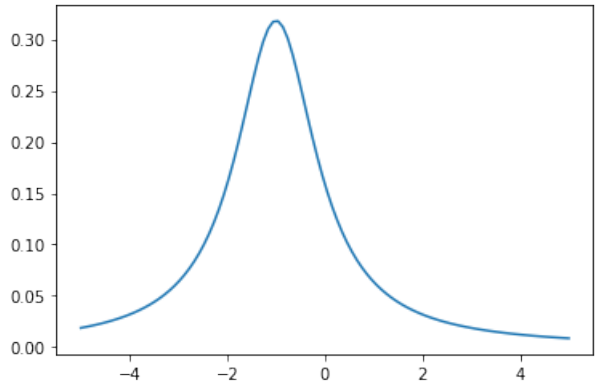
```
[0.01872411 0.01964618 0.02063644 0.02170159 0.02284916 0.02408761
 0.0254265  0.0268766  0.02845012 0.03016092 0.03202475 0.03405958
 0.03628593 0.0387273  0.04141066 0.04436701 0.04763203 0.05124686
 0.05525896 0.05972309 0.06470239 0.0702695  0.07650772 0.08351193
 0.09138926 0.10025887 0.11025039 0.12149999 0.13414263 0.14829848
 0.16405086 0.18141276 0.20027959 0.22036838 0.24114982 0.26179032
 0.28113501 0.2977718  0.31020734 0.31714498 0.3177911  0.31206914
 0.30064134 0.28472713 0.26580516 0.24532163 0.22449127 0.20421255
 0.18507179 0.16739578 0.15131955 0.13684937 0.1239129  0.11239526
 0.10216312 0.09307979 0.0850139  0.07784403 0.07146059 0.06576628
 0.06067556 0.05611373 0.05201586 0.04832559 0.04499409 0.04197902
 0.03924368 0.03675619 0.03448882 0.03241742 0.0305209  0.02878083
 0.02718101 0.02570725 0.02434703 0.02308929 0.02192425 0.02084325
 0.01983858 0.01890337 0.0180315  0.01721748 0.0164564  0.01574386
 0.01507587 0.01444886 0.0138596  0.01330517 0.0127829  0.01229039
 0.01182545 0.01138609 0.01097049 0.01057698 0.01020405 0.00985029
 0.00951444 0.0091953  0.00889181 0.00860297]
```

The results of the calculation are stored in the array `y_points`.

(Again I print this array to see that it contains 100 numbers.)

Now we will make a plot using the x coordinates contained in the array `x_points` and the y coordinates contained in the array `y_points`.

```
plt.plot(x_points,y_points);
```

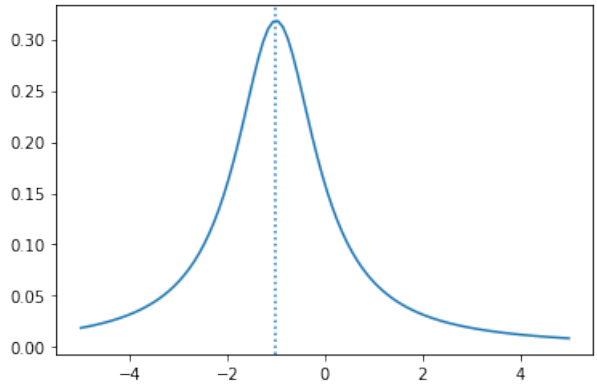


Observe where the center (i.e., peak) of the line is located. Go back and change the center or the

width of the Lorentzian and plot the result again.

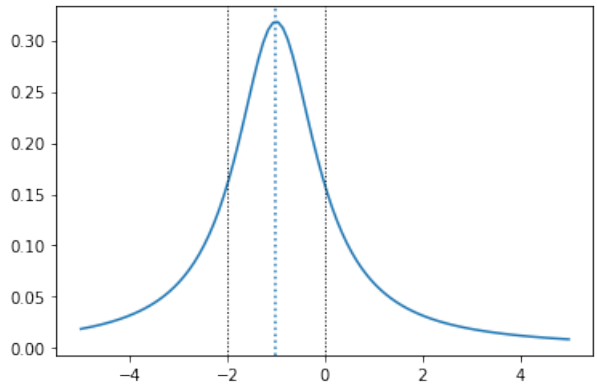
Let us now show the position of the center on the plot by also plotting a vertical line at $x = c$.

```
plt.axvline(x=c,linestyle=":") #plots a vertical line at x = c
plt.plot(x_points,y_points);
```



It will also be nice to show how far from the center the points $c - \gamma$ and $c + \gamma$ are.

```
plt.plot(x_points,y_points)
plt.axvline(x=c,linestyle=":");
plt.axvline(x=c-gamma,linestyle=":",color="k", linewidth=1) #a vertical line at x = c-gamma
plt.axvline(x=c+gamma,linestyle=":",color="k", linewidth=1); #a vertical line at x = c+gamm
```



Go back and change the center and with of the Lorentzian, and observe how the positions of the vertical lines in the plot change after that.

