calibration

December 19, 2022

[1]:
```python
# Notebook for the custom calibration
```

[2]:
```python
import numpy as np
import plotly.graph_objects as go
from scipy.signal import find_peaks
from scipy.optimize import curve_fit

from utils.get_emsa_data import get_counts_and_name
```

[3]:
```python
# helper functions important for the calibration

# converts channel to keV
def channel_to_kev(
    value=None, arr=None, dispersion=0.0100283, offset=21.078, use_offset=True
):
    if not use_offset:
        offset = 0
    if value is not None:
        return (value - offset) * dispersion
    if arr is not None:
        return (np.array(arr) - offset) * dispersion
    else:
        raise ValueError("No value or array provided to chennel_to_keV(...)")


# this function makes one gaussian
def gaussian(xaxis, mu, amp, std):
    return amp * np.exp(-((xaxis - mu) ** 2) / (2 * std**2))


# this function makes n gaussians and a m=deg order polynomial on xaxis
def n_gaussians_and_m_order_bg(xaxis, deg, *args):
    poly = args[: deg + 1]
    gauss = args[deg + 1 :]
    n = int(len(gauss) / 3)
    y = np.zeros(len(xaxis))
    for i in range(n):
        y += gaussian(xaxis, gauss[3 * i], gauss[3 * i + 1], gauss[3 * i + 2])
```

```
    y += np.polyval(poly, xaxis)
    return y


# this function removes the background linearly, used to make an initial bg␣
↪guess
def remove_linear_bg(y, peaks, pixel_removal=40, offset=10):
    bg = y.copy()
    for p in peaks:
        # if the peak is too close to the edge, just say the bg is 0
        if p < pixel_removal:
            bg[:p] = 0
            continue
        else:
            bg[p - pixel_removal : p + pixel_removal] = np.nan
    # make linear interpolation between nan values
    bg = np.interp(np.arange(len(bg)), np.flatnonzero(~np.isnan(bg)), bg[~np.
↪isnan(bg)])
    bg[: int(offset)] = 0  # first 10 channels or the offset should be 0
    return bg


# This function makes the mu, amp, std and polynomial coefficients
def make_fit(array, x, deg=12, prominence=0.01, pixel_removal=50, offset=20):
    # fits one array to n gaussians and a m order polynomial
    peaks, _ = find_peaks(array, prominence=prominence)
    std = np.ones(len(peaks))
    amp = np.ones(len(peaks))

    # must define the function to cure_fit, since deg is a variable which is␣
↪not allowed in the function,
    # because it is not a parameter of the function that curve_fit is trying to␣
↪fit
    def fit_func_peaks_and_bg(x, *args):
        poly = args[: deg + 1]
        gauss = args[deg + 1 :]
        n = int(len(gauss) / 3)
        y = np.zeros(len(x))
        for i in range(n):
            y += gaussian(x, gauss[3 * i], gauss[3 * i + 1], gauss[3 * i + 2])
        y += np.polyval(poly, x)
        return y

    # removing the background linearly for an initial guess of the polynomial
    bg = remove_linear_bg(array, peaks, pixel_removal=pixel_removal,␣
↪offset=offset)
    # fitting the polynomial
```

```python
    poly_init = np.polyfit(x, bg, deg)

    # fitting
    init_vals = list(poly_init)
    for i in range(len(peaks)):
        init_vals += [peaks[i], amp[i], std[i]]
    fit_vals, covar = curve_fit(fit_func_peaks_and_bg, x, array, p0=init_vals)

    return fit_vals, covar


def calculate_calibration(peaks_channel, peaks_keV):
    """
    Calibration of the channel width and offset using two peaks.
    The given channels and keV must correspond.

    Parameters
    ----------
    peaks_channel : list
        channel value of peaks, fitted
    peaks_keV : list
        theoretical energy of the two peaks

    Returns
    -------
    float, float
        dispersion, offset
    """

    # figure out the distances between the peaks
    channel_distance = peaks_channel[1] - peaks_channel[0]
    kev_distance = peaks_keV[1] - peaks_keV[0]

    # dispersion = (p1_keV - p0_keV) / (p1_channel - p0_channel)
    dispersion = kev_distance / channel_distance  # kev_per_channel
    offset = peaks_channel[0] - peaks_keV[0] / dispersion  #␣
 ↪calibrated_zero_channel

    print(
        f"Calibration: {dispersion:.08f} keV/channel, {offset:.03f} channels␣
 ↪zero offset"
    )

    return dispersion, offset
```

