algorithm_modification

September 20, 2024

0.1

[ ]:
```
import sys
import numpy as np
sys.path.append("C:\\Users\\RS\\VSCode\\matchedfiltermethod")

from matplotlib import pyplot as plt # type: ignore
import matplotlib.cm as cm
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

from MatchedFilter import matched_filter as mf
from MatchedFilter import generate_uas as gu
from MyFunctions import needed_function as nf
from MyFunctions import radiance_lut as rl
from MyFunctions import AHSI_data as ad
from MyFunctions import EMIT_data as ed
from MyFunctions.needed_function import open_unit_absorption_spectrum
```

0.2

[ ]:
```
def profile_matched_filter(base_array, data_array: np.array,␣
 ↪unit_absorption_spectrum: np.array) :
    #
    background_spectrum = base_array
    target_spectrum = background_spectrum*unit_absorption_spectrum
    concentration, _, _, _ = np.linalg.lstsq(target_spectrum[:, np.
 ↪newaxis],(data_array - background_spectrum), rcond=None)
    return concentration[0]


def profile_matched_filter_ML(base_array, data_array: np.
 ↪array,unit_absorption_spectrum: np.array) :
    #
    background_spectrum = base_array
    target_spectrum = background_spectrum*unit_absorption_spectrum[0]
```

```
    concentration, _, _, _ = np.linalg.lstsq(target_spectrum[:, np.
 ↪newaxis],(data_array - background_spectrum), rcond=None)

    threshold = 4000
    if concentration[0] > threshold and threshold < 33000:
        background_spectrum = background_spectrum + 4000*target_spectrum
        target_spectrum =␣
 ↪background_spectrum*unit_absorption_spectrum[threshold//4000]
        concentration, _, _, _ = np.linalg.lstsq(target_spectrum[:, np.
 ↪newaxis],(data_array - background_spectrum), rcond=None)
        concentration[0] = concentration[0] + threshold

    return concentration[0]


def matched_filter_with_fixed_bg(base_array, data_array: np.array,␣
 ↪unit_absorption_spectrum: np.array) :
    #
    bands,rows,cols = data_array.shape
    concentration = np.zeros((rows,cols))

    #
    background_spectrum = base_array
    target_spectrum = background_spectrum*unit_absorption_spectrum
    radiancediff_with_back = data_array - background_spectrum[:,np.newaxis,np.
 ↪newaxis]

    #
    d_covariance = radiancediff_with_back
    covariance = np.zeros((bands,bands))
    for i in range(rows):
        for j in range(cols):
            covariance += np.outer(d_covariance[:,i,j], d_covariance[:,i,j])
    covariance /= rows*cols
    covariance_inverse = np.linalg.pinv(covariance)

    #
    for i in range(rows):
        for j in range(cols):
            up = radiancediff_with_back[:,i,j].T @ covariance_inverse @␣
 ↪target_spectrum
            down = target_spectrum.T @ covariance_inverse @ target_spectrum
            concentration[i,j] = up / down

    return concentration
```

```
def ML_matched_filter_with_fixed_bg(base_array, data_array: np.array,␣
 ↪unit_absorption_spectrum: np.array) :
    #
    bands,rows,cols = data_array.shape
    concentration = np.zeros((rows,cols))

    #
    background_spectrum = base_array
    target_spectrum = background_spectrum*unit_absorption_spectrum[0]
    radiancediff_with_back = data_array - background_spectrum[:,np.newaxis,np.
 ↪newaxis]

    #
    d_covariance = radiancediff_with_back
    covariance = np.zeros((bands,bands))
    for i in range(rows):
        for j in range(cols):
            covariance += np.outer(d_covariance[:,i,j], d_covariance[:,i,j])
    covariance /= rows*cols
    covariance_inverse = np.linalg.inv(covariance)

    #
    for i in range(rows):
        for j in range(cols):
            up = (radiancediff_with_back[:,i,j].T @ covariance_inverse @␣
 ↪target_spectrum)
            down = target_spectrum.T @ covariance_inverse @ target_spectrum
            concentration[i,j] = up / down

    #
    background_spectrum = background_spectrum + 4000*target_spectrum
    target_spectrum = background_spectrum*unit_absorption_spectrum[1]
    radiancediff_with_back = data_array - background_spectrum[:,np.newaxis,np.
 ↪newaxis]
    for i in range(rows):
        for j in range(cols):
            if concentration[i,j] > 4000:
                up = (radiancediff_with_back[:,i,j].T @ covariance_inverse @␣
 ↪target_spectrum)
                down = target_spectrum.T @ covariance_inverse @ target_spectrum
                concentration[i,j] = up / down + 4000

    #
    background_spectrum = background_spectrum + 4000*target_spectrum
    target_spectrum = background_spectrum*unit_absorption_spectrum[2]
```

```
    radiancediff_with_back = data_array - background_spectrum[:,np.newaxis,np.
 ↪newaxis]
    for i in range(rows):
        for j in range(cols):
            if concentration[i,j] > 8000:
                up = (radiancediff_with_back[:,i,j].T @ covariance_inverse @␣
 ↪target_spectrum)
                down = target_spectrum.T @ covariance_inverse @ target_spectrum
                concentration[i,j] = up / down + 8000

    background_spectrum = background_spectrum + 4000*target_spectrum
    target_spectrum = background_spectrum*unit_absorption_spectrum[3]
    radiancediff_with_back = data_array - background_spectrum[:,np.newaxis,np.
 ↪newaxis]
    for i in range(rows):
        for j in range(cols):
            if concentration[i,j] > 12000:
                up = (radiancediff_with_back[:,i,j].T @ covariance_inverse @␣
 ↪target_spectrum)
                down = target_spectrum.T @ covariance_inverse @ target_spectrum
                concentration[i,j] = up / down + 12000

    background_spectrum = background_spectrum + 4000*target_spectrum
    target_spectrum = background_spectrum*unit_absorption_spectrum[4]
    radiancediff_with_back = data_array - background_spectrum[:,np.newaxis,np.
 ↪newaxis]
    for i in range(rows):
        for j in range(cols):
            if concentration[i,j] > 16000:
                up = (radiancediff_with_back[:,i,j].T @ covariance_inverse @␣
 ↪target_spectrum)
                down = target_spectrum.T @ covariance_inverse @ target_spectrum
                concentration[i,j] = up / down + 16000

    return concentration


def image_simulation(plume, lower_wavelength, upper_wavelength, row_num,␣
 ↪col_num,noise_level):
    # Load the simulated emit radiance spectrum
    total_bands,lut = rl.load_lookup_table("C:
 ↪\\Users\\RS\\VSCode\matchedfiltermethod\\MyData\\enhanced_radiance\\AHSI_rad_lookup_table.
 ↪npz")

    bands,unenhanced_radiance = rl.
 ↪lookup_spectrum(0,total_bands,lut,lower_wavelength,upper_wavelength)
```

```
    # Set the shape of the image that want to simulate
    band_num = len(bands)
    simulated_image = np.zeros([band_num, row_num, col_num])
    # Generate the universal radiance cube image
    for i in range(row_num) :
        for j in range(col_num) :
            if plume[i,j] > 0:
                _,enhanced_radiance = rl.
↪lookup_spectrum(plume[i,j],total_bands,lut,lower_wavelength,upper_wavelength)
                simulated_image[:,i,j] = enhanced_radiance
            else:
                simulated_image[:,i,j] = unenhanced_radiance
    #      1%
    noise_std = noise_level * simulated_image  #        1%
    noise = np.random.normal(0, noise_std)
    simulated_noisy_image = simulated_image + noise

    return simulated_noisy_image


def enhancement_2perc():
    """
      2%        0-20000ppmm
    """
    np.random.seed(42)  #
    matrix_size = 200
    num_pixels = matrix_size * matrix_size
    num_enhanced_pixels = int(0.02 * num_pixels)  # 2%

    #   2%
    indices = np.random.choice(num_pixels, num_enhanced_pixels, replace=False)

    #            0-20000 ppmm
    random_enhancement_values = np.random.uniform(0, 20000, num_enhanced_pixels)

    #
    plume = np.zeros((matrix_size, matrix_size))
    np.put(plume, indices, random_enhancement_values)

    #
    all_indices = np.arange(plume.size)
    unenhanced_indices = np.setdiff1d(all_indices, indices)
    enhanced_mask = np.unravel_index(indices, (matrix_size, matrix_size))
    unenhanced_mask = np.unravel_index(unenhanced_indices, (matrix_size,␣
↪matrix_size))

    #
```

```
    simulated_image = image_simulation(plume, 2150, 2500, 200, 200, 0.01)

    return simulated_image, enhanced_mask, unenhanced_mask,␣
 ↪random_enhancement_values


def polyfit_plot(enhancements,resultlist,ax,labelstr):
    slope,intercept = np.polyfit(enhancements,resultlist,1)
    x_fit = np.linspace(min(enhancements), max(enhancements), 100)
    y_fit = slope * x_fit + intercept
    if intercept > 0 :
        ax.plot(x_fit,y_fit,label=f'{labelstr}:y = {slope:.2f}x + {np.
 ↪abs(intercept):.2f}' )
    elif intercept < 0 :
        ax.plot(x_fit,y_fit,label=f'{labelstr}:y = {slope:.2f}x - {np.
 ↪abs(intercept):.2f}' )
    else:
        ax.plot(x_fit,y_fit,label=f'{labelstr}:y = {slope:.2f}x' )


def set_plot_details(ax, title, xlabel, ylabel):
    ax.set_title(title)
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    ax.legend()


def test(method,ax,*args):
    tif_file_path = r"F:\\AHSI_part1\\GF5B_AHSI_E100.0_N26.
 ↪4_20231004_011029_L10000400374\\GF5B_AHSI_E100.0_N26.
 ↪4_20231004_011029_L10000400374_SW.tif"
    bands,radiance = ad.get_calibrated_radiance(tif_file_path,2100,2500)
    # define the path of the unit absorption spectrum file
    ahsi_unit_absorption_spectrum_path = r"C:
 ↪\\Users\\RS\\VSCode\\matchedfiltermethod\\MyData\\uas\\AHSI_UAS_end_50000.
 ↪txt"
    bands, uas =␣
 ↪open_unit_absorption_spectrum(ahsi_unit_absorption_spectrum_path,2100,2500)
    # call the main function to process the radiance file
    c = method(radiance,uas,*args)
    #
    x = np.arange(c.shape[1])  #
    y = np.arange(c.shape[0])  #
    X, Y = np.meshgrid(x, y)
    contour = ax.contourf(X, Y, c, 20, cmap='RdGy')
```

