Code and summary

Jensen Lee, Gabe Valletto, Liam Ziegenhorn

December 4, 2025

1 Code for Data Processing

Below is the code used for processing the coalated data in the given H5 file. Roughly, the code pulls data for each date and cell and writes the data to files. It generates one file for each date, cell, and pixel if data is avaliable. EQE data is then calculated from each of the 8 pixels to create a file for the EQE data for each cell and date. From there, EQE data for each cell is grouped into three groups by C60 usage. For each C60 usage group is then averaged for each date, and a figure is generated for each C60 usage group. These figures are displayed after the code. At this point, three axis exist for the data: wavelength, time, and EQE. The data is then averaged on the wavelength axis to create a plot of average EQE as a function of time. This plot is also displayed after the code.

Here starts the code:

```
'''
the following code requires the modules listed:
numpy as p
matplotlib.pyplot as plt
pandas pd
os
h5py

the code requires the following file structure:

parent:
---> dataAnalysis.py
---> data_fa25.h5


The log files lists operations that the code performs, including data omission, reading
files out of the H5 file, and more, Do note that it overwrites each time the code runs
and previous logs are deleted.

WARNINGS:
* do note that the code generates about 2400 files in the rawData folder, about 280 files
in the processedData folder, 4 images and a log file.
the large amount of files might overload some cloud based system, however, this has
not been tested


enjoy!
'''

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os
import h5py
```

```
from datetime import datetime

# Constants
h = 6.62607015e-34  # Planck’s constant (Joule second)
c = 3.0e8  # Speed of light (meters per second)
e = 1.602176634e-19  # Elementary charge (Coulombs)
active_area_cm2 = 0.14  # Active area in cm^2
active_area_m2 = active_area_cm2 * 1e-4  # Convert cm^2 to m^2

### need to make sure some dates start on 09_11
# array of dates for looping and file detection
dates = np.array(['2025_09_09', '2025_09_11', '2025_09_16', '2025_09_23', '2025_09_25',
    '2025_09_30', '2025_10_02', '2025_10_03', '2025_10_14', '2025_10_16', '2025_10_21',
    '2025_10_23', '2025_10_28', '2025_10_30'])

hours = np.array([0,48,168,336,384,504,552,576,840,888,1008,1056,1176,1224])
other_hours = np.array([0, 0, 168, 336, 336, 504, 504, 528, 840, 840, 1008, 1008, 1176, 1176])

# cell # in each set of C60 cycles
C60_00Cycle = np.array(['129', '131', '132', '134', '135', '136', '141', '145', '146',
    '161', '162', '170', '184', '190', '194'])
C60_05Cycle = np.array(['139', '143', '144', '157', '166', '172', '174', '176', '187',
    '189', '191', '193', '196'])
C60_10Cycle = np.array(['126', '127', '130', '133', '137', '138', '147', '158', '159',
    '163', '164', '169', '179', '180', '182', '183', '186', '195'])
all_cells = np.concatenate((C60_00Cycle, C60_05Cycle, C60_10Cycle))
all_cells.sort()

#USER SETTING VARIABLES -- SET BEFORE RUNNING
log_behavior = 'o' #'o' = overwrite, 'a' = append
pull_h5_data = True
process_data = True
h5_path = 'data_fa25.h5'

# upper and lower bounds for the wavelength region that is being observed
wavelength_upper_bound = 751
wavelength_lower_bound = 700

# cells and dates with bad data
black_list = np.array([
    ['2025_09_11', '192'],
    ['2025_10_02', '195'],
    ['2025_10_02', '184']
])


#print to signal to user that the processing has begun
print('running...\n')
```

```
#sets log file behavior set by variables above
if log_behavior == 'o':
    open('dataAnalysis.log', 'w').close()
else:
    print('not valid behavior specifier for log file, overwriting')
    open('dataAnalysis.log', 'w').close()

with open('dataAnalysis.log', 'a') as f:
    f.write(f'current operation started at {datetime.now()}\n')

#resuable function for slicing data
def slice_data(df, lower_range, upper_range):
    temp = df[df['Wavelength (nm)'] > lower_range]
    temp = temp[temp['Wavelength (nm)'] < upper_range]
    return temp

#function for getting power data
