La analysis

September 13, 2017

```
In [1]: import matplotlib.pyplot as plt # for general plotting
        import matplotlib.cm as cm # for colorscales in 2D maps. Shown in https://r
        import h5py
        import numpy as np
        import datetime
        from mpl_toolkits.axes_grid1 import make_axes_locatable

In [2]: def print_map_info(mapnr, element, xpixsize, ypixsize, xrange, yrange, mapm
            """Print size information of XRF map"""
            print("Map nr: " + str(mapnr) + ", element " + element + ":")
            print("  Map min: " + str(mapmin) + " ug/cm2")
            print("  Map max: " + str(mapmax) + " ug/cm2")
            print("  Pixel size Y: " + str(xpixsize) + " um")
            print("  Pixel size Y: " + str(ypixsize) + " um")
            print("  Map size Y: " + str(xrange) + " um")
            print("  Map size Y: " + str(yrange) + " um")

        def print_MAPS_H5_file_content(f):
            """Print groups of H5 files created by XRF measurements by MAPS"""
            for dataset in f: # Loop through all elements in the h5 file
                print("Groups in file:  " + dataset)
            g = f['/MAPS'] # This is the group with the relevant data in the h5 fi
            channel_names = f['/MAPS/channel_names']
            print("Content of '/MAPS': ")
            for dataset in g: # Loop through all elements in g
                h = f['/MAPS/' + dataset]
                print(h)
            for element in channel_names:
                print(element)

            return 0

        def scaletoflux(fluxmeas):
            """Select to which flux measurement shall be scaled to"""
            if fluxmeas == 'ds_ic': # scale data to ds_ic
                return 0
            elif fluxmeas == 'us_ic': # scale data to us_ic
```

```
            return 1
        elif fluxmeas == 'SRcurrent': # scale data to SRcurrent
            return 2
        else: # No scale indicator --> don't scale
            return -1

    def e2i(channel_names, element):
        """Convert element to the index for a given measurement"""
        j = 0
        for i in channel_names:
            if i.decode('utf-8') == element:  # Need to decode the binary stri
                return j
            j += 1
        print("Element not found in 'channel_names', sorry for that!")
        return -1

    def isnan(num):
        """Return true if the argument is NaN, otherwise return false"""
        return num != num

In [3]: datapath = "C:/MSdata/postdoc/studies/beamtimes/2017-03 APS 2-ID-D/fitimgLa
        savepath  = "C:/MSdata/postdoc/studies/2017/2017-03 La material/map plottin

In [6]: def plotmap(fullpath, savename, element, fluxnorm, fitnorm):
            """Plot a H5 files created by XRF measurements by MAPS"""

            """Important settings"""
            savepng = True
            savepdf = True
            savetxt = True
            plotting= False
            respng = 500 # Resolution for png images in dpi
            respdf = 500 # Resolution for png images in dpi

            """Load file and relevant h5 groups"""
            f = h5py.File(fullpath, 'r')
            # f = h5py.File(r"C:\MSdata\postdoc\studies\beamtimes\2017-03 APS 2-ID-
            # print_MAPS_H5_file_content(f)
            channel_names = f['/MAPS/channel_names']  # Names of fitted channels su
            scaler_names = f['/MAPS/scaler_names']  # Names of scalers such as 'SRc
            scalers = f['/MAPS/scalers']  # Scaler values for [scaler, x, y]