1

1.1

1.1.1

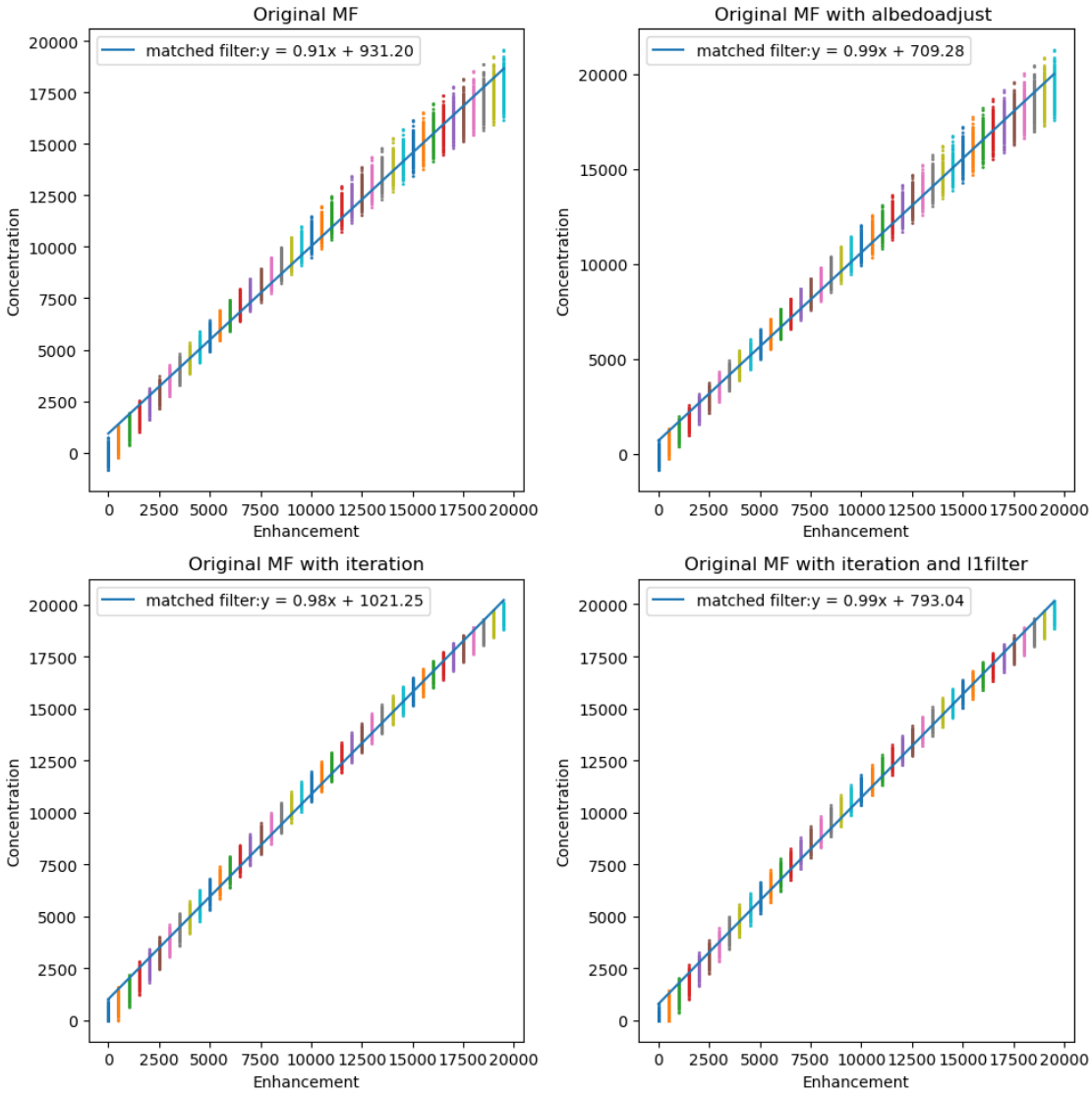
[ ]:
```
# original matched filter algorithm
def matched_filter(data_cube: np.array, unit_absorption_spectrum: np.array,␣
 ↪albedoadjust, iterate, sparsity):
    """
    Calculate the methane enhancement of the image data based on the original␣
 ↪matched filter
    and the unit absorption spectrum.

    :param data_array: numpy array of the image data
    :param unit_absorption_spectrum: list of the unit absorption spectrum
    :param albedoadjust: bool, whether to adjust the albedo
    :param iterate: bool, whether to iterate
    :param sparsity: bool, whether to use l1filter

    :return: numpy array of the methane enhancement
    """
    #           concentration
    bands, rows, cols = data_cube.shape
    concentration = np.zeros((rows, cols))

    #
    background_spectrum = np.nanmean(data_cube, axis=(1,2))
    target_spectrum = background_spectrum*unit_absorption_spectrum
    radiancediff_with_bg = data_cube - background_spectrum[:, None,None]

    #
    d_covariance = radiancediff_with_bg
    covariance = np.zeros((bands, bands))
    for row in range(rows):
        for col in range(cols):
            covariance += np.outer(d_covariance[:, row, col], d_covariance[:,␣
 ↪row, col])
    covariance = covariance/(rows*cols)
    covariance_inverse = np.linalg.pinv(covariance)

    #
    albedo = np.ones((rows, cols))
    if albedoadjust:
        for row in range(rows):
            for col in range(cols):
                albedo[row, col] = (
                        (data_cube[:,row,col].T @ background_spectrum) /
```

```
                        (background_spectrum.T @ background_spectrum)
                )

    #
    for row in range(rows):
        for col in range(cols):
            numerator = (radiancediff_with_bg[:,row,col].T @ covariance_inverse␣
↪@ target_spectrum)
            denominator = albedo[row,col]*(target_spectrum.T @␣
↪covariance_inverse @ target_spectrum)
            concentration[row,col] = numerator/denominator

    #
    if iterate:
        #    l1filter
        l1filter = np.zeros((rows,cols))
        for iter_num in range(5):
            #     l1filter           l1filter
            if sparsity:
                for row in range(rows):
                    for col in range(cols):
                        l1filter[row,col] = 1 / (concentration[row,col] + np.
↪finfo(np.float64).tiny)

            #       ,
            background_spectrum = np.mean(data_cube -␣
↪(albedo*concentration)[None,:,:]*target_spectrum[:,None,None],
                                          axis=(1,2))
            target_spectrum = np.multiply(background_spectrum,␣
↪unit_absorption_spectrum)
            radiancediff_with_bg = data_cube - background_spectrum[:,None,None]

            #
            d_covariance = data_cube -(albedo*concentration)[None,:,:
↪]*target_spectrum[:,None,None] - background_spectrum[:,None,None]
            covariance = np.zeros((bands, bands))
            for row in range(rows):
                for col in range(cols):
                    covariance += np.outer(d_covariance[:,row,col],␣
↪d_covariance[:,row,col])
            covariance = covariance/(rows*cols)
            covariance_inverse = np.linalg.pinv(covariance)

            #
            for row in range(rows):
                for col in range(cols):
```

```
                    numerator = (radiancediff_with_bg[:,row,col].T @␣
 ↪covariance_inverse @ target_spectrum) - l1filter[row,col]
                    denominator = albedo[row,col] * (target_spectrum.T @␣
 ↪covariance_inverse @ target_spectrum)
                    concentration[row,col] = np.maximum(numerator /␣
 ↪denominator, 0.0)

    return concentration


def mf_run_plot(ax,*args):
    _,uas = gu.generate_range_uas_AHSI(0,36000,2150,2500)
    resultlist = []
    enhancements = np.arange(0,20000,500)
    for enhancement in enhancements:
        #  2%
        simulated_image,enhanced_mask,unenhanced_mask =␣
 ↪enhancement_2perc(enhancement)
        #
        result = matched_filter(simulated_image,uas,*args)
        #
        enhanced = result[enhanced_mask]
        unenhanced = result[unenhanced_mask]
        resultlist.append(np.mean(enhanced))
        ax.plot(enhancement*np.ones(len(enhanced)), enhanced, marker='o',␣
 ↪markersize = 1, linestyle='None')

    polyfit_plot(enhancements,resultlist,ax,"matched filter")


def matched_filter_test():
    fig,axes = plt.subplots(2,2,figsize=(10,10))
    ax1,ax2,ax3,ax4 = axes.flatten()
    mf_run_plot(ax1,False,False,False)
    mf_run_plot(ax2,True,False,False)
    mf_run_plot(ax3,False,True,False)
    mf_run_plot(ax4,False,True,True)
    set_plot_details(ax1, "Original MF", "Enhancement", "Concentration")
    set_plot_details(ax2, "Original MF with albedoadjust ", "Enhancement",␣
 ↪"Concentration")
    set_plot_details(ax3, "Original MF with iteration", "Enhancement",␣
 ↪"Concentration")
    set_plot_details(ax4, "Original MF with iteration and l1filter",␣
 ↪"Enhancement", "Concentration")

    plt.tight_layout()
```

```
    plt.show()

    return None

matched_filter_test()
```

```
C:\Users\RS\AppData\Local\Temp\ipykernel_36536\3663452450.py:81: RuntimeWarning:
overflow encountered in scalar divide
  concentration[row,col] = np.maximum(numerator / denominator, 0.0)
```



1.1.2 column-wise

[ ]:
```
# orginal matched filter algorithm
def columnwise_matched_filter(data_cube: np.array, unit_absorption_spectrum: np.
 ↪array, iterate=False,
                  albedoadjust=False, sparsity=False):
    """
    Calculate the methane enhancement of the image data based on the original␣
 ↪matched filter
    and the unit absorption spectrum.

    :param data_array: numpy array of the image data
    :param unit_absorption_spectrum: list of the unit absorption spectrum
    :param albedoadjust: bool, whether to adjust the albedo
    :param iterate: bool, whether to iterate
    :param sparsity: bool, whether to use l1filter
    :return: numpy array of the methane enhancement
    """
    #         ,    concentration
    bands, rows, cols = data_cube.shape
    concentration = np.zeros((rows, cols))

    #
    for col_index in range(cols):
        #
        current_column = data_cube[:, :, col_index]
        valid_rows = ~np.isnan(current_column[0, :])
        count_not_nan = np.count_nonzero(valid_rows)

        #           nan
        if count_not_nan == 0:
            concentration[:, col_index] = np.nan
            continue

        #                   ,
        background_spectrum = np.nanmean(current_column, axis=1)
        target_spectrum = background_spectrum*unit_absorption_spectrum
        radiancediff_with_bg = current_column[:, valid_rows] -␣
 ↪background_spectrum[:, None]

        #
        d_covariance = radiancediff_with_bg
        covariance = np.zeros((bands, bands))
        for i in range(count_not_nan):
            covariance += np.outer(d_covariance[:, i], d_covariance[:, i])
        covariance = covariance/count_not_nan
        covariance_inverse = np.linalg.pinv(covariance)
```

```
        #
        albedo = np.ones((rows, cols))
        if albedoadjust:
            albedo[valid_rows, col_index] = (
                    (current_column[:, valid_rows].T @ background_spectrum) /
                    (background_spectrum.T @ background_spectrum)
            )

        #
        numerator = (radiancediff_with_bg.T @ covariance_inverse @␣
↪target_spectrum)
        denominator = albedo[valid_rows, col_index] * (target_spectrum.T @␣
↪covariance_inverse @ target_spectrum)
        concentration[valid_rows, col_index] = numerator/denominator

        #
        if iterate:
            #    l1filter
            l1filter = np.zeros((rows, cols))
            epsilon = np.finfo(np.float64).tiny
            #
            for iter_num in range(5):
                print("iteration: No.", iter_num + 1)
                #     l1filter           l1filter
                if sparsity:
                    l1filter[valid_rows, col_index] = 1 /␣
↪(concentration[valid_rows, col_index] + epsilon)

                #
                background_spectrum = np.nanmean(current_column[:, valid_rows]␣
↪- (albedo[valid_rows, col_index] *concentration[valid_rows,␣
↪col_index])[None,:]*target_spectrum[:,None],
                                                 axis=1)
                target_spectrum = np.multiply(background_spectrum,␣
↪unit_absorption_spectrum)
                radiancediff_with_bg = current_column[:, valid_rows] -␣
↪background_spectrum

                #
                d_covariance = current_column[:, valid_rows]␣
↪-(albedo[valid_rows, col_index] *concentration[valid_rows, col_index])[None,:
↪]*target_spectrum[:,None] - background_spectrum[:,None]
                covariance = np.zeros((bands, bands))
                for i in range(valid_rows.shape[0]):
```

```
                    covariance += np.outer(d_covariance[:, i], d_covariance[:,␣
↪i])
                covariance = covariance/count_not_nan
                covariance_inverse = np.linalg.pinv(covariance)

                #
                numerator = (radiancediff_with_bg.T @ covariance_inverse @␣
↪target_spectrum) - l1filter[valid_rows, col_index]
                denominator = albedo[valid_rows, col_index] * (target_spectrum.
↪T @ covariance_inverse @ target_spectrum)
                concentration[valid_rows, col_index] = np.maximum(numerator /␣
↪denominator, 0.0)

    #
    return concentration
```