def get_power_data(hdf5_path, cell_number, date):
    '''
    Retrieve Power data (wavelength vs power) for a given cell and date.

    Power is measured once per cell (not per pixel).
    UPDATED FOR FA25: Cell numbers now use 3-digit padding.

    Parameters:
    -----------
    hdf5_path : str
        Path to the HDF5 file.
    cell_number : str or int
        Cell number (e.g., 129 or '129').
    date : str
        Date in the format 'YYYY_MM_DD' (e.g., '2025_09_23').

    Returns:
    --------
    pd.DataFrame or None
        DataFrame containing columns ['Wavelength (nm)', 'Power (W)'],
        or None if not found.

    Example:
    --------
    power_df = get_power_data(hdf5_path, 129, '2025_09_23')
    '''
    # FA25: Ensure 3-digit zero-padding
    cell_number = str(cell_number).zfill(3)

    with h5py.File(hdf5_path, 'r') as hdf:
        # Construct the full path to the Power data for this cell and date
        group_path = f'Cell{cell_number}/Power/{date}'

        # Check if the data exists at this path
        if group_path in hdf:
            group = hdf[group_path]
            # Read the numerical data array
            data = group['Data'][:]
            # Read and decode the column headers (stored as bytes)
            headers = [h.decode('utf-8') for h in group['Headers'][:]]
            # Create a pandas DataFrame with the data and headers
            df = pd.DataFrame(data, columns=headers)
            return df
        else:
            with open('dataAnalysis.log', 'a') as f:
                    f.write(f'Power data for Cell{cell_number} on {date} not found.\n')
            return None

#function for getting current data
def get_current_data(hdf5_path, cell_number, pixel_number, date):
    '''
    Retrieve Current data (wavelength vs current) for a specific cell, pixel, and date.

    Current is measured separately for each pixel.
    UPDATED FOR FA25: Cell numbers now use 3-digit padding.

    Parameters:
    -----------
    hdf5_path : str
        Path to the HDF5 file.
    cell_number : str or int
        Cell number (e.g., 129 or '129').
    pixel_number : int
        Pixel number (1-8 for FA25).
    date : str
        Date in the format 'YYYY_MM_DD'.

    Returns:
    --------
    pd.DataFrame or None
        DataFrame containing columns ['Wavelength (nm)', 'Current (A)'],
        or None if not found.

    Example:
    --------
    current_df = get_current_data(hdf5_path, 129, 4, '2025_09_23')
    '''
    # FA25: Ensure 3-digit zero-padding
    cell_number = str(cell_number).zfill(3)
    pixel_number = str(pixel_number)

    with h5py.File(hdf5_path, 'r') as hdf:
        # Construct the full path to the Current data
        group_path = f'Cell{cell_number}/Pixel{pixel_number}/Current/{date}'

        if group_path in hdf:
            group = hdf[group_path]
            data = group['Data'][:]
            headers = [h.decode('utf-8') for h in group['Headers'][:]]
            df = pd.DataFrame(data, columns=headers)
            return df
        else:
            with open('dataAnalysis.log', 'a') as f:
                f.write(f'Current data for Cell{cell_number}, Pixel{pixel_number} on {date}
                    not found.\n')
            return None

#function for writing data from h5 file to csvs
def retrieve_and_write_data(hdf5_path, cells, dates):
    '''
    batch retrieve and write data from HDF5 file as csv files

    creates files of the following form:
    YYYY_MM_DD_power_cell###.csv
    YYYY_MM_DD_current_cell###pixel#.csv
    '''
    #loops through all dates and cells
    for date in dates:
        for cell in cells:
            # Retrieve Power data (one per cell, not per pixel)
            power_data = get_power_data(hdf5_path, cell, date)
            if power_data is not None:
                # Create a descriptive variable name
                var_name = f'rawData/{date}_power_cell{cell}.csv'
                # Store the DataFrame as a global variable
                power_data.to_csv(var_name, index=False)

            # Retrieve Current data (specific to this pixel)
            pixel = 1