            XRF_fits = f['/MAPS/XRF_fits']  # Quantified channel [channel, x, y]
            XRF_fits_quant = f['/MAPS/XRF_fits_quant']  # Number of cts per ug/cm2
            XRF_roi = f['/MAPS/XRF_roi']  # Quantified channel [channel, x, y]
            XRF_roi_quant = f['/MAPS/XRF_roi_quant']  # Number of cts per ug/cm2 [
            x_axis = f['/MAPS/x_axis']  # x position of pixels  [position in um]
            y_axis = f['/MAPS/y_axis']  # y position of pixels  [position in um]
```

```python
"""Select to which flux measurement shall be scaled to"""
fluxnormmatrix = scalers[1, :, :];  fluxnormmatrix[:, :] = 1  # Matrix
if fluxnorm == 'ds_ic':  # scale data to ds_ic
    fluxnormindex = 0; fluxnormmatrix = scalers[fluxnormindex, :, :]
elif fluxnorm == 'us_ic':  # scale data to us_ic
    fluxnormindex = 1; fluxnormmatrix = scalers[fluxnormindex, :, :]
elif fluxmeas == 'SRcurrent':  # scale data to SRcurrent
    fluxnormindex = 2; fluxnormmatrix = scalers[fluxnormindex, :, :]

"""Select XRF element of interest"""
elementindex = e2i(channel_names, element)

"""Select which fitting to be used for quantification"""
if fitnorm == 'roi':  # Normalization with fitted data --> is default i
    fitnormvalue = XRF_roi_quant[fluxnormindex, 0, elementindex]
    rawmatrix = XRF_roi[elementindex, :, :]
elif fitnorm == 'fit':  # Normalization with ROI fitted --> To be used
    fitnormvalue = XRF_fits_quant[fluxnormindex, 0, elementindex]
    rawmatrix = XRF_fits[elementindex, :, :]

"""Calculate quantified element matrix in ug/cm2"""
m = rawmatrix / fluxnormmatrix / fitnormvalue
# # t = ds / us  # Transmittance: ds_ic normalized to us_ic

"""Preparation for proper map scaling"""
xrange = max(x_axis) - min(x_axis)
xpixsize = xrange / len(x_axis)
yrange = max(y_axis) - min(y_axis)
ypixsize = yrange / len(y_axis)
"""Remove column with nan"""
m = m[:, 0:len(x_axis)-1]
yrange = yrange - ypixsize
x = np.arange(-xrange / 2 + xpixsize / 2, xrange / 2, xpixsize)
y = np.arange(-yrange / 2 + ypixsize / 2, yrange / 2, ypixsize)
 # print_map_info(mapnr, element, xpixsize, ypixsize, xrange, yrange, m
matrixm = np.asmatrix(m) # needed to evaluate min & max

"""Replace NaN in center of map (if part of dead line) with average val
for i in range(0, len(m), 1):
    for j in range(0, len(m[0]), 1):
        if(isnan(m[i, j])):
            print("NaN @ i: " + str(i) + ", j: " + str(j))
            m[i, j] = (m[i-1, j] + m[i+1, j]) / 2

"""Plot raw map"""
xsize = 5 # Size of figure in inches
f, ax = plt.subplots(1, 1, figsize=(xsize, xsize * xrange / yrange))
ax1 = plt.subplot(1, 1, 1)
```

```
ax1.set_aspect(xrange / yrange)
plt.pcolor(y, x, m, cmap=cm.afmhot, vmin=0, vmax=matrixm.max())
plt.xlabel("X position  $(\mu m)$")
plt.ylabel("Y position  $(\mu m)$")
ax2 = plt.gca()
divider = make_axes_locatable(ax2)
cax = divider.append_axes("right", size="5%", pad=0.05)
plt.colorbar(cax=cax)
cax.set_ylabel('Area density of La  $(\mu g/cm^2)$', rotation=90)
plt.tight_layout()
if plotting == True:
    plt.show()
if savetxt == True:
    np.savetxt(savename + ".asc", m, fmt='%1.4e')
if savepdf == True:
    plt.savefig(savename + ".pdf", dpi=respdf)
if savepng == True:
    plt.savefig(savename + ".png", dpi=respng)
plt.close()

"""Plot masked map"""
nx = m.shape[0]  # Number of x values of the ROI
ny = m.shape[1]  # Number of x values of the ROI
# masknoise = np.zeros((nx, ny))
cutoff = 10
mnonoise = np.ma.masked_where(m <  cutoff, m)