1.2

1.2.1

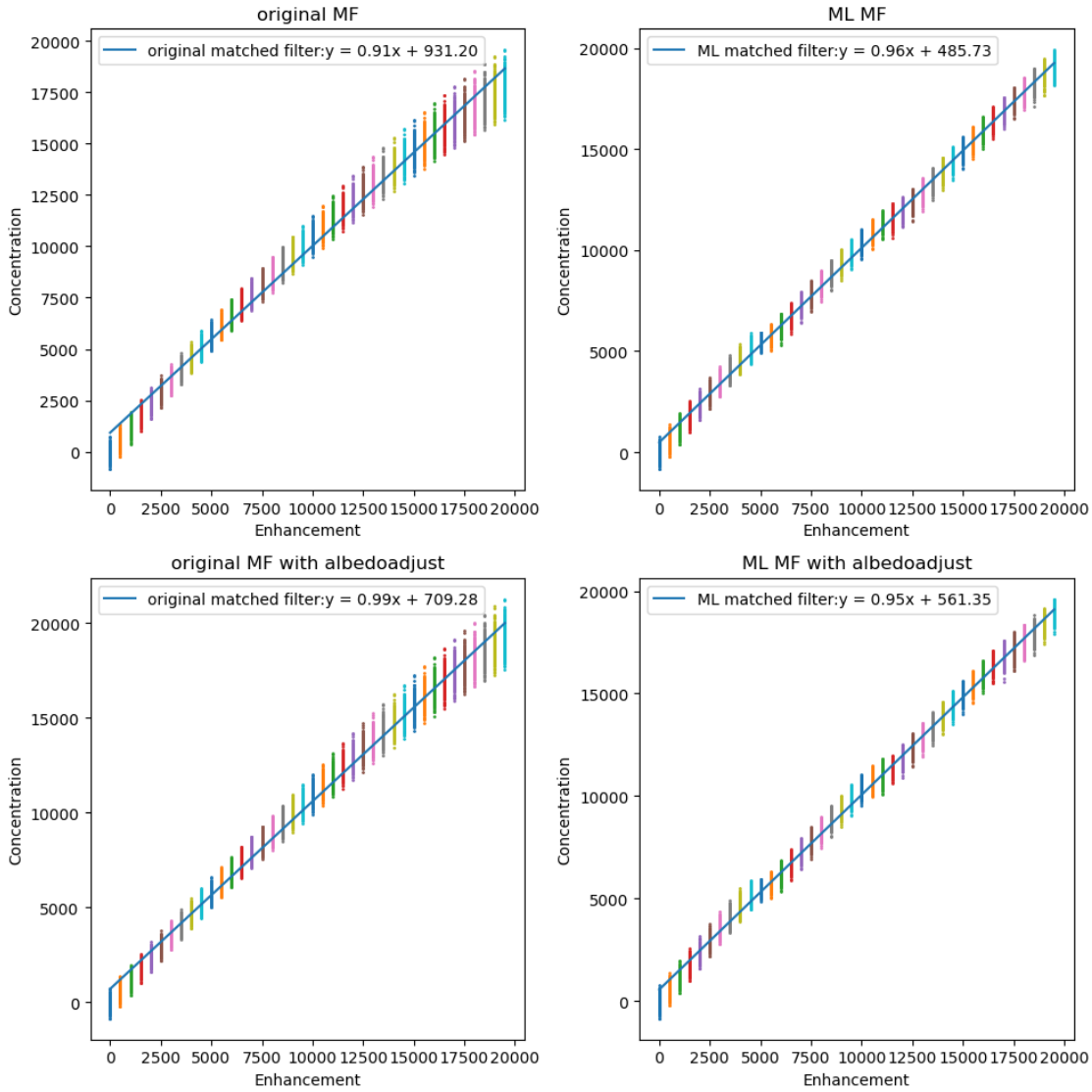
[ ]:
```
def ML_matched_filter(data_cube: np.array, unit_absorption_spectrum: np.
↪array,albedoadjust) -> np.array:
    """
    Calculate the methane enhancement of the image data based on the modified␣
↪matched filter
    and the unit absorption spectrum.

    :param data_cube: numpy array of the image data
    :param unit_absorption_spectrum: list of the unit absorption spectrum
    :param albedoadjust: bool, whether to adjust the albedo
    :param sparsity: bool, whether to use l1filter
    :return: numpy array of methane enhancement result
    """
    #           concentration
    bands, rows, cols = data_cube.shape
    concentration = np.zeros((rows, cols))

    #
    background_spectrum = np.nanmean(data_cube, axis=(1,2))
    target_spectrum = background_spectrum*unit_absorption_spectrum[0]
    radiancediff_with_bg = data_cube - background_spectrum[:, None,None]

    #
    d_covariance = radiancediff_with_bg
    covariance = np.zeros((bands, bands))
    for row in range(rows):
        for col in range(cols):
```

```
            covariance += np.outer(d_covariance[:, row, col], d_covariance[:,␣
↪row, col])
    covariance = covariance/(rows*cols)
    covariance_inverse = np.linalg.pinv(covariance)

    #
    albedo = np.ones((rows, cols))
    if albedoadjust:
        for row in range(rows):
            for col in range(cols):
                albedo[row, col] = (
                        (data_cube[:,row,col].T @ background_spectrum) /
                        (background_spectrum.T @ background_spectrum)
                )

    #
    for row in range(rows):
        for col in range(cols):
            numerator = (radiancediff_with_bg[:,row,col].T @ covariance_inverse␣
↪@ target_spectrum)
            denominator = albedo[row,col]*(target_spectrum.T @␣
↪covariance_inverse @ target_spectrum)
            concentration[row,col] = numerator/denominator

    #
    original_concentration = concentration.copy()

    #
    levelon = True
    adaptive_threshold = 6000
    i = 1
    high_concentration_mask = original_concentration > adaptive_threshold*(0.
↪99**i)
    low_concentration_mask = original_concentration <= adaptive_threshold*(0.
↪99**i)

    while levelon:
        if np.sum(high_concentration_mask) > 0 and adaptive_threshold < 32000:
            background_spectrum = np.nanmean(data_cube - concentration *␣
↪target_spectrum[:, None, None], axis=(1, 2))
            target_spectrum = background_spectrum * unit_absorption_spectrum[0]

            new_background_spectrum = background_spectrum
            for n in range(i):
                new_background_spectrum +=␣
↪6000*new_background_spectrum*unit_absorption_spectrum[n]
```

```
            high_target_spectrum = new_background_spectrum *␣
↪unit_absorption_spectrum[i]

            radiancediff_with_bg[:,high_concentration_mask] = (
                data_cube[:,high_concentration_mask] - new_background_spectrum[:
↪,None]
            )
            radiancediff_with_bg[:,low_concentration_mask] = (
                data_cube[:,low_concentration_mask] - background_spectrum[:
↪,None]
            )

            # d_covariance[:,high_concentration_mask] = data_cube[:
↪,high_concentration_mask] - (
            #     ␣
↪(concentration[high_concentration_mask]-adaptive_threshold)*high_target_spectrum[:
↪,None] + new_background_spectrum[:,None]
            # )
            # d_covariance[:,high_concentration_mask] = data_cube[:
↪,high_concentration_mask] - (
            #     background_spectrum[:,None] +␣
↪concentration[high_concentration_mask]*target_spectrum[:,None]
            # )

            # d_covariance[:,low_concentration_mask] = data_cube[:
↪,low_concentration_mask] - (
            #     background_spectrum[:,None] +␣
↪concentration[low_concentration_mask]*target_spectrum[:,None]
            # )
            # covariance = np.zeros((bands, bands))
            # for row in range(rows):
            #     for col in range(cols):
            #         covariance += np.outer(d_covariance[:, row, col],␣
↪d_covariance[:, row, col])
            # covariance /= rows*cols
            # covariance_inverse = np.linalg.pinv(covariance)

            # concentration[high_concentration_mask] = (
            #     (radiancediff_with_bg[:, high_concentration_mask].T @␣
↪covariance_inverse @high_target_spectrum) /
            #     (high_target_spectrum.T @ covariance_inverse @␣
↪high_target_spectrum)
            # ) + adaptive_threshold

            concentration[high_concentration_mask] = (
```

```
                (radiancediff_with_bg[:, high_concentration_mask].T␣
↪@high_target_spectrum) /
                (high_target_spectrum.T  @ high_target_spectrum)
            ) + adaptive_threshold


            high_concentration_mask = original_concentration >␣
↪adaptive_threshold*0.99
            low_concentration_mask = original_concentration <=␣
↪adaptive_threshold*0.99

            adaptive_threshold += 6000
            i += 1

        else:
            levelon = False

    return original_concentration,concentration


def MLmf_run_plot(ax1,ax2,*args):
    uaslist = []
    _,uas = gu.generate_range_uas_AHSI(0,36000,2150,2500)
    uaslist.append(uas)
    uasrange = np.arange(4000,46000,4000)
    for i in uasrange:
        _,uas = gu.generate_range_uas_AHSI(i,i+6000,2150,2500)
        uaslist.append(uas)

    resultlist = []
    resultlist2 = []
    enhancements = np.arange(0,20000,500)
    for enhancement in enhancements:
        #  2%
        simulated_image,enhanced_mask,unenhanced_mask =␣
↪enhancement_2perc(enhancement)
        #
        originalresult,result = ML_matched_filter(simulated_image,uaslist,*args)
        #

        enhanced = originalresult[enhanced_mask]
        unenhanced = originalresult[unenhanced_mask]
        resultlist.append(np.mean(enhanced))
        ax1.plot(enhancement*np.ones(len(enhanced)), enhanced, marker='o',␣
↪markersize = 1, linestyle='None')

        enhanced2 = result[enhanced_mask]
```

```
        unenhanced2 = result[unenhanced_mask]
        resultlist2.append(np.mean(enhanced2))
        ax2.plot(enhancement*np.ones(len(enhanced2)), enhanced2, marker='o',␣
 ↪markersize = 1, linestyle='None')
    polyfit_plot(enhancements,resultlist,ax1,"original matched filter")
    polyfit_plot(enhancements,resultlist2,ax2,"ML matched filter")


def ML_matched_filter_test():
    fig,axes = plt.subplots(2,2,figsize=(10,10))
    ax1,ax2,ax3,ax4 = axes.flatten()

    MLmf_run_plot(ax1,ax2,False)
    MLmf_run_plot(ax3,ax4,True)


    set_plot_details(ax1, "original MF", "Enhancement", "Concentration")
    set_plot_details(ax2, "ML MF", "Enhancement", "Concentration")
    set_plot_details(ax3, "original MF with albedoadjust", "Enhancement",␣
 ↪"Concentration")
    set_plot_details(ax4, "ML MF with albedoadjust", "Enhancement",␣
 ↪"Concentration")

    plt.tight_layout()
    plt.show()

    return None

ML_matched_filter_test()
```



1.2.2 column-wise

[ ]:
```
# modified matched filter algorithm
def columnwise_ML_matched_filter(data_array: np.array,␣
↪stacked_unit_absorption_spectrum: np.array, is_iterate=False,
                  is_albedo=False, is_filter=False, is_columnwise=False) -> np.
↪array:
    """
    Calculate the methane enhancement of the image data based on the original␣
↪matched filter
    and the unit absorption spectrum.
```

```
    :param data_array: numpy array of the image data
    :param unit_absorption_spectrum: list of the unit absorption spectrum
    :param is_iterate: flag to decide whether to iterate the matched filter
    :param is_albedo: flag to decide whether to do the albedo correction
    :param is_filter: flag to decide whether to add the l1-filter correction
    :return: numpy array of methane enhancement result
    """
    #
    bands, rows, cols = data_array.shape
    #    concentration
    concentration = np.zeros((rows, cols))
    #
    if is_columnwise:
        for col_index in range(cols):
            #
            current_column = data_array[:, :, col_index]
            #
            valid_rows = ~np.isnan(current_column[0, :])
            count_not_nan = np.count_nonzero(valid_rows)
            #           nan
            if count_not_nan == 0:
                concentration[:, col_index] = np.nan
                continue

            #
            background_spectrum = np.nanmean(current_column, axis=1)
            target_spectrum =␣
↪background_spectrum*stacked_unit_absorption_spectrum[0,:]

            #
            radiancediff_with_bg = current_column[:, valid_rows] -␣
↪background_spectrum[:, None]
            covariance = np.zeros((bands, bands))
            for i in range(count_not_nan):
                covariance += np.outer(radiancediff_with_bg[:, i],␣
↪radiancediff_with_bg[:, i])
            covariance = covariance/count_not_nan
            covariance_inverse = np.linalg.inv(covariance)

            #
            albedo = np.ones((rows, cols))
            if is_albedo:
                albedo[valid_rows, col_index] = (
                        (current_column[:, valid_rows].T @ background_spectrum)␣
↪/
                        (background_spectrum.T @ background_spectrum)
                )
```

```
            #
            up = (radiancediff_with_bg.T @ covariance_inverse @ target_spectrum)
            down = albedo[valid_rows, col_index] * (target_spectrum.T @␣
↪covariance_inverse @ target_spectrum)
            concentration[valid_rows, col_index] = up / down

            levelon = True
            #
            mean_concentration = np.nanmean(concentration[valid_rows,␣
↪col_index])  #   NaN
            std_concentration = np.nanstd(concentration[valid_rows, col_index])␣
↪   #   NaN
            #
            adaptive_threshold = mean_concentration + std_concentration
            while levelon:
                high_concentration_mask = concentration[valid_rows, col_index]␣
↪> adaptive_threshold
                #
                #
                background_spectrum = np.nanmean(current_column[:,valid_rows] +␣
↪albedo[valid_rows,col_index]*concentration[valid_rows,col_index]*target_spectrum[:
↪, np.newaxis], axis=1)
                background_spectrum = background_spectrum +␣
↪adaptive_threshold*stacked_unit_absorption_spectrum[1,:]
                target_spectrum = np.multiply(background_spectrum,␣
↪stacked_unit_absorption_spectrum[1,:])
                radiancediff_with_bg = current_column[:, valid_rows] -␣
↪background_spectrum[:, None] -␣
↪albedo[valid_rows,col_index]*(concentration[valid_rows,col_index]-adaptive_threshold)*targe
↪, np.newaxis]
                covariance = np.zeros((bands, bands))
                for i in range(valid_rows.shape[0]):
                    covariance += np.outer(radiancediff_with_bg[:, i],␣
↪radiancediff_with_bg[:, i])
                covariance = covariance/count_not_nan
                covariance_inverse = np.linalg.inv(covariance)
                #
                up = (radiancediff_with_bg[:, high_concentration_mask].T @␣
↪covariance_inverse @ target_spectrum)
                down = albedo[valid_rows, col_index][high_concentration_mask] *␣
↪(target_spectrum.T @ covariance_inverse @ target_spectrum)
                #
                valid_indices = np.where(valid_rows)[0]
                high_concentration_indices =␣
↪valid_indices[high_concentration_mask]
```

```
                concentration[high_concentration_indices, col_index] = up /␣
↪down + adaptive_threshold
                #
                mean_concentration = np.nanmean(concentration[valid_rows,␣
↪col_index])  #   NaN
                std_concentration = np.nanstd(concentration[valid_rows,␣
↪col_index])    #   NaN
                #
                new_adaptive_threshold = mean_concentration + std_concentration
                if np.abs((new_adaptive_threshold-adaptive_threshold)/
↪adaptive_threshold) < 0.1:
                    adaptive_threshold = new_adaptive_threshold
                else:
                    levelon = False

            #
            if is_iterate:
                l1filter = np.zeros((rows, cols))
                epsilon = np.finfo(np.float32).tiny
                for iter_num in range(5):
                    if is_filter:
                        l1filter[valid_rows, col_index] = 1 /␣
↪(concentration[valid_rows, col_index] + epsilon)
                    else:
                        l1filter[valid_rows, col_index] = 0

                    #
                    column_replacement = current_column[:, valid_rows] -␣
↪(albedo[valid_rows, col_index] *concentration[valid_rows, col_index])[None,:
↪]*target_spectrum[:,None]
                    #
                    background_spectrum = np.mean(column_replacement, axis=1)
                    target_spectrum = np.multiply(background_spectrum,␣
↪stacked_unit_absorption_spectrum[0,:])
                    #
                    radiancediff_with_bg = current_column[:, valid_rows]␣
↪-(albedo[valid_rows, col_index] *concentration[valid_rows, col_index])[None,:
↪]*target_spectrum[:,None] - background_spectrum[:,None]
                    covariance = np.zeros((bands, bands))
                    for i in range(valid_rows.shape[0]):
                        covariance += np.outer(radiancediff_with_bg[:, i],␣
↪radiancediff_with_bg[:, i])
                    covariance = covariance/count_not_nan
                    covariance_inverse = np.linalg.inv(covariance)

                    #
```

```
                    up = (radiancediff_with_bg.T @ covariance_inverse @␣
↪target_spectrum) - l1filter[valid_rows, col_index]
                    down = albedo[valid_rows, col_index] * (target_spectrum.T @␣
↪covariance_inverse @ target_spectrum)
                    concentration[valid_rows, col_index] = np.maximum(up /␣
↪down, 0.0)
                    high_concentration_mask = concentration[valid_rows,␣
↪col_index] > 5000

                    if np.any(high_concentration_mask):
                        #
                        con = concentration[valid_rows, col_index].copy()
                        background_spectrum = np.nanmean(current_column[:
↪,valid_rows] - albedo[valid_rows,col_index]*con*target_spectrum[:, np.
↪newaxis], axis=1)
                        target_spectrum = np.multiply(background_spectrum,␣
↪stacked_unit_absorption_spectrum)
                        radiancediff_with_bg = current_column[:, valid_rows]␣
↪-albedo[valid_rows,col_index]*con*target_spectrum[:, np.newaxis] -␣
↪background_spectrum[:, None]
                        covariance = np.zeros((bands, bands))
                        for i in range(valid_rows.shape[0]):
                            covariance += np.outer(radiancediff_with_bg[:, i],␣
↪radiancediff_with_bg[:, i])
                        covariance = covariance/count_not_nan
                        covariance_inverse = np.linalg.inv(covariance)
                        #
                        up = (radiancediff_with_bg[:, high_concentration_mask].
↪T @ covariance_inverse @ target_spectrum)
                        down = albedo[valid_rows,␣
↪col_index][high_concentration_mask] * (target_spectrum.T @␣
↪covariance_inverse @ target_spectrum)
                        #
                        valid_indices = np.where(valid_rows)[0]
                        high_concentration_indices =␣
↪valid_indices[high_concentration_mask]
                        concentration[high_concentration_indices, col_index] =␣
↪up / down + 2500

    if not is_columnwise:
        count_not_nan = np.count_nonzero(~np.isnan(data_array[0, :, :]))
        background_spectrum = np.nanmean(data_array, axis=(1,2))
        target_spectrum = np.multiply(background_spectrum,␣
↪stacked_unit_absorption_spectrum[0,:])
        radiancediff_with_bg = data_array - background_spectrum[:, None, None]
        covariance = np.zeros((bands, bands))
```

```
        for i in range(rows):
            for j in range(cols):
                covariance = covariance + np.outer(radiancediff_with_bg[:, i,␣
↪j], radiancediff_with_bg[:, i, j])
        covariance = covariance / count_not_nan
        covariance_inverse = np.linalg.inv(covariance)
        albedo = np.ones((rows, cols))
        for row_index in range(rows):
            for col_index in range(cols):
                if is_albedo:
                    albedo[row_index, col_index] = (
                        (data_array[:, row_index, col_index].T @␣
↪background_spectrum) /
                        (background_spectrum.T @ background_spectrum)
                    )
                up = (radiancediff_with_bg[:,row_index,col_index].T @␣
↪covariance_inverse @ target_spectrum)
                down = albedo[row_index, col_index] * (target_spectrum.T @␣
↪covariance_inverse @ target_spectrum)
                concentration[row_index, col_index] = up / down

        if is_iterate:
            l1filter = np.zeros((rows, cols))
            epsilon = np.finfo(np.float32).tiny
            iter_data = data_array.copy()

            for iter_num in range(5):
                if is_filter:
                    l1filter = 1 / (concentration + epsilon)
                iter_data = data_array - (
                    target_spectrum[:, None, None] * albedo[None, :, :] *␣
↪concentration[None, :, :]
                )
                background_spectrum = np.nanmean(iter_data, axis=(1,2))
                target_spectrum = np.multiply(background_spectrum,␣
↪stacked_unit_absorption_spectrum[0,:])
                radiancediff_with_bg = data_array - background_spectrum[:,␣
↪None, None]
                covariance = np.zeros((bands, bands))
                for i in range(rows):
                    for j in range(cols):
                        covariance += np.outer(radiancediff_with_bg[:, i, j],␣
↪radiancediff_with_bg[:, i, j])
                covariance = covariance / count_not_nan
                covariance_inverse = np.linalg.inv(covariance)
```

```
                for row_index in range(rows):
                    for col_index in range(cols):
                        up = (radiancediff_with_bg[:, row_index, col_index].T @␣
↪covariance_inverse @ target_spectrum)
                        down = albedo[row_index, col_index] * (target_spectrum.
↪T @ covariance_inverse @ target_spectrum)
                        concentration[row_index, col_index] = np.maximum(up /␣
↪down, 0)

    #
    return concentration
```