            while pixel <= 8:
                current_data = get_current_data(hdf5_path, cell, pixel, date)
                #current_data.drop(current_data.columns[0], axis=1)
                if current_data is not None:
                    var_name = f'rawData/{date}_current_cell{cell}_pixel{pixel}.csv'
                    current_data.to_csv(var_name, index=False)
                pixel += 1
```

```
#function for processing data pulled from the h5 file
def process_data():
    #variable init (arrays)
    TEMP_current_files = np.empty(0)
    TEMP_power_file = ''

    #variable init (dataframes)
    current_df = pd.DataFrame()
    power_df = pd.DataFrame()
    eqe = pd.DataFrame()

    #loops through all dates and cells
    for i, date in enumerate(dates):
        for j, cell in enumerate(all_cells):
            for file_name in os.listdir('rawData'):
                #if the file in question has the same date and cell# as the desired, adds
                    to list of file names
                if file_name == f'{date}_power_cell{cell}.csv':
                    TEMP_power_file = file_name
                if file_name[0:26] == f'{date}_current_cell{cell}':
                    TEMP_current_files = np.append(TEMP_current_files, file_name)
            if TEMP_current_files.size == 8 and TEMP_power_file != '':
                # reads power data from file
                power_df = pd.read_csv(f'rawData/{TEMP_power_file}')

                # reads wavelength data from second file in array, it does not matter which
                    file we read from, but the first entry in the array contains information
                    about the array type
                current_df['Wavelength (nm)'] = pd.read_csv(f'rawData/{date}_current_cell
                    {TEMP_current_files[1][23:26]}_pixel{1}.csv')['Wavelength (nm)']

                # loops over all files and adds current data to the dataframe
                i = 0
                while i < 8:
                    current_df[f'Pixel {i+1} Current (A)'] = pd.read_csv(f'rawData/{date}
                        _current_cell{TEMP_current_files[1][23:26]}_pixel{i+1}.csv')
                        ['Current (A)']
                    i += 1

                #inits eqe df's first column
                eqe['Wavelength (nm)'] = current_df['Wavelength (nm)']

                # loops over all the columns in the current df and does math to convert
                    it to eqe
                i = 0
                while i < 8:
                    eqe[f'EQE pixel {i+1}'] = ((current_df[f'Pixel {i+1} Current (A)'] /
                    power_df['Power (W)']) * (h * c / (e * current_df['Wavelength (nm)']*1e-9)
                    i += 1
                eqe = slice_data(eqe, wavelength_lower_bound, wavelength_upper_bound)
                # writes final EQE data for each date for each cell to a file in folder
                    processed data
                eqe.to_csv(f'processedData/eqe_results_{date}_cell{TEMP_current_files
                    [1][23:26]}.csv', index=False)
            else:
                #writes to log file that data processing was skipped
                with open('dataAnalysis.log', 'a') as f:
                    f.write(f'insufficient number of files for {date} and cell {cell}.
                        Found files are as follows: {TEMP_power_file} \n
                        {TEMP_current_files}\n')
            # clears temp variables for reuse
            TEMP_current_files = np.empty(0)
            TEMP_power_file = ''

#function for plotting data
def create_plot(data, sdev, size_x, size_y, title, save_file_name, plot_errors=True):
    #sets fig size
    plt.figure(figsize=(size_x, size_y))
    #loops through data in the given dataframe
    for i in range(len(data.columns) - 1):
        plt.plot(data['Wavelength (nm)'], data[data.columns[i]], color=((i/len(data.columns)),
            0, (len(data.columns) - i)/len(data.columns)))
        #adds error bars in wanted
        if plot_errors:
            plt.errorbar(data['Wavelength (nm)'], data[data.columns[i]],