Observe that 2γ is the width of the spectral line (our Lorentzian function) at an elevation that is half of the maximum elevation. Thus we say that 2γ is the *full width at half-maximum*. Alternatively, we can say that γ is the *half-width at half-maximum*.

Some people prefer to write the Lorentzian function in terms of the full width at half-maximum,

$\Gamma = 2\gamma$. In terms of this parameter the Lorentzian function becomes

$$L(x; c, \Gamma) = \frac{1}{\pi}\left[\frac{\frac{\Gamma}{2}}{\left(\frac{\Gamma}{2}\right)^2 + (x-c)^2}\right]. \tag{2}$$

b) Derivative of the Lorentzian function

In the lecture you learned that an EPR spectrometer working in a continuous mode (as opposed to pulsed mode) produces the derivative of the spectral line.

The derivative of the Lorentzian function $L(x; c, \gamma)$ with respect to x is

$$L_D(x; c, \gamma) = \frac{\partial L(x; c, \gamma)}{\partial x} = -\frac{\gamma}{\pi}\frac{2(x-c)}{[\gamma^2 + (x-c)^2]^2}. \tag{3}$$

To plot this derivative function, I create a Python function that calculates the derivative for given values of x and the two parameters.

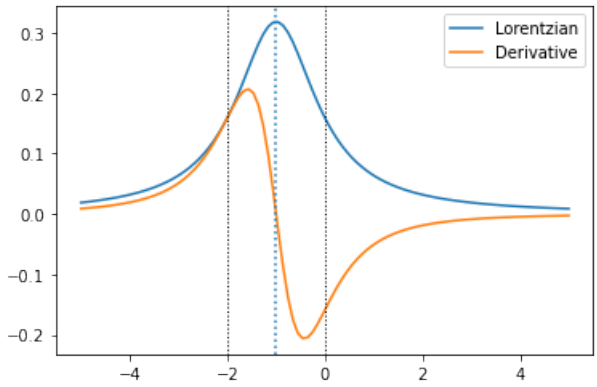
```
def LorentzD(x,center,width):
  return -(width/np.pi) * 2*(x-center) /( (width**2 + (x - center)**2)**2 )
```

Now I use this (Python) function to calculate the derivatives of the Lorentzian at the points contained in the array `x_points`.

```
yD_points = LorentzD(x_points, c, gamma)
```

Finally, I plot this derivative on top of the previously calculated function.

```
plt.plot(x_points,y_points,label="Lorentzian") #the Loretzian that was calculated before
plt.plot(x_points,yD_points,label="Derivative") #the derivative of the Lorentzian
plt.legend() #the legend will show the labels of the lines on the plot
plt.axvline(x=c,linestyle=":") #vertical line at x = c
plt.axvline(x=c-gamma,linestyle=":",color="k", linewidth=1) #vertical line at x = c-gamma
plt.axvline(x=c+gamma,linestyle=":",color="k", linewidth=1); #vertical line at x = c+gamma
```



When there are several lines on the same plot, it is useful to label them so that we know which line is which.

Examine the above code to see how I provided the labels of the two lines.

For the labels to appear on the plot it is necessary to activate the "legend" with `plt.legend()` the way I have done above.

c) Peak-to-peak width of the derivative of the Lorentzian

The last plot shows that the peak-to-peak width of the derivative lineshape (orange) is different than the width parameter γ of the Lorentzian.

As this peak-to-peak width is easy to extract from the EPR spectrum, it is useful to know how it relates to the parameter γ.

Obtain an analytical expression for the peak-to-peak width Δ_{pp} in terms of γ.

Check whether your expression is correct by plotting two vertical lines at $x = c - \Delta_{\mathrm{pp}}/2$ and $x = c + \Delta_{\mathrm{pp}}/2$. These lines should go through the peaks of the orange line.