1.3 lognornal

1.3.1

[ ]:
```
# convert the radiance into log space
def lognormal_matched_filter(data_cube: np.array, unit_absorption_spectrum: np.
↪array):
    #
    bands, rows, cols = data_cube.shape
    #    concentration
    concentration = np.zeros((rows, cols))
    #
    log_background_spectrum = np.nanmean(np.log(data_cube), axis=(1,2))
    background_spectrum = np.exp(log_background_spectrum)

    #
    radiancediff_with_bg = np.log(data_cube) - log_background_spectrum[:,␣
↪None,None]
    d_covariance = np.log(data_cube)-background_spectrum[:,None,None]
    covariance = np.zeros((bands, bands))
    for row in range(rows):
        for col in range(cols):
            covariance += np.outer(d_covariance[:, row, col], d_covariance[:,␣
↪row, col])
    covariance = covariance/(rows*cols)
    covariance_inverse = np.linalg.inv(covariance)

    for row in range(rows):
        for col in range(cols):
            #
            numerator = (radiancediff_with_bg[:,row,col].T @ covariance_inverse␣
↪@ unit_absorption_spectrum)
            denominator = (unit_absorption_spectrum.T @ covariance_inverse @␣
↪unit_absorption_spectrum)
            concentration[row,col] = numerator/denominator
```

```
    return concentration
```

1.3.2 column-wise

[ ]:
```
# convert the radiance into log space
def columnwise_lognormal_matched_filter(data_cube: np.array,␣
 ↪unit_absorption_spectrum: np.array):
    #
    bands, rows, cols = data_cube.shape
    #    concentration
    concentration = np.zeros((rows, cols))
    #
    background_spectrum = np.nanmean(data_cube, axis=(1,2))
    target_spectrum = background_spectrum*unit_absorption_spectrum

    #
    radiancediff_with_bg = data_cube - background_spectrum[:, None,None]
    covariance = np.zeros((bands, bands))
    for row in range(rows):
        for col in range(cols):
            covariance += np.outer(radiancediff_with_bg[:, row, col],␣
 ↪radiancediff_with_bg[:, row, col])
    covariance = covariance/(rows*cols)
    covariance_inverse = np.linalg.inv(covariance)

    for row in range(rows):
        for col in range(cols):
            #
            numerator = (radiancediff_with_bg[:,row,col].T @ covariance_inverse␣
 ↪@ target_spectrum)
            denominator = (target_spectrum.T @ covariance_inverse @␣
 ↪target_spectrum)
            concentration[row,col] = numerator/denominator
    return concentration
```

1.4 Kalman filter

1.4.1

[ ]:
```
def Kalman_filterr_matched_filter(data_cube: np.array, unit_absorption_spectrum:
 ↪ np.array, albedoadjust, iterate, sparsity):

    return None
```

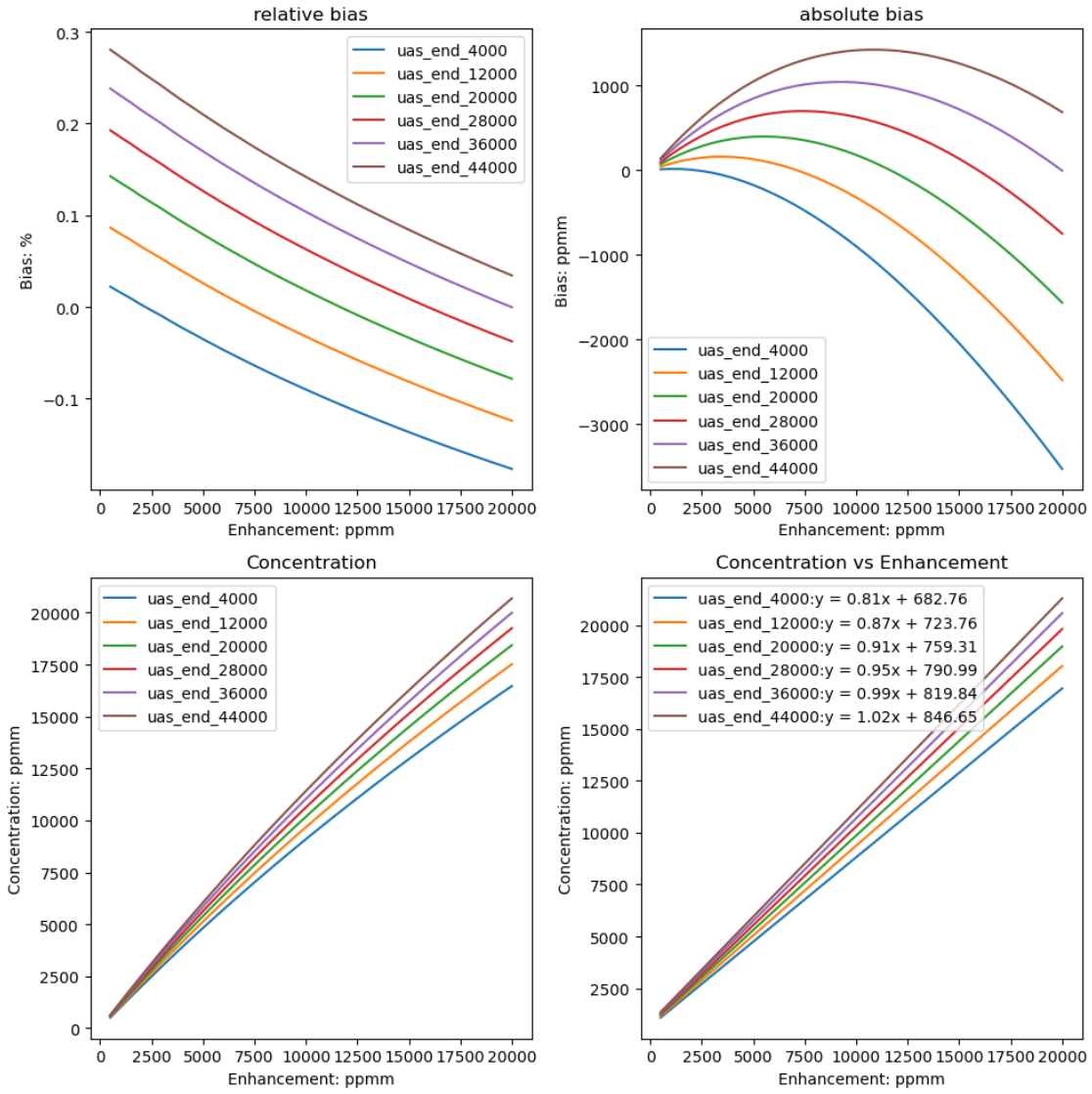
2

2.1

2.1.1

[ ]:
```
def spectrumlevel_test1():
    """



    """

    fig, ax = plt.subplots(2,2,figsize=(10,10))
    ax1,ax2,ax3,ax4 = ax.flatten()
    for end in range(4000,45000,8000):
        #      UAS
        ahsi_unit_absorption_spectrum_path = f"C:
↪\\Users\\RS\\VSCode\\matchedfiltermethod\\MyData\\uas\\AHSI_UAS_end_{end}.
↪txt"
        _,uas = nf.
↪open_unit_absorption_spectrum(ahsi_unit_absorption_spectrum_path,2150,2500)

        #
        base = None
        channels_path=r"C:
↪\\Users\\RS\\VSCode\\matchedfiltermethod\\MyData\\AHSI_channels.npz"
        base_filepath = f"C:\\PcModWin5\\Bin\\batch\\AHSI_Methane_0_ppmm_tape7.
↪txt"
        _,base = nf.
↪get_simulated_satellite_radiance(base_filepath,channels_path,2150,2500)
        concentration = 0
        concentrationlist = []
        re_biaslists = []
        ab_biaslists = []
        #
        enhancements = np.arange(500,20500,500)
        for enhancement in enhancements:
            #
            filepath = f"C:
↪\\PcModWin5\\Bin\\batch\\AHSI_Methane_{int(enhancement)}_ppmm_tape7.txt"
            _,radiance = nf.
↪get_simulated_satellite_radiance(filepath,channels_path,2150,2500)
            #
            concentration = profile_matched_filter(base,radiance,uas)
            ab_biaslists.append(concentration-enhancement)
            re_biaslists.append(((concentration-enhancement)/enhancement))
```

```
            concentrationlist.append(concentration)

        ax1.plot(enhancements,re_biaslists,label=f"uas_end_{end}")
        ax2.plot(enhancements,ab_biaslists,label=f"uas_end_{end}")
        ax3.plot(enhancements,concentrationlist,label=f"uas_end_{end}")
        polyfit_plot(enhancements,concentrationlist,ax4,f"uas_end_{end}")

    set_plot_details(ax1,"relative bias","Enhancement: ppmm","Bias: %")
    set_plot_details(ax2,"absolute bias","Enhancement: ppmm","Bias: ppmm")
    set_plot_details(ax3,"Concentration","Enhancement: ppmm","Concentration:␣
 ↪ppmm")
    set_plot_details(ax4,"Concentration vs Enhancement","Enhancement:␣
 ↪ppmm","Concentration: ppmm")

    plt.tight_layout()
    plt.show()

    return re_biaslists,ab_biaslists

re_biaslists,ab_biaslists = spectrumlevel_test1()
```