                yerr=sdev[data.columns[i]],color=((i/len(data.columns)), 0,
                (len(data.columns) - i)/len(data.columns)), fmt='o', capsize=5, capthick=2)
    #variable init for labels
    labels = np.empty(0)
    #sets the labels for each line for data
    for i in range(len(data.columns)-1):
        labels = np.append(labels, data.columns[i][0:4] + ' '
            + data.columns[i][5:7] + ' ' + data.columns[i][8:10])

    #adds legend, title, and axis labels
    plt.legend(labels, loc='upper right')
    plt.title(title)
    plt.xlabel('Wavelength (nm)')
    plt.ylabel('EQE')
    plt.savefig(save_file_name)


print('pulling data from h5 file... \n')

#pulls data from h5 file and writes to csv's in folder rawData
```

```
if pull_h5_data: #check if user wants to pull data
    if not os.path.isdir('rawData'): #checks to see if the desired directory exists
        os.mkdir('rawData') #if not, make directory
        #add to log file that directory was created
        with open('dataAnalysis.log', 'a') as f:
            f.write(f'directory rawData did not exist. created directory rawData\n')
    if len(os.listdir('rawData')) == 0: #checks if data has been pulled before, if not:
        #pulls all data for cells and dates given in all_cells and dates
        retrieve_and_write_data(h5_path, all_cells, dates)
    else:
        with open('dataAnalysis.log', 'a') as f:
            f.write('skipping get data from h5 file as files already exist. \n')
else:
    with open('dataAnalysis.log', 'a') as f:
        f.write('skipping get data from h5 file as user specified \n')

print('done pulling data from h5 file. \n')
print('processing pulled data... \n')

#processes data from the h5 file
if process_data:
    if not os.path.isdir('processedData'): #checks if desired directory exists
        os.mkdir('processedData') #if not, make directory
        #add to log file that directory was created
        with open('dataAnalysis.log', 'a') as f:
            f.write(f'directory processedData did not exist. created directory
                processedData\n')
    if len(os.listdir('processedData')) == 0:
        process_data()
    else:
        #adds to log file that data processing is skipped
        with open('dataAnalysis.log', 'a') as f:
            f.write('skipping data processing as processed data already exists. \n')
else:
    #adds to log file that data processing is skipped as per user specified
    with open('dataAnalysis.log', 'a') as f:
        f.write('skipping data processing as user specified \n')

print('done processing data. \n')

#inits data frames for averaging mean values
mean_eqe_C60_00 = pd.DataFrame()
mean_eqe_C60_00_SD = pd.DataFrame()
mean_eqe_C60_00['Wavelength (nm)'] = pd.read_csv(f'processedData/{os.listdir
    ('processedData')[0]}')['Wavelength (nm)']

mean_eqe_C60_05 = pd.DataFrame()
mean_eqe_C60_05_SD = pd.DataFrame()
```

```
mean_eqe_C60_05['Wavelength (nm)'] = pd.read_csv(f'processedData/{os.listdir
    ('processedData')[0]}')['Wavelength (nm)']

mean_eqe_C60_10 = pd.DataFrame()
mean_eqe_C60_10_SD = pd.DataFrame()
mean_eqe_C60_10['Wavelength (nm)'] = pd.read_csv(f'processedData/{os.listdir
    ('processedData')[0]}')['Wavelength (nm)']

#checks if the output directory exists, if not, creates directory and adds to log file
if not os.path.isdir('outputData'):
    os.mkdir('outputData')
    with open('dataAnalysis.log', 'a') as f:
        f.write('Directory outputData does not exist, created directory outputData \n')

#calculates mean values for each date of each set of C60
for file in os.listdir('processedData'):
    if file[27:30] in C60_00Cycle:
        file_df = pd.read_csv(f'processedData/{file}')
        columns = file_df.columns
        columns = columns[1:]
        mean_eqe_C60_00[file[12:22]] = file_df[columns].mean(axis=1)
        mean_eqe_C60_00_SD[file[12:22]] = file_df[columns].std(axis=1)
    if file[27:30] in C60_05Cycle:
        file_df = pd.read_csv(f'processedData/{file}')
        columns = file_df.columns
        columns = columns[1:]
        mean_eqe_C60_05[file[12:22]] = file_df[columns].mean(axis=1)