2 Direct product of two matrices

`https://colab.research.google.com/drive/1di80UexNwIylgSHcp65TsiC40i9G74-e?usp=sharing`

In question 2 of the previous problem set we defined the Pauli spin matrices

$$S_x = \frac{1}{2}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad S_y = \frac{1}{2}\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad S_z = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{4}$$

as **SymPy** matrices in the following way:

```
import sympy as sp
#define the three spin matrices of the electron as SymPy matices
Sx = sp.Matrix([[0,1],[1,0]]) / 2
Sy = sp.Matrix([[0,-sp.I],[sp.I,0]]) / 2  #sp.I is the imaginary unit i
Sz = sp.Matrix([[1,0],[0,-1]]) / 2
```

In the lecture you learned that the *hyperfine interaction* between an electron spin and a nuclear spin can be described by the Hamiltonian operator

$$\hat{H}_{\mathrm{HF}} = a_{\mathrm{iso}}\, \hat{S}_z \otimes \hat{I}_z, \tag{5}$$

where the operators $\hat{S}_z$ and $\hat{I}_z$ refer to, respectively, the electron spin and the nuclear spin.

For a nuclear spin with $I = 1/2$, the matrix representation of $\hat{I}_z$ is identical to that of $\hat{S}_z$ of the electron spin. In fact, the matrix representations of $\hat{I}_x$ and $\hat{I}_y$ are also the same as those of $\hat{S}_x$ and $\hat{S}_y$. We can thus define the matrices of the nuclear spin operators as **SymPy** matrices in the same way as we did above for the electron.

```
#define the three spin matrices of the NUCLEUS as SymPy matices
Ix = sp.Matrix([[0,1],[1,0]]) / 2
Iy = sp.Matrix([[0,-sp.I],[sp.I,0]]) / 2  #sp.I is the imaginary unit i
Iz = sp.Matrix([[1,0],[0,-1]]) / 2
```

We will now use **SymPy** to obtain the matrix representation of the direct product

$$\hat{S}_z \otimes \hat{I}_z \tag{6}$$

which appears in the hyperfine-coupling Hamiltonian. The operation of direct product is called `TensorProduct` in sympy. We import this function from the quantum package of sympy as follows:

```
from sympy.physics.quantum import TensorProduct
```

Now we can use the `TensorProduct` function to calculate the desired direct product.

```
Sz_Iz = TensorProduct(Sz,Iz)
```

The strength of coding is that you can calculate any other direct products between the operators of the electron spin and the nuclear spin. Use the function `TensorProduct` to calculate the following direct products:

$$\hat{S}_z \otimes \hat{I}_x, \quad \hat{S}_x \otimes \hat{I}_y. \tag{7}$$

Examine the results and make sure that you can obtain them on your own with pen and paper. (You will not have access to SymPy in the oral exam.)

Now use `TensorProduct` to calculate these more involved direct products which appear very often in magnetic resonance:

$$\hat{S}_z \otimes \hat{I}_+ = \hat{S}_z \otimes (\hat{I}_x + i\hat{I}_y), \quad \hat{S}_- \otimes \hat{I}_+ = (\hat{S}_x - i\hat{S}_y) \otimes \hat{I}_+. \tag{8}$$

3 Detection sensitivity

`https://colab.research.google.com/drive/1RVsi3x47Y8ndJie_UmwAkmiBt6ofQJRq?usp=sharing`

The signal strength (in Volts) for a cw-EPR spectrometer is given by the relationship

$$V_S = \chi'' \, \eta \, Q \, \sqrt{P_{\mathrm{MW}} \, Z_0}. \tag{9}$$

In this expression,

η is the filling factor of the resonator, and has a typical value of 0.01;

Q is the quality factor of the resonator, and has a typical value of 1000;

P_{MW} is the microwave power, with a typical value of 100 mW;

Z_0 is the wave resistance of the resonator, with typical value of 50 Ohm;

χ'' is the imaginary component of the magnetic susceptibility of the sample.

The imaginary component of the magnetic susceptibility can be expressed in terms of other properties as follows:

$$\chi'' = \frac{N \gamma_{\mathrm{e}}^2 \hbar^2 S(S+1) \mu_0}{3 k_{\mathrm{B}} T} \frac{\nu_{\mathrm{MW}} T_2}{2}, \tag{10}$$

where N is the number of unpaired electrons in one cubic meter, ν_{MW} is the microwave frequency at which the spectrometer operates, and T_2 is a relaxation time scale that is specific to the free radical in the sample.

For a free radical with $T_2 = 10\,\mu$s, determine the minimum concentration that can be detected at $T = 300$ K on a spectrometer with $\nu_{\mathrm{MW}} = 9.8$ GHz, if the noise at the detector has a magnitude of 1 μV.