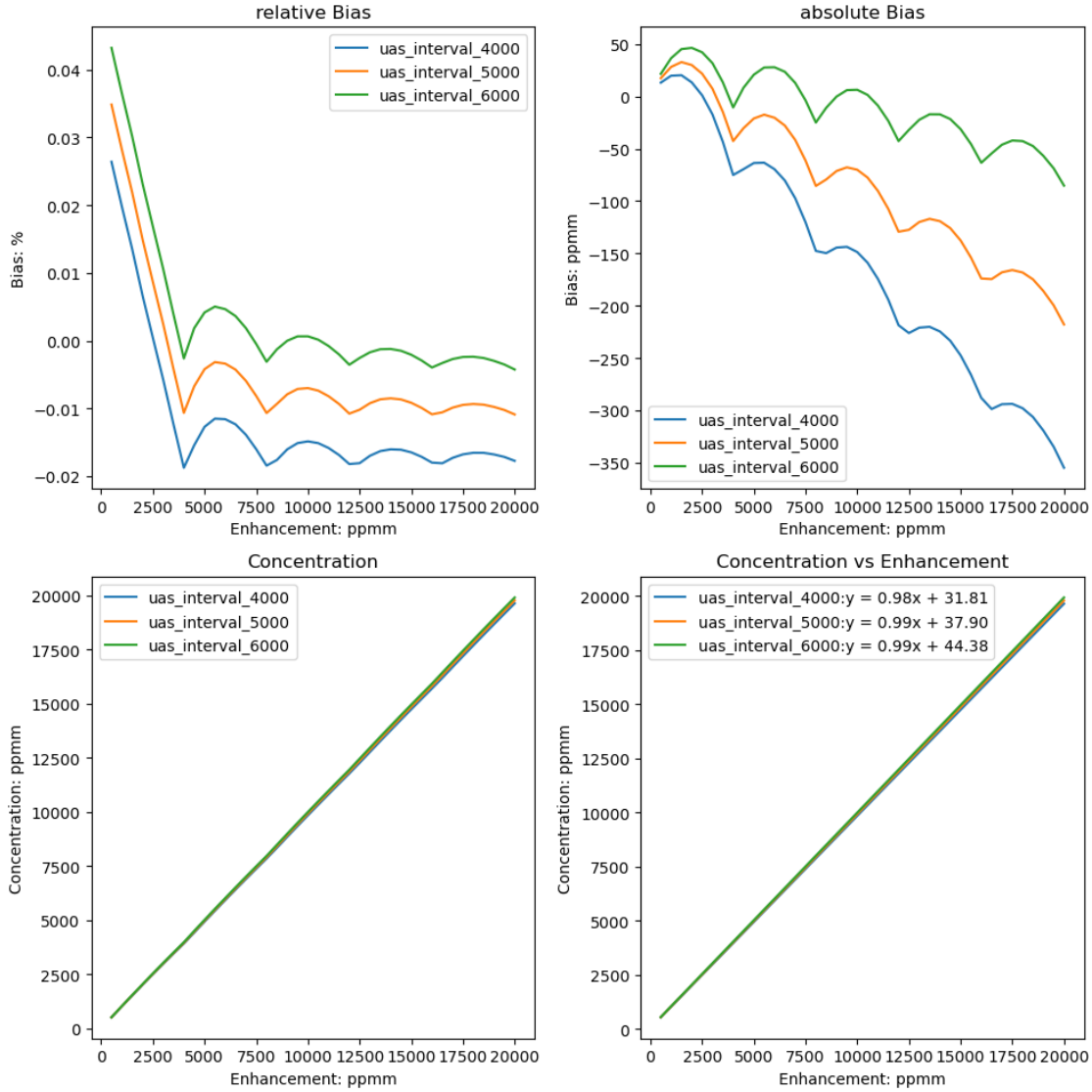


2.1.2

[ ]:
```
def spectrumlevel_test2():
    """
                ,

    """
    #
    fig, ax = plt.subplots(2,2,figsize=(10,10))
    ax1,ax2,ax3,ax4 = ax[0,0],ax[0,1],ax[1,0],ax[1,1]
    for interval in range(4000,7000,1000):
        #       ,
        uaslist = []
```

```
        uasrange = np.arange(0,46000,4000)
        for i in uasrange:
            # ahsi_unit_absorption_spectrum_path = f"C:
↪\\Users\\RS\\VSCode\\matchedfiltermethod\\MyData\\uas\\AHSI_UAS_end_{i}.txt"
            # _,uas = nf.
↪open_unit_absorption_spectrum(ahsi_unit_absorption_spectrum_path,2150,2500)
            _,uas = gu.generate_range_uas_AHSI(i,i+interval,2150,2500)
            uaslist.append(uas)

        #
        channels_path=r"C:
↪\\Users\\RS\\VSCode\\matchedfiltermethod\\MyData\\AHSI_channels.npz"
        base_filepath = f"C:\\PcModWin5\\Bin\\batch\\AHSI_Methane_0_ppmm_tape7.
↪txt"
        _,base_radiance = nf.
↪get_simulated_satellite_radiance(base_filepath,channels_path,2150,2500)

        ml_concentrationlist = []
        ml_biaslists = []
        ml_difflists = []

        #
        enhancements = np.arange(500,20500,500)
        for enhancement in enhancements:
            #
            filepath = f"C:
↪\\PcModWin5\\Bin\\batch\\AHSI_Methane_{enhancement}_ppmm_tape7.txt"
            _,radiance = nf.
↪get_simulated_satellite_radiance(filepath,channels_path,2150,2500)

            concentration =␣
↪profile_matched_filter_ML(base_radiance,radiance,uaslist)
            ml_biaslists.append(((concentration-enhancement)/enhancement))
            ml_difflists.append(concentration-enhancement)
            ml_concentrationlist.append(concentration)

        ax1.plot(enhancements,ml_biaslists,label=f"uas_interval_{interval}")
        ax2.plot(enhancements,ml_difflists,label=f"uas_interval_{interval}")
        ax3.
↪plot(enhancements,ml_concentrationlist,label=f"uas_interval_{interval}")
      ␣
↪polyfit_plot(enhancements,ml_concentrationlist,ax4,f"uas_interval_{interval}")

    set_plot_details(ax1, "relative Bias", "Enhancement: ppmm", "Bias: %")
    set_plot_details(ax2, "absolute Bias", "Enhancement: ppmm", "Bias: ppmm")
```

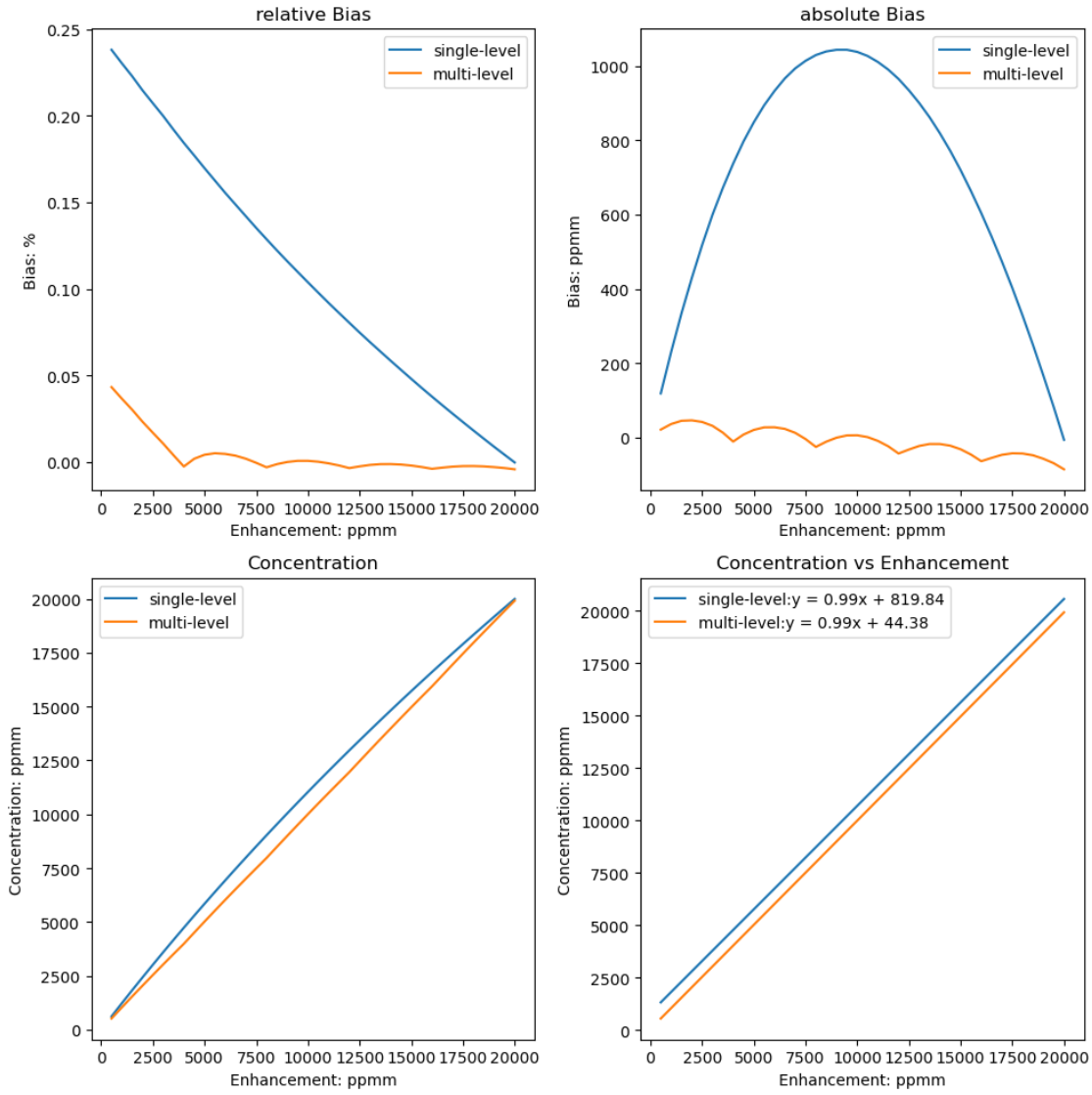
```
    set_plot_details(ax3, "Concentration", "Enhancement: ppmm", "Concentration:␣
↪ppmm")
    set_plot_details(ax4, "Concentration vs Enhancement", "Enhancement: ppmm",␣
↪"Concentration: ppmm")

    plt.tight_layout()
    plt.show()

    return ml_biaslists,ml_difflists

ml_biaslists,ml_difflists = spectrumlevel_test2()
```



2.1.3

```
[ ]: def spectrumlevel_test3():
        """
                    ,

        """
        #
        ahsi_unit_absorption_spectrum_path = f"C:
    ↪\\Users\\RS\\VSCode\\matchedfiltermethod\\MyData\\uas\\AHSI_UAS_end_36000.
    ↪txt"
        _,base_uas = nf.
    ↪open_unit_absorption_spectrum(ahsi_unit_absorption_spectrum_path,2150,2500)

        #       ,
        uaslist = []
        uasrange = np.arange(0,46000,4000)
        for i in uasrange:
            # ahsi_unit_absorption_spectrum_path = f"C:
    ↪\\Users\\RS\\VSCode\\matchedfiltermethod\\MyData\\uas\\AHSI_UAS_end_{i}.txt"
            # _,uas = nf.
    ↪open_unit_absorption_spectrum(ahsi_unit_absorption_spectrum_path,2150,2500)
            _,uas = gu.generate_range_uas_AHSI(i,i+6000,2150,2500)
            uaslist.append(uas)

        #
        channels_path=r"C:
    ↪\\Users\\RS\\VSCode\\matchedfiltermethod\\MyData\\AHSI_channels.npz"
        base_filepath = f"C:\\PcModWin5\\Bin\\batch\\AHSI_Methane_0_ppmm_tape7.txt"
        _,base_radiance = nf.
    ↪get_simulated_satellite_radiance(base_filepath,channels_path,2150,2500)

        # initiatate variables
        sg_concentrationlist = []
        sg_biaslists = []
        sg_difflists = []

        ml_concentrationlist = []
        ml_biaslists = []
        ml_difflists = []

        #
        enhancements = np.arange(500,20500,500)
        for enhancement in enhancements:
            #
```

```
        filepath = f"C:
↪\\PcModWin5\\Bin\\batch\\AHSI_Methane_{enhancement}_ppmm_tape7.txt"
        _,radiance = nf.
↪get_simulated_satellite_radiance(filepath,channels_path,2150,2500)

        #
        concentration = profile_matched_filter(base_radiance,radiance,base_uas)
        sg_biaslists.append(((concentration-enhancement)/enhancement))
        sg_difflists.append(concentration-enhancement)
        sg_concentrationlist.append(concentration)

        concentration =␣
↪profile_matched_filter_ML(base_radiance,radiance,uaslist)
        ml_biaslists.append(((concentration-enhancement)/enhancement))
        ml_difflists.append(concentration-enhancement)
        ml_concentrationlist.append(concentration)

    #
    fig, ax = plt.subplots(2,2,figsize=(10,10))
    ax1,ax2,ax3,ax4 = ax[0,0],ax[0,1],ax[1,0],ax[1,1]

    ax1.plot(enhancements,sg_biaslists,label="single-level")
    ax1.plot(enhancements,ml_biaslists,label="multi-level")
    ax2.plot(enhancements,sg_difflists,label="single-level")
    ax2.plot(enhancements,ml_difflists,label="multi-level")
    ax3.plot(enhancements,sg_concentrationlist,label="single-level")
    ax3.plot(enhancements,ml_concentrationlist,label="multi-level")
    polyfit_plot(enhancements,sg_concentrationlist,ax4,"single-level")
    polyfit_plot(enhancements,ml_concentrationlist,ax4,"multi-level")

    set_plot_details(ax1, "relative Bias", "Enhancement: ppmm", "Bias: %")
    set_plot_details(ax2, "absolute Bias", "Enhancement: ppmm", "Bias: ppmm")
    set_plot_details(ax3, "Concentration", "Enhancement: ppmm", "Concentration:␣
↪ppmm")
    set_plot_details(ax4, "Concentration vs Enhancement", "Enhancement: ppmm",␣
↪"Concentration: ppmm")

    plt.tight_layout()
    plt.show()

    return sg_biaslists,ml_biaslists

ml_biaslists,sg_biaslists = spectrumlevel_test3()
```



2.2 _1( )