        mean_eqe_C60_05_SD[file[12:22]] = file_df[columns].std(axis=1)
    if file[27:30] in C60_10Cycle:
        file_df = pd.read_csv(f'processedData/{file}')
        columns = file_df.columns
        columns = columns[1:]
        mean_eqe_C60_10[file[12:22]] = file_df[columns].mean(axis=1)
        mean_eqe_C60_10_SD[file[12:22]] = file_df[columns].std(axis=1)

#sorts columns by date, from 2025_09_09 to 2025_10_30
mean_eqe_C60_00 = mean_eqe_C60_00.reindex(sorted(mean_eqe_C60_00.columns), axis=1)
mean_eqe_C60_05 = mean_eqe_C60_05.reindex(sorted(mean_eqe_C60_05.columns), axis=1)
mean_eqe_C60_10 = mean_eqe_C60_10.reindex(sorted(mean_eqe_C60_10.columns), axis=1)

mean_eqe_C60_00_SD = mean_eqe_C60_00_SD.reindex(sorted(mean_eqe_C60_00_SD.columns), axis=1)
mean_eqe_C60_05_SD = mean_eqe_C60_05_SD.reindex(sorted(mean_eqe_C60_05_SD.columns), axis=1)
mean_eqe_C60_10_SD = mean_eqe_C60_10_SD.reindex(sorted(mean_eqe_C60_10_SD.columns), axis=1)

#writes to files
mean_eqe_C60_00.to_csv('outputData/C60_00_mean.csv', index=False)
mean_eqe_C60_05.to_csv('outputData/C60_05_mean.csv', index=False)
mean_eqe_C60_10.to_csv('outputData/C60_10_mean.csv', index=False)
```

```
mean_eqe_C60_00_SD.to_csv('outputData/C60_00_SD.csv', index=False)
mean_eqe_C60_00_SD.to_csv('outputData/C60_05_SD.csv', index=False)
mean_eqe_C60_00_SD.to_csv('outputData/C60_10_SD.csv', index=False)

#plots data
create_plot(mean_eqe_C60_00, mean_eqe_C60_00_SD, 10, 10, 'EQE of the C60 0 cycle set',
    'outputData/C60_00_cycle_day_average.png')
create_plot(mean_eqe_C60_05, mean_eqe_C60_05_SD, 10, 10, 'EQE of the C60 5 cycle set',
    'outputData/C60_05_cycle_day_average.png')
create_plot(mean_eqe_C60_10, mean_eqe_C60_10_SD, 10, 10, 'EQE of the C60 10 cycle set',
    'outputData/C60_10_cycle_day_average.png')

#inits mean of mean variables
mean_mean_eqe_00 = pd.DataFrame()
mean_mean_eqe_05 = pd.DataFrame()
mean_mean_eqe_10 = pd.DataFrame()
#sets mean of mean variables
mean_mean_eqe_00 = np.array(mean_eqe_C60_00.mean(axis=0))
mean_mean_eqe_05 = np.array(mean_eqe_C60_05.mean(axis=0))
mean_mean_eqe_10 = np.array(mean_eqe_C60_10.mean(axis=0))

mean_mean_eqe_05 = np.insert(mean_mean_eqe_05, 7, 0)

plot_hours = np.array([0, 168, 336, 504, 528, 840, 1008, 1176])

#manually edits arrays to account for the difference in starting times for the
stressing on tuesdays and thursdays
plot_data_00 = np.empty(0)
plot_data_00 = np.append(plot_data_00, [(mean_mean_eqe_00[0] + mean_mean_eqe_00[1])/2,
    mean_mean_eqe_00[2], (mean_mean_eqe_00[3] + mean_mean_eqe_00[4])/2,
    (mean_mean_eqe_00[5] + mean_mean_eqe_00[6])/2, mean_mean_eqe_00[7],
    (mean_mean_eqe_00[8] + mean_mean_eqe_00[9])/2, (mean_mean_eqe_00[10] +
    mean_mean_eqe_00[11])/2, (mean_mean_eqe_00[12] + mean_mean_eqe_00[13])/2])

plot_data_05 = np.empty(0)
plot_data_05 = np.append(plot_data_05, [(mean_mean_eqe_05[0] + mean_mean_eqe_05[1])/2,
    mean_mean_eqe_05[2], (mean_mean_eqe_05[3] + mean_mean_eqe_05[4])/2,
    (mean_mean_eqe_05[5] + mean_mean_eqe_05[6])/2, mean_mean_eqe_05[7],
    (mean_mean_eqe_05[8] + mean_mean_eqe_05[9])/2, (mean_mean_eqe_05[10] +
    mean_mean_eqe_05[11])/2, (mean_mean_eqe_05[12] + mean_mean_eqe_05[13])/2])

plot_data_10 = np.empty(0)
plot_data_10 = np.append(plot_data_10, [(mean_mean_eqe_10[0] + mean_mean_eqe_10[1])/2,
    mean_mean_eqe_10[2], (mean_mean_eqe_10[3] + mean_mean_eqe_10[4])/2,
    (mean_mean_eqe_10[5] + mean_mean_eqe_10[6])/2, mean_mean_eqe_10[7],
    (mean_mean_eqe_10[8] + mean_mean_eqe_10[9])/2, (mean_mean_eqe_10[10] +
    mean_mean_eqe_10[11])/2, (mean_mean_eqe_10[12] + mean_mean_eqe_10[13])/2])
```

```
#creates a final figure with all the data from mean of mean data
plt.figure(figsize=[10,10])
plt.plot(plot_hours, plot_data_00)
plt.plot(plot_hours, plot_data_05)
plt.plot(plot_hours, plot_data_10)
plt.plot(plot_hours[4], plot_data_05[4], 'bo')
plt.legend(['C60 0 cycle', 'C60 5 cycle', 'C60 10 cycle'])
plt.title('Comparison of C60 cycle sets')
plt.xlabel('Time (hr)')
plt.ylabel('EQE (%)')
plt.savefig('outputData/C60_cycles.png')

#signals to user that processing has ended
print('complete.')
```