f, ax = plt.subplots(1, 1, figsize=(xsize, xsize * xrange / yrange))
ax1 = plt.subplot(1, 1, 1)
ax1.set_aspect(xrange / yrange)
plt.pcolor(y, x, mnonoise, cmap=cm.afmhot, vmin=0, vmax=matrixm.max())
plt.xlabel("X position  $(\mu m)$")
plt.ylabel("Y position  $(\mu m)$")
ax2 = plt.gca()
divider = make_axes_locatable(ax2)
cax = divider.append_axes("right", size="5%", pad=0.05)
plt.colorbar(cax=cax)
cax.set_ylabel('Area density of La  $(\mu g/cm^2)$', rotation=90)
plt.tight_layout()
if plotting == True:
    plt.show()
if savetxt == True:
    np.savetxt(savename + "_nonoise.asc", np.ma.filled(mnonoise, 0), fm
if savepdf == True:
    plt.savefig(savename + "_nonoise.pdf", dpi=respdf)
if savepng == True:
    plt.savefig(savename + "_nonoise.png", dpi=respng)
plt.close()
```

```python
        """Plot mask"""
        maskbool = ~np.ma.getmask(mnonoise)
        maskint = maskbool.astype(int)
        f, ax = plt.subplots(1, 1, figsize=(xsize, xsize * xrange / yrange))
        ax1 = plt.subplot(1, 1, 1)
        ax1.set_aspect(xrange / yrange)
        plt.pcolor(y, x, maskint, cmap=cm.afmhot, vmin=0, vmax=1)
        plt.xlabel("X position  $(\mu m)$")
        plt.ylabel("Y position  $(\mu m)$")
        ax2 = plt.gca()
        divider = make_axes_locatable(ax2)
        cax = divider.append_axes("right", size="5%", pad=0.05)
        plt.colorbar(cax=cax)
        cax.set_ylabel('Black: No La, White: With La', rotation=90)
        plt.tight_layout()
        if plotting == True:
            plt.show()
        if savetxt == True:
            np.savetxt(savename + "_bool.asc", maskint, fmt='%1d')
        if savepdf == True:
            plt.savefig(savename + "_bool.pdf", dpi=respdf)
        if savepng == True:
            plt.savefig(savename + "_bool.png", dpi=respng)
        plt.close()

        return 0
```

```python
In [7]: description = {
        #     "175" : "s1_planview_HR_large",
        #     "089" : "s5_planview_LR",
        #     "091" : "s5_planview_HR_large",
        #     "092" : "s5_planview_HR_small",
        #     "160" : "s1_planview_LR",
        #     "175" : "s1_planview_HR_large",
        #     "176" : "s1_planview_HR",
        #     "185" : "s15_planview_LR",
        #     "186" : "s15_planview_HR_large",
        #     "187" : "s15_planview_HR_small",
        #     "196" : "s5c_planview_LR",
        #     "197" : "s5c_planview_HR_large",
        #     "198" : "s5c_planview_HR_small",
        #     "209" : "s2.5_planview_LR",
        #     "210" : "s2.5_planview_HR_large",
        #     "212" : "s2.5_planview_HR_small",
        #     "221" : "s10_planview_LR",
        #     "222" : "s10_planview_HR_large",
        #     "223" : "s10_planview_HR_small",
        #     "234" : "s15_crosssection_HR_a",
```

```
#      "238" : "s15_crosssection_HR_b",
#      "257" : "s5c_crosssection_HR_a",
#      "259" : "s5c_crosssection_HR_b",
#      "273" : "s5_crosssection_HR_a",
#      "277" : "s5_crosssection_HR_b",
#      "289" : "s1_crosssection_HR_a",
#      "291" : "s1_crosssection_HR_b",
#      "301" : "s10_crosssection_HR_a"
}

"""Loop over all maps from 'description'"""
for key in description:
    print("Analyzing map no: ", key, " which is: ", description[key])
    timestart = datetime.datetime.now()
    plotmap(datapath+"/2idd_0"+key+".h5", savepath+"/"+description[key]+"_'
    timediff = datetime.datetime.now() - timestart
    print("   Time needed: ", timediff)
```

```
Analyzing map no:  175  which is:  s1_planview_HR_large
   Time needed:  0:01:59.598032