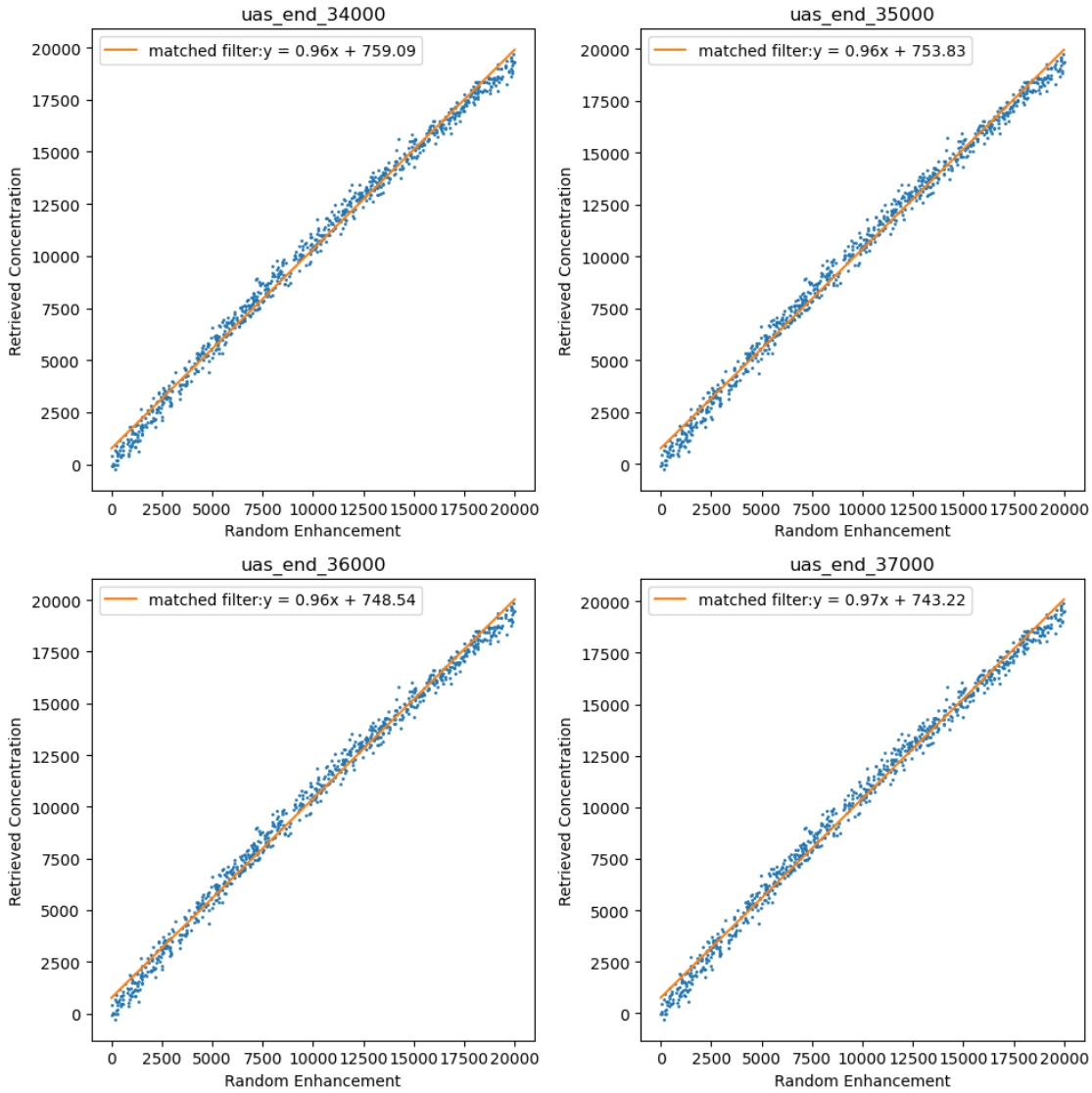
uniformly random 2%

2.2.1

```
[ ]: def imagelevel_test2_1():
         """
            2%
                          bias
         """
         fig, ax = plt.subplots(2, 2, figsize=(10, 10))
         axf = ax.flatten()
```

```
    i = 0

    #    uas
    for end in range(34000, 38000, 1000):
        ax1 = axf[i]

        #   AHSI
        _, uas = gu.generate_range_uas_AHSI(0, end, 2150, 2500)

        #
        basefilepath = f"C:\\PcModWin5\\Bin\\batch\\AHSI_Methane_0_ppmm_tape7.
↪txt"
        channels_path = r"C:
↪\\Users\\RS\\VSCode\\matchedfiltermethod\\MyData\\AHSI_channels.npz"
        _, base_radiance = nf.get_simulated_satellite_radiance(basefilepath,␣
↪channels_path, 2150, 2500)

        resultlist = []
        enhancements = np.arange(500, 20000, 500)


        #
        simulated_image, enhanced_mask, unenhanced_mask,␣
↪random_enhancement_values = enhancement_2perc()

        #
        result = matched_filter_with_fixed_bg(base_radiance, simulated_image,␣
↪uas)

        #
        enhanced = result[enhanced_mask]
        unenhanced = result[unenhanced_mask]

        #
        resultlist.append(np.mean(enhanced))

        #    x       y
        ax1.plot(random_enhancement_values, enhanced, marker='o', markersize=1,␣
↪linestyle='None')

        #
        polyfit_plot(random_enhancement_values, enhanced, ax1, "matched filter")
        set_plot_details(ax1, f"uas_end_{end}", "Random Enhancement",␣
↪"Retrieved Concentration")

        i += 1
```

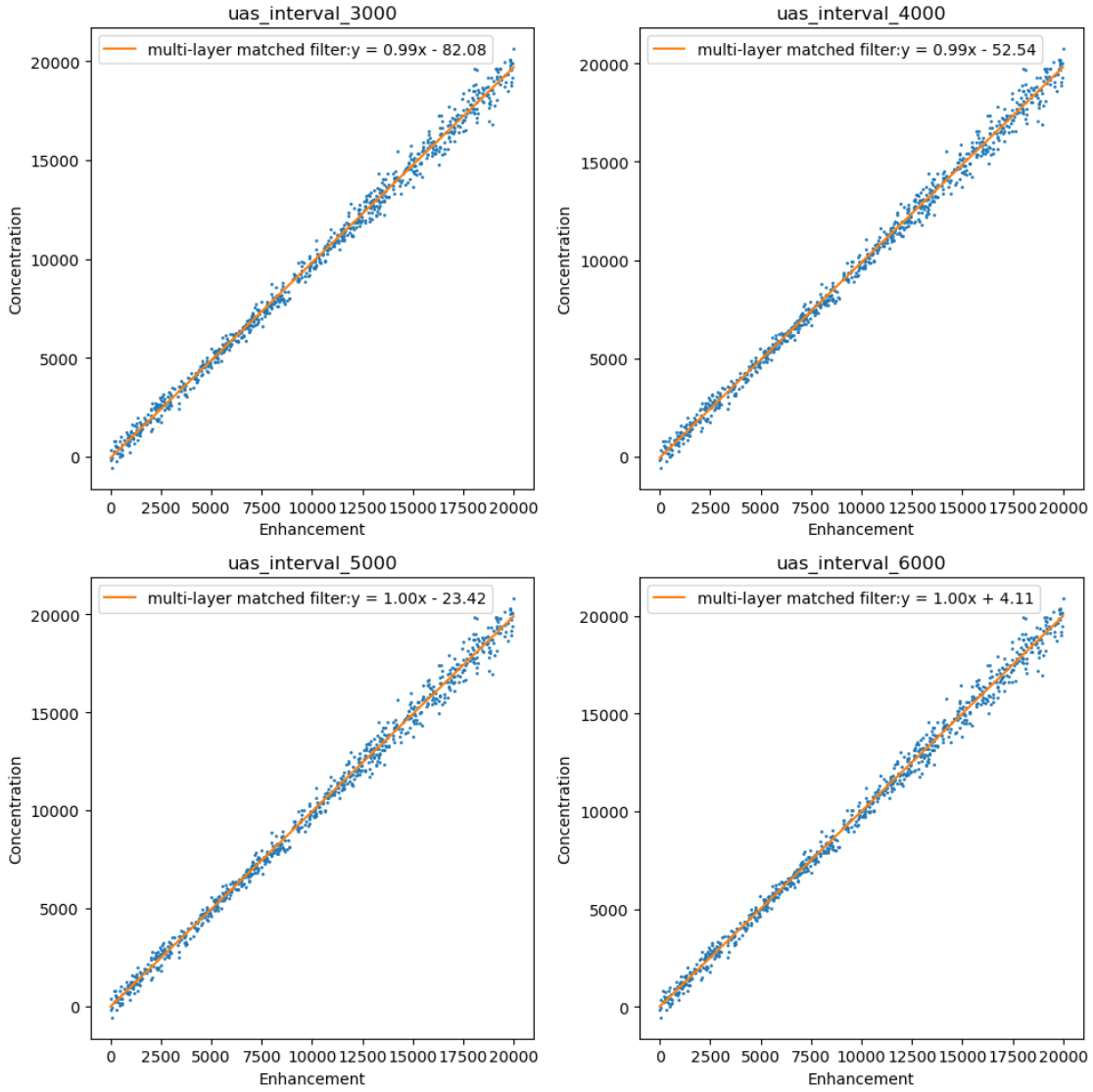
```
    #
    plt.tight_layout()
    plt.show()

    return None

imagelevel_test2_1()
```



2.2.2

```
[ ]: def imagelevel_test2_2():
        """
           2%
                    ,     bias
        """
        #
        basefilepath = f"C:\\PcModWin5\\Bin\\batch\\AHSI_Methane_0_ppmm_tape7.txt"
        channels_path = r"C:
    ↪\\Users\\RS\\VSCode\\matchedfiltermethod\\MyData\\AHSI_channels.npz"
        _,base_radiance = nf.
    ↪get_simulated_satellite_radiance(basefilepath,channels_path,2150,2500)

        fig,ax = plt.subplots(2,2,figsize=(10,10))
        axf = ax.flatten()
        axindex = 0
        for interval in range(3000,7000,1000):
            #   AHSI
            uaslist = []
            uasrange = np.arange(0,46000,4000)
            for i in uasrange:
                _,uas = gu.generate_range_uas_AHSI(i,i+interval,2150,2500)
                uaslist.append(uas)

            ax1 = axf[axindex]
            resultlist = []

            simulated_image,enhanced_mask,unenhanced_mask,␣
    ↪random_enhancement_values  = enhancement_2perc()
            #
            result = ML_matched_filter_with_fixed_bg(base_radiance,simulated_image,␣
    ↪uaslist)
            enhanced = result[enhanced_mask]
            unenhanced = result[unenhanced_mask]

            ax1.plot(random_enhancement_values, enhanced, marker='o', markersize =␣
    ↪1, linestyle='None')
            polyfit_plot(random_enhancement_values,enhanced,ax1,"multi-layer␣
    ↪matched filter")
            set_plot_details(ax1, f"uas_interval_{interval}", "Enhancement",␣
    ↪"Concentration")
            axindex += 1
        plt.tight_layout()
        plt.show()

        return resultlist
```

```
resultlist = imagelevel_test2_2()
```

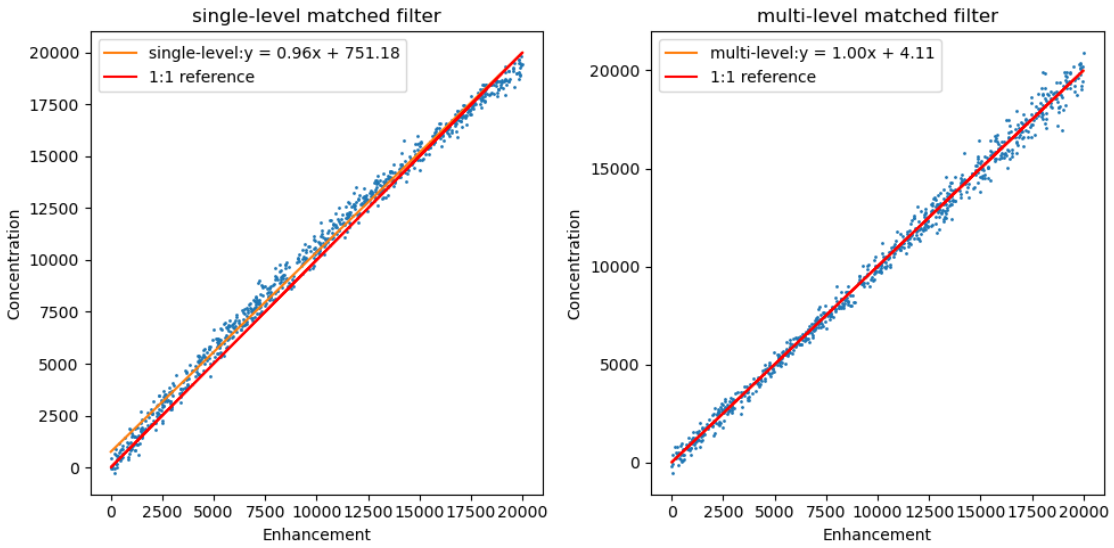


2.2.3

[ ]:
```
def imagelevel_test2_12():
    """
       2%
                 ,      bias
    """
    resultlist1 = []
    resultlist2 = []
```

```
    basefilepath = f"C:\\PcModWin5\\Bin\\batch\\AHSI_Methane_0_ppmm_tape7.txt"
    channels_path = r"C:
↪\\Users\\RS\\VSCode\\matchedfiltermethod\\MyData\\AHSI_channels.npz"
    _,base_radiance = nf.
↪get_simulated_satellite_radiance(basefilepath,channels_path,2150,2500)

    #   AHSI
    uas_filepath = r"C:
↪\\Users\\RS\\VSCode\\matchedfiltermethod\\MyData\\uas\\AHSI_UAS_end_36000.
↪txt"
    _,base_uas = nf.open_unit_absorption_spectrum(uas_filepath,2150,2500)

    #       ,
    uaslist = []
    uasrange = np.arange(0,46000,4000)
    for i in uasrange:
        _,uas = gu.generate_range_uas_AHSI(i,i+6000,2150,2500)
        uaslist.append(uas)


    fig,ax = plt.subplots(1,2,figsize=(10,5))
    simulated_image,enhanced_mask,unenhanced_mask,randon_enhancement_values =␣
↪enhancement_2perc()

    #
    result = matched_filter_with_fixed_bg(base_radiance,simulated_image,␣
↪base_uas)
    enhanced = result[enhanced_mask]
    unenhanced = result[unenhanced_mask]
    ax[0].plot(randon_enhancement_values, enhanced, marker='o', markersize = 1,␣
↪linestyle='None')
    polyfit_plot(randon_enhancement_values,enhanced,ax[0],"single-level")

    #
    result = ML_matched_filter_with_fixed_bg(base_radiance,simulated_image,␣
↪uaslist)
    enhanced = result[enhanced_mask]
    unenhanced = result[unenhanced_mask]

    ax[1].plot(randon_enhancement_values, enhanced, marker='o', markersize = 1,␣
↪linestyle='None')
    polyfit_plot(randon_enhancement_values,enhanced,ax[1],"multi-level")

    ax[0].plot(randon_enhancement_values,randon_enhancement_values,label="1:1␣
↪reference",color = "red")
```

```
    ax[1].plot(randon_enhancement_values,randon_enhancement_values,label="1:1␣
↪reference",color = "red")
    set_plot_details(ax[0], "single-level matched filter", "Enhancement",␣
↪"Concentration")
    set_plot_details(ax[1], "multi-level matched filter", "Enhancement",␣
↪"Concentration")
    plt.tight_layout()
    plt.show()
    return resultlist1,resultlist2

resultlist1,resultlist2 = imagelevel_test2_12()
```