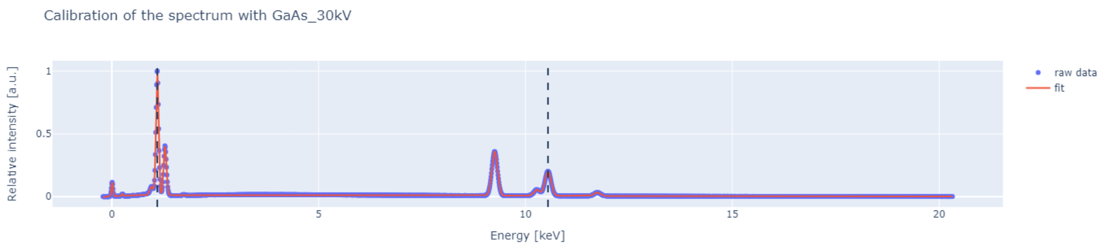
[4]:
```python
# variables used
channels = np.arange(2048)
```

```
ga_La = 1.098
as_Ka = 10.5436
peaks_keV = [ga_La, as_Ka]
initial_dispersion = 0.01
initial_offset = 20
deg = 12
prominence = 0.01

file = "GaAs_30kV"

ga30, _ = get_counts_and_name(file)
spectrum = ga30
```

[5]:
```
# find peaks
peaks_channel_est, _ = find_peaks(spectrum, prominence=0.01)
peaks_keV_est = channel_to_kev(peaks_channel_est,␣
 ↪dispersion=initial_dispersion, offset=initial_offset)

# find the peaks that are closest to the peaks_keV
peaks_to_fit_index = []
for peak in peaks_keV:
    peaks_to_fit_index.append(np.argmin(np.abs(peaks_keV_est - peak)))

fit_vals, _ = make_fit(spectrum, channels, deg=deg, prominence=prominence)

fit_peaks = fit_vals[deg + 1::3]
dispersion, offset = calculate_calibration(fit_peaks[peaks_to_fit_index],␣
 ↪peaks_keV)
fit_peaks_keV = channel_to_kev(fit_peaks, dispersion=dispersion, offset=offset)
keV = channel_to_kev(channels, dispersion=dispersion, offset=offset)
```

```
Calibration: 0.01003001 keV/channel, 21.127 channels zero offset
```

[6]:
```
fig = go.Figure()
fit = n_gaussians_and_m_order_bg(channels, deg, *fit_vals)
fig.add_trace(go.Scatter(x=keV, y=spectrum, name='raw data', mode='markers'))
fig.add_trace(go.Scatter(x=keV, y=fit, name='fit'))
fig.add_vline(x=fit_peaks_keV[peaks_to_fit_index[0]], line_width=2,␣
 ↪line_dash="dash")
fig.add_vline(x=fit_peaks_keV[peaks_to_fit_index[1]], line_width=2,␣
 ↪line_dash="dash")
fig.update_layout(title=f'Calibration of the spectrum with {file}')
fig.update_layout(xaxis_title='Energy [keV]', yaxis_title='Relative intensity␣
 ↪[a.u.]')
fig.show()
```


Calibration of the spectrum with GaAs_30kV
Relative intensity [a.u.]
Energy [keV]
raw data
fit
0
0.5
1
0
5
10
15
20