Here ends the code.

Below are the figures generated by the code above. In order from first to last, the figures are of the C60 zero cycle usage group, C60 five cycle usage group, C60 ten cycle usage group, and the wavelength averaged EQE data for all three groups.

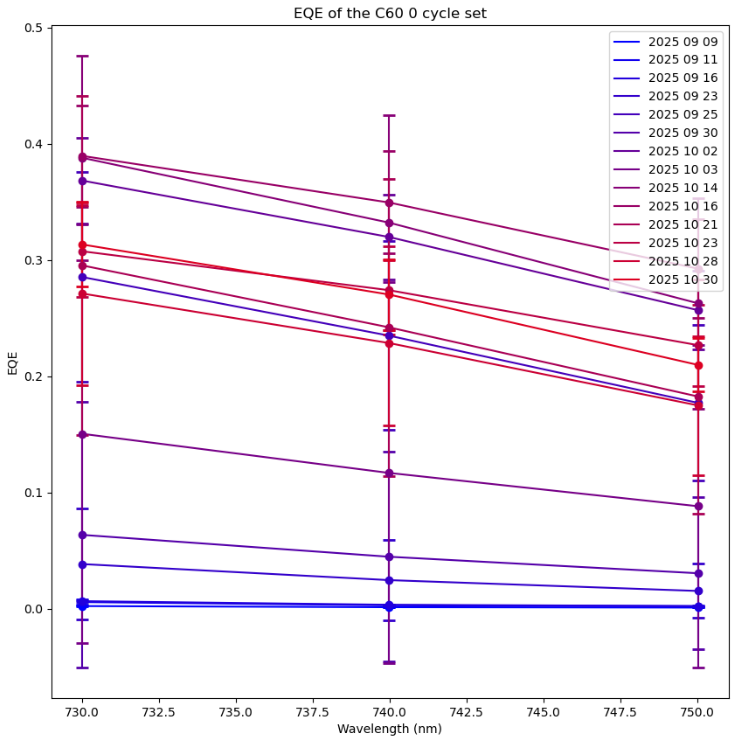

Figure 1: EQE of the C60 zero cycle group

The above graph shows the EQE data for C60 zero cycle as a function of wavelength. Time variation is shown through color difference with blue being earlier in time and red being later in time. While the color difference is not a exemplary method of displaying time variation, it is sufficent in this case.

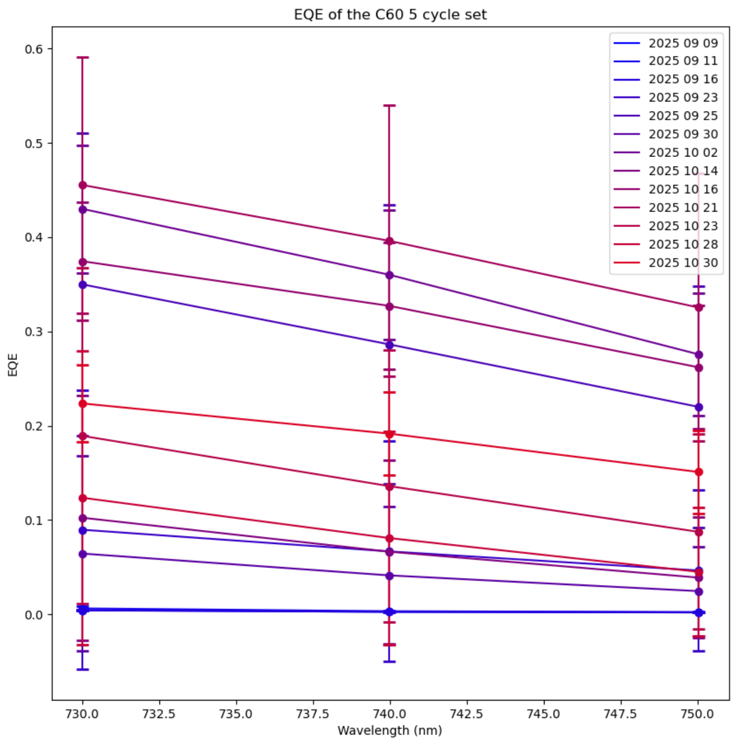

Figure 2: EQE of the C60 zero cycle group

The above graph shows the EQE data for C60 five cycle