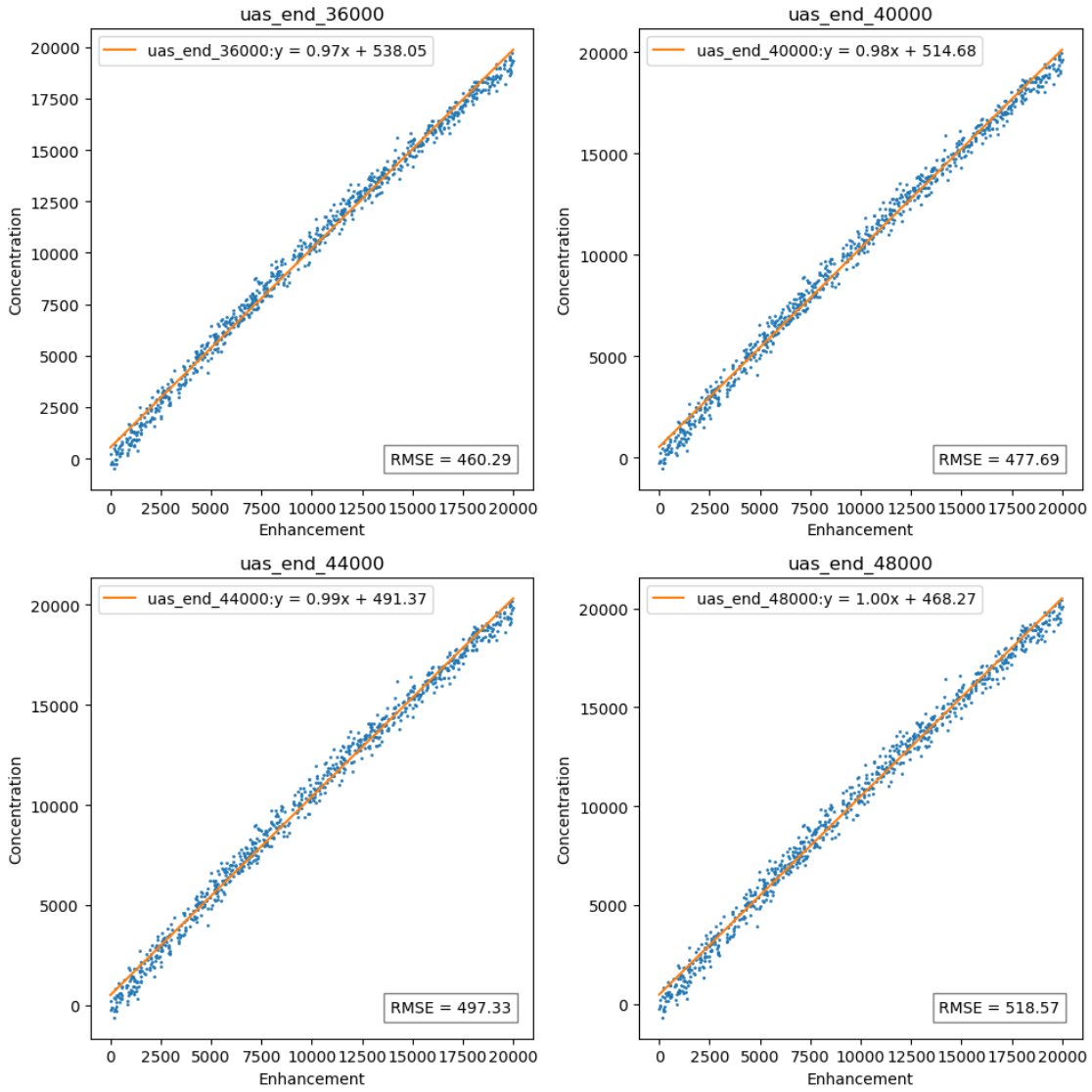
2.3 _2( )

2%

2.3.1

[ ]:
```
def imagelevel_test2_3():
    """
       2%
                ,     bias

    """
    #   AHSI
    fig,axes = plt.subplots(2,2,figsize=(10,10))
    axf = axes.flatten()
    axindex = 0
```

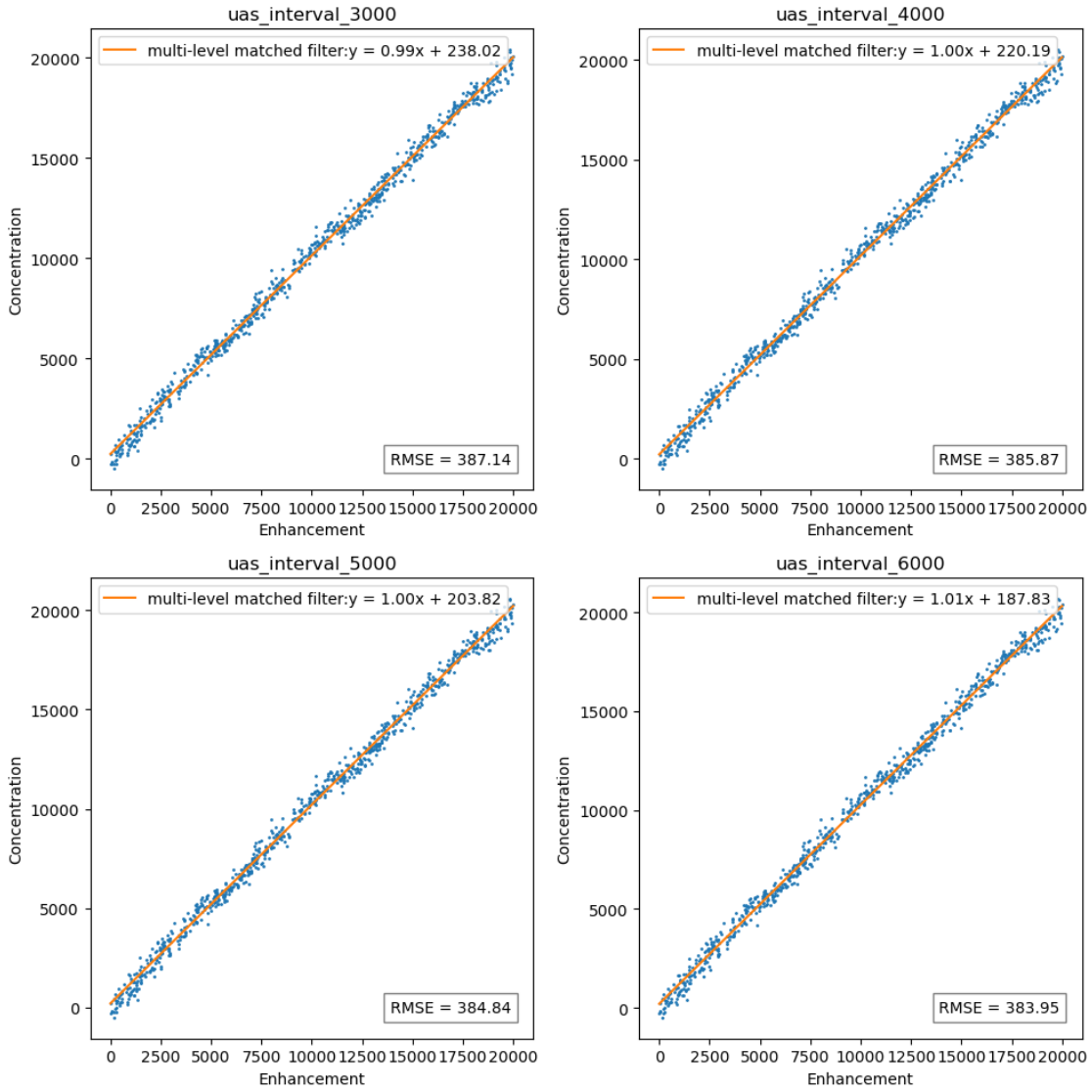
```
    for end in range(36000,50000,4000):
        ax = axf[axindex]
        #
        _,uas = gu.generate_range_uas_AHSI(0,end,2150,2500)

        simulated_image,enhanced_mask,unenhanced_mask,random_enhancement_values␣
↪= enhancement_2perc()
        result = mf.matched_filter(simulated_image, uas, iterate = False,␣
↪albedoadjust=False, sparsity= False)
        enhanced = result[enhanced_mask]
        unenhanced = result[unenhanced_mask]
        ax.plot(random_enhancement_values, enhanced, marker='o', markersize =␣
↪1, linestyle='None')
        polyfit_plot(random_enhancement_values,enhanced,ax,f"uas_end_{end}")

        #   RMSE
        slope, intercept = np.polyfit(random_enhancement_values, enhanced, 1)
        predicted = slope * random_enhancement_values + intercept
        rmse = np.sqrt(mean_squared_error(enhanced, predicted))

        #    RMSE
        ax.text(0.95, 0.05, f'RMSE = {rmse:.2f}', transform=ax.transAxes,
        fontsize=10, verticalalignment='bottom', horizontalalignment='right',
        bbox=dict(facecolor='white', alpha=0.5))
        set_plot_details(ax, f"uas_end_{end}", "Enhancement", "Concentration")
        axindex += 1

    plt.tight_layout()
    plt.show()

    return resultlist

resultlist = imagelevel_test2_3()
```



2.3.2

[ ]:
```
def imagelevel_test2_4():
    """
       2%
                ,      bias
    """
    fig,axes = plt.subplots(2,2,figsize=(10,10))
    axf = axes.flatten()
    axindex = 0
    for interval in range(3000,7000,1000):
        ax = axf[axindex]
        #
```

```
        uaslist = []
        _,uas = gu.generate_range_uas_AHSI(0,36000,2150,2500)
        uaslist.append(uas)
        uasrange = np.arange(6000,46000,6000)
        for i in uasrange:
            _,uas = gu.generate_range_uas_AHSI(i,i+interval,2150,2500)
            uaslist.append(uas)


        simulated_image,enhanced_mask,unenhanced_mask,␣
↪random_enhancement_values = enhancement_2perc()
        _,result = ML_matched_filter(simulated_image, uaslist,False)

        enhanced = result[enhanced_mask]
        unenhanced = result[unenhanced_mask]
        resultlist.append(np.mean(enhanced))
        ax.plot(random_enhancement_values, enhanced, marker='o', markersize =␣
↪1, linestyle='None')
        polyfit_plot(random_enhancement_values,enhanced,ax,"multi-level matched␣
↪filter")

        #   RMSE
        slope, intercept = np.polyfit(random_enhancement_values, enhanced, 1)
        predicted = slope * random_enhancement_values + intercept
        rmse = np.sqrt(mean_squared_error(enhanced, predicted))

        #    RMSE
        ax.text(0.95, 0.05, f'RMSE = {rmse:.2f}', transform=ax.transAxes,
        fontsize=10, verticalalignment='bottom', horizontalalignment='right',
        bbox=dict(facecolor='white', alpha=0.5))

        set_plot_details(ax, f"uas_interval_{interval}", "Enhancement",␣
↪"Concentration")
        axindex += 1

    plt.tight_layout()
    plt.show()
    return resultlist

resultlist = imagelevel_test2_4()
```

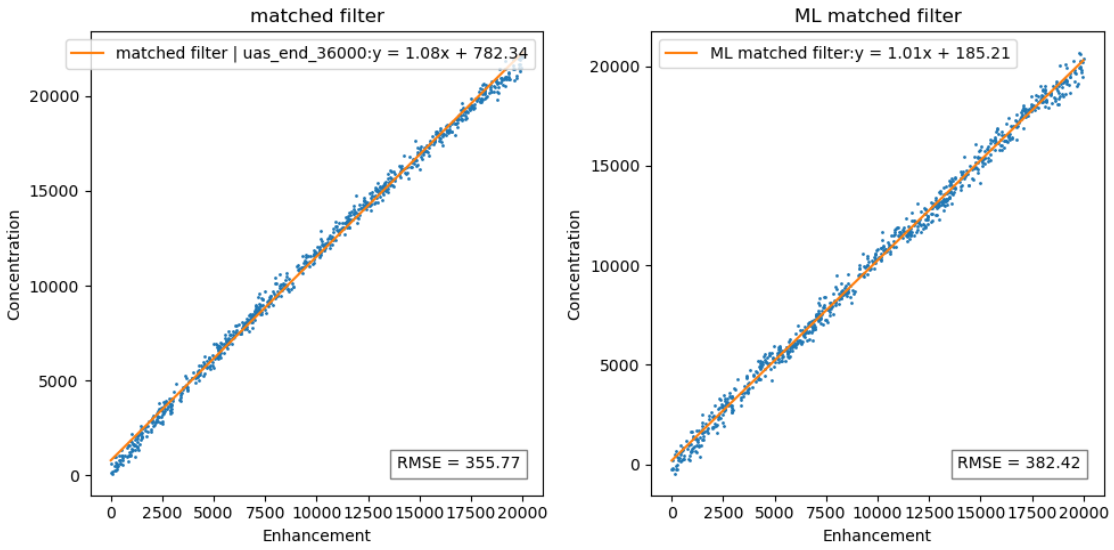


2.3.3

```
[ ]: def imagelevel_test2_34():
         """
            2%


         """
         resultlist = []  #
         resultlist2 = []  #
```

```
    #
    uas_path = r"C:
↪\\Users\\RS\\VSCode\\matchedfiltermethod\\MyData\\uas\\AHSI_UAS_end_36000.
↪txt"
    _, general_uas = nf.open_unit_absorption_spectrum(uas_path, 2150, 2500)

    #       ,
    uaslist = []
    uaslist.append(general_uas)
    uasrange = np.arange(6000, 46000, 6000)
    for i in uasrange:
        _, uas = gu.generate_range_uas_AHSI(i, i + 6000, 2150, 2500)
        uaslist.append(uas)

    #
    fig, ax = plt.subplots(1, 2, figsize=(10, 5))
    ax1, ax2 = ax[0], ax[1]

    #
    simulated_image, enhanced_mask, unenhanced_mask, random_enhancement_values␣
↪= enhancement_2perc()

    #
    result = matched_filter(simulated_image, general_uas, True, True, False)
    enhanced = result[enhanced_mask]
    resultlist.append(np.mean(enhanced))
    ax1.plot(random_enhancement_values, enhanced, marker='o', markersize=1,␣
↪linestyle='None')
    polyfit_plot(random_enhancement_values, enhanced, ax1, "matched filter |␣
↪uas_end_36000")

    #   RMSE
    slope, intercept = np.polyfit(random_enhancement_values, enhanced, 1)
    predicted = slope * random_enhancement_values + intercept
    rmse = np.sqrt(mean_squared_error(enhanced, predicted))

    #   ax1    RMSE
    ax1.text(0.95, 0.05, f'RMSE = {rmse:.2f}', transform=ax1.transAxes,
             fontsize=10, verticalalignment='bottom',␣
↪horizontalalignment='right',
             bbox=dict(facecolor='white', alpha=0.5))


    #
    _, result = ML_matched_filter(simulated_image, uaslist, False)
    enhanced = result[enhanced_mask]
    resultlist2.append(np.mean(enhanced))
```

```
    ax2.plot(random_enhancement_values, enhanced, marker='o', markersize=1,␣
↪linestyle='None')
    polyfit_plot(random_enhancement_values, enhanced, ax2, "ML matched filter")

    #   RMSE
    slope, intercept = np.polyfit(random_enhancement_values, enhanced, 1)
    predicted = slope * random_enhancement_values + intercept
    rmse = np.sqrt(mean_squared_error(enhanced, predicted))

    #   ax2    RMSE
    ax2.text(0.95, 0.05, f'RMSE = {rmse:.2f}', transform=ax2.transAxes,
             fontsize=10, verticalalignment='bottom',␣
↪horizontalalignment='right',
             bbox=dict(facecolor='white', alpha=0.5))

    set_plot_details(ax1, "matched filter", "Enhancement", "Concentration")
    set_plot_details(ax2, "ML matched filter", "Enhancement", "Concentration")

    plt.tight_layout()
    plt.show()

    return resultlist, resultlist2

#
resultlist, resultlist2 = imagelevel_test2_34()
```



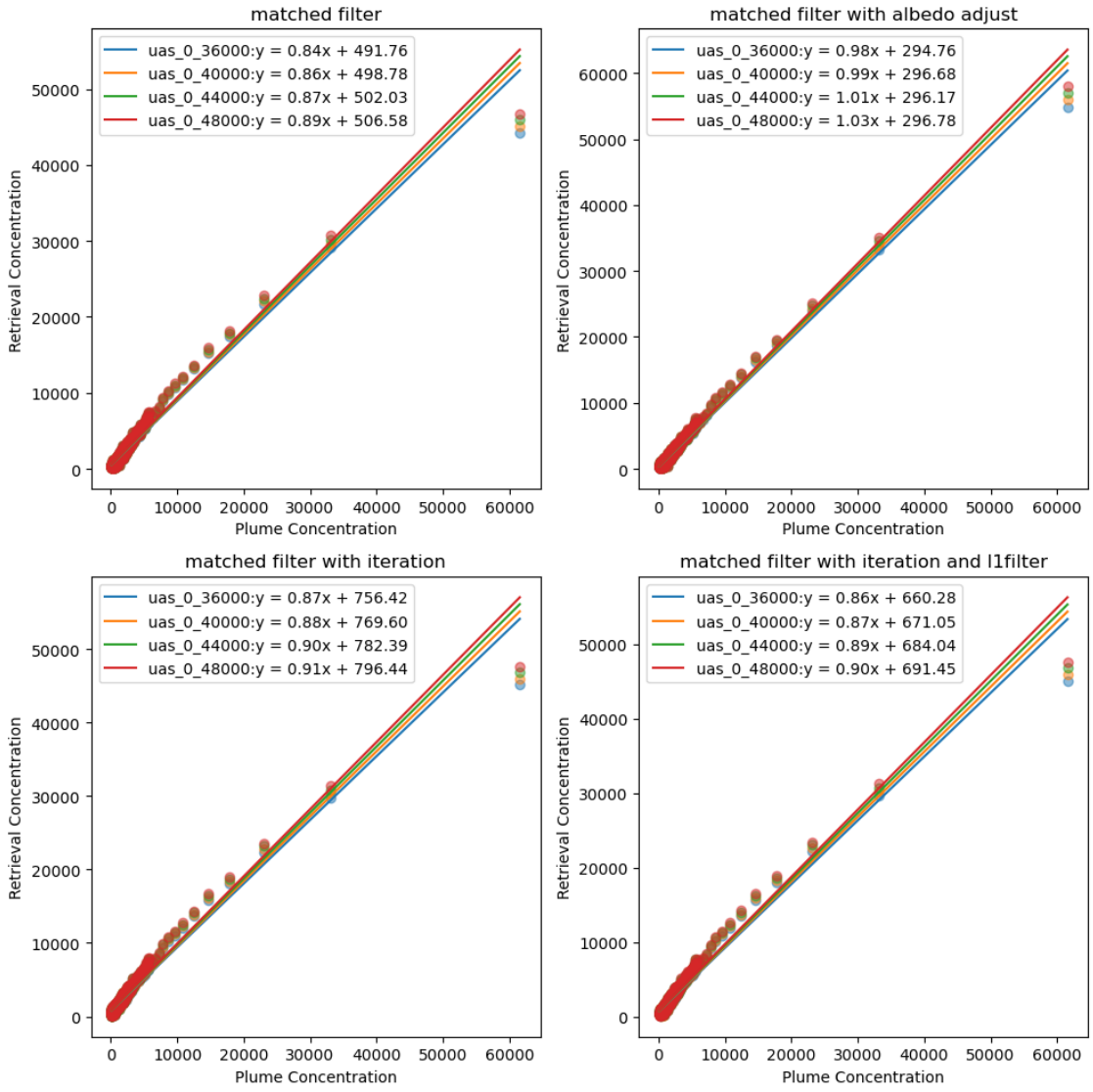
2.4 _3( )

2.4.1

[ ]:
```
def imagelevel_test3_1():
    """

    """
    #
    plume_path = f"C:
↪\\Users\\RS\\VSCode\\matchedfiltermethod\\MyData\\plumes\\gaussianplume_1000_2_stability_D.
↪npy"
    plume = np.load(plume_path)
    simulated_image = image_simulation(plume, 2150, 2500, 100, 100, 0.01)

    fig, axes = plt.subplots(2,2,figsize=(10,10))
    ax1,ax2,ax3,ax4 = axes.flatten()

    sub_function(plume,simulated_image,ax1,False,False,False)
    sub_function(plume,simulated_image,ax2,True,False,False)
    sub_function(plume,simulated_image,ax3,False,True,False)
    sub_function(plume,simulated_image,ax4,False,True,True)
    set_plot_details(ax1, "matched filter", "Plume pixel Concentration",␣
↪"Retrieval Concentration")
    set_plot_details(ax2, "matched filter with albedo adjust", "Plume pixel␣
↪Concentration", "Retrieval Concentration")
    set_plot_details(ax3, "matched filter with iteration", "Plume pixel␣
↪Concentration", "Retrieval Concentration")
    set_plot_details(ax4, "matched filter with iteration and l1filter", "Plume␣
↪pixel Concentration", "Retrieval Concentration")

    plt.tight_layout()
    plt.show()

    return None

def sub_function(plume,simulated_image,ax,*args):
    for end in range(36000,50000,4000):
            #
            _,uas = gu.generate_range_uas_AHSI(0,end,2150,2500)
            enhancement = matched_filter(simulated_image, uas,*args)

            plume_mask = plume > 100
            result_mask = enhancement > 100
            total_mask = plume_mask*result_mask

            #
            ax.scatter(plume[total_mask],enhancement[total_mask], alpha=0.5)
```

```
            polyfit_plot(plume[total_mask].flatten(),enhancement[total_mask].
 ↪flatten(), ax, f"uas_0_{end}")

            # #
            # molar_mass_CH4 = 16.04  #      g/mol
            # molar_volume_STP = 0.0224  #    m^3/mol at STP
            # emission = np.sum(plume[total_mask])*900*(molar_mass_CH4/
 ↪molar_volume_STP) * 1e-6
            # retrieval_emission = np.
 ↪sum(enhancement[total_mask])*900*(molar_mass_CH4/molar_volume_STP) * 1e-6

imagelevel_test3_1()
```

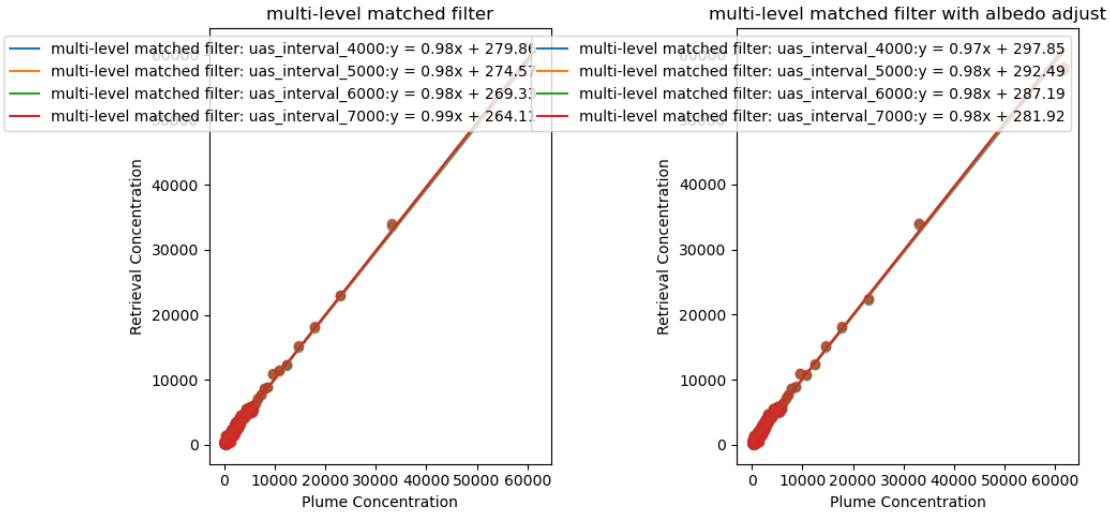
```
C:\Users\RS\AppData\Local\Temp\ipykernel_36536\3663452450.py:81: RuntimeWarning:
overflow encountered in scalar divide
  concentration[row,col] = np.maximum(numerator / denominator, 0.0)
```



2.4.2

```
[ ]: def imagelevel_test3_2():
         """

         """
         #
         plume_path = f"C:
      ↪\\Users\\RS\\VSCode\\matchedfiltermethod\\MyData\\plumes\\gaussianplume_1000_2_stability_D.
      ↪npy"
         plume = np.load(plume_path)
         simulated_image = image_simulation(plume, 2150, 2500, 100, 100, 0.01)
```

```
    #
    fig, ax = plt.subplots(1,2,figsize=(10, 5))
    ax1,ax2 = ax.flatten()
    for interval in range(4000,8000,1000):
        uaslist = []
        _,uas = gu.generate_range_uas_AHSI(0,36000,2150,2500)
        uaslist.append(uas)
        uasrange = np.arange(6000,46000,6000)
        for i in uasrange:
            _,uas = gu.generate_range_uas_AHSI(i,i+interval,2150,2500)
            uaslist.append(uas)

        sub_function_2(plume,simulated_image,uaslist,ax1,interval,False)
        sub_function_2(plume,simulated_image,uaslist,ax2,interval,True)
        set_plot_details(ax1, "multi-level matched filter", "Plume␣
↪Concentration", "Retrieval Concentration")
        set_plot_details(ax2, "multi-level matched filter with albedo adjust",␣
↪"Plume Concentration", "Retrieval Concentration")
    plt.tight_layout()
    plt.show()

    return None


def sub_function_2(plume,simulated_image,uaslist,ax,interval,*args):
    #
    _,enhancement = ML_matched_filter(simulated_image, uaslist,*args)
    plume_mask = plume > 100
    result_mask = enhancement > 100
    total_mask = plume_mask*result_mask

    #
    ax.scatter(plume[total_mask],enhancement[total_mask], alpha=0.5)
    polyfit_plot(plume[total_mask].flatten(),enhancement[total_mask].flatten(),␣
↪ax, f"multi-level matched filter: uas_interval_{interval}")

    #
    molar_mass_CH4 = 16.04  #     g/mol
    molar_volume_STP = 0.0224  #    m^3/mol at STP
    emission = np.sum(plume[total_mask])*900*(molar_mass_CH4/molar_volume_STP)␣
↪* 1e-6
    retrieval_emission = np.sum(enhancement[total_mask])*900*(molar_mass_CH4/
↪molar_volume_STP) * 1e-6


imagelevel_test3_2()
```



2.4.3

[ ]:
```
def imagelevel_test3_3():
    """

    """
    #
    plume_path = f"C:
↪\\Users\\RS\\VSCode\\matchedfiltermethod\\MyData\\plumes\\gaussianplume_1000_2_stability_D.
↪npy"
    plume = np.load(plume_path)
    simulated_image = image_simulation(plume, 2150, 2500, 100, 100, 0.01)
    _,general_uas = gu.generate_range_uas_AHSI(0,44000,2150,2500)

    #
    uaslist = []
    _,uas = gu.generate_range_uas_AHSI(0,36000,2150,2500)
    uaslist.append(uas)
    uasrange = np.arange(6000,46000,6000)
    for i in uasrange:
        _,uas = gu.generate_range_uas_AHSI(i,i+7000,2150,2500)
        uaslist.append(uas)

    fig, axes = plt.subplots(2,2,figsize=(10,10))
    ax1,ax2,ax3,ax4 = axes.flatten()

    enhancement= matched_filter(simulated_image, general_uas,True,False,False)
    plume_mask = plume > 100
    result_mask = enhancement > 100
```

```
    total_mask = plume_mask*result_mask

    ax1.scatter(plume[total_mask],enhancement[total_mask], alpha=0.5)
    ax1.plot(plume[total_mask],plume[total_mask],label="1:1 reference")
    polyfit_plot(plume[total_mask].flatten(),enhancement[total_mask].flatten(),␣
↪ax1, f"matched filter")

    _,enhancement2= ML_matched_filter(simulated_image, uaslist,False)
    plume_mask = plume > 100
    result_mask = enhancement2 > 100
    total_mask = plume_mask*result_mask

    ax2.scatter(plume[total_mask],enhancement2[total_mask], alpha=0.5)
    ax2.plot(plume[total_mask],plume[total_mask],label="1:1 reference")
    polyfit_plot(plume[total_mask].flatten(),enhancement2[total_mask].
↪flatten(), ax2, f"multi-level matched filter")

    set_plot_details(ax1, "matched filter", "Plume Concentration", "Retrieval␣
↪Concentration")
    set_plot_details(ax2, "multi-level matched filter", "Plume Concentration",␣
↪"Retrieval Concentration")

    #
    mse1 = mean_squared_error(plume[total_mask], enhancement[total_mask])
    mse2 = mean_squared_error(plume[total_mask], enhancement2[total_mask])

    #
    errors = [mse1, mse2]
    labels = ["Matched Filter", "Multi-level Matched Filter"]
    ax3.bar(labels, errors)
    ax3.set_title("MSE Comparison")
    ax3.set_ylabel("Mean Squared Error")

    #
    for i, v in enumerate(errors):
        ax3.text(i, v + 0.1 * max(errors), f"{v:.2f}", ha='center')

    # RMSE
    rmse1 = np.sqrt(mse1)
    rmse2 = np.sqrt(mse2)
    rmses = [rmse1, rmse2]

    ax4.plot(labels, rmses, marker='o')
    ax4.set_title("RMSE Comparison")
    ax4.set_ylabel("Root Mean Squared Error")

    plt.tight_layout()
```

```
    plt.show()

    return None

imagelevel_test3_3()
```