Analyzing map no:  089  which is:  s5_planview_LR
NaN @ i: 221, j: 311
NaN @ i: 221, j: 312
NaN @ i: 221, j: 313
NaN @ i: 221, j: 314
NaN @ i: 221, j: 315
NaN @ i: 221, j: 316
NaN @ i: 221, j: 317
NaN @ i: 221, j: 318
NaN @ i: 221, j: 319
NaN @ i: 221, j: 320
NaN @ i: 221, j: 321
NaN @ i: 221, j: 322
NaN @ i: 221, j: 323
NaN @ i: 221, j: 324
NaN @ i: 221, j: 325
NaN @ i: 221, j: 326
NaN @ i: 221, j: 327
NaN @ i: 221, j: 328
NaN @ i: 221, j: 329
NaN @ i: 221, j: 330
NaN @ i: 221, j: 331
NaN @ i: 221, j: 332
NaN @ i: 221, j: 333
NaN @ i: 221, j: 334
NaN @ i: 221, j: 335
NaN @ i: 221, j: 336
NaN @ i: 221, j: 337
```

```
NaN @ i: 221, j: 338
NaN @ i: 221, j: 339
NaN @ i: 221, j: 340
NaN @ i: 221, j: 341
NaN @ i: 221, j: 342
NaN @ i: 221, j: 343
NaN @ i: 221, j: 344
NaN @ i: 221, j: 345
NaN @ i: 221, j: 346
NaN @ i: 221, j: 347
NaN @ i: 221, j: 348
NaN @ i: 221, j: 349
NaN @ i: 221, j: 350
NaN @ i: 221, j: 351
NaN @ i: 221, j: 352
NaN @ i: 221, j: 353
NaN @ i: 221, j: 354
NaN @ i: 221, j: 355
NaN @ i: 221, j: 356
NaN @ i: 221, j: 357
NaN @ i: 221, j: 358
NaN @ i: 221, j: 359
NaN @ i: 221, j: 360
NaN @ i: 221, j: 361
NaN @ i: 221, j: 362
NaN @ i: 221, j: 363
NaN @ i: 221, j: 364
NaN @ i: 221, j: 365
NaN @ i: 221, j: 366
NaN @ i: 221, j: 367
NaN @ i: 221, j: 368
NaN @ i: 221, j: 369
NaN @ i: 221, j: 370
NaN @ i: 221, j: 371
NaN @ i: 221, j: 372
NaN @ i: 221, j: 373
NaN @ i: 221, j: 374
NaN @ i: 221, j: 375
NaN @ i: 221, j: 376
NaN @ i: 221, j: 377
NaN @ i: 221, j: 378
NaN @ i: 221, j: 379
NaN @ i: 221, j: 380
NaN @ i: 221, j: 381
NaN @ i: 221, j: 382
NaN @ i: 221, j: 383
NaN @ i: 221, j: 384
NaN @ i: 221, j: 385
```

```
NaN @ i: 221, j: 386
NaN @ i: 221, j: 387
NaN @ i: 221, j: 388
NaN @ i: 221, j: 389
NaN @ i: 221, j: 390
NaN @ i: 221, j: 391
NaN @ i: 221, j: 392
NaN @ i: 221, j: 393
NaN @ i: 221, j: 394
NaN @ i: 221, j: 395
NaN @ i: 221, j: 396
NaN @ i: 221, j: 397
NaN @ i: 221, j: 398
NaN @ i: 221, j: 399
NaN @ i: 221, j: 400
NaN @ i: 221, j: 401
NaN @ i: 221, j: 402
NaN @ i: 221, j: 403
NaN @ i: 221, j: 404
NaN @ i: 221, j: 405
NaN @ i: 221, j: 406
NaN @ i: 221, j: 407
NaN @ i: 221, j: 408
NaN @ i: 221, j: 409
NaN @ i: 221, j: 410
NaN @ i: 221, j: 411
NaN @ i: 221, j: 412
NaN @ i: 221, j: 413
NaN @ i: 221, j: 414
NaN @ i: 221, j: 415
NaN @ i: 221, j: 416
NaN @ i: 221, j: 417
NaN @ i: 221, j: 418
NaN @ i: 221, j: 419
NaN @ i: 221, j: 420
NaN @ i: 221, j: 421
NaN @ i: 221, j: 422
NaN @ i: 221, j: 423
NaN @ i: 221, j: 424
NaN @ i: 221, j: 425
NaN @ i: 221, j: 426
NaN @ i: 221, j: 427
NaN @ i: 221, j: 428
NaN @ i: 221, j: 429
NaN @ i: 221, j: 430
NaN @ i: 221, j: 431
NaN @ i: 221, j: 432
NaN @ i: 221, j: 433
```

```
NaN @ i: 221, j: 434
NaN @ i: 221, j: 435
NaN @ i: 221, j: 436
NaN @ i: 221, j: 437
NaN @ i: 221, j: 438
NaN @ i: 221, j: 439
NaN @ i: 221, j: 440
NaN @ i: 221, j: 441
NaN @ i: 221, j: 442
NaN @ i: 221, j: 443
NaN @ i: 221, j: 444
NaN @ i: 221, j: 445
NaN @ i: 221, j: 446
NaN @ i: 221, j: 447
NaN @ i: 221, j: 448
NaN @ i: 221, j: 449
NaN @ i: 221, j: 450
NaN @ i: 221, j: 451
NaN @ i: 221, j: 452
NaN @ i: 221, j: 453
NaN @ i: 221, j: 454
NaN @ i: 221, j: 455
NaN @ i: 221, j: 456
NaN @ i: 221, j: 457
NaN @ i: 221, j: 458
NaN @ i: 221, j: 459
NaN @ i: 221, j: 460
NaN @ i: 221, j: 461
NaN @ i: 221, j: 462
NaN @ i: 221, j: 463
NaN @ i: 221, j: 464
NaN @ i: 221, j: 465
NaN @ i: 221, j: 466
NaN @ i: 221, j: 467
NaN @ i: 221, j: 468
NaN @ i: 221, j: 469
NaN @ i: 221, j: 470
NaN @ i: 221, j: 471
NaN @ i: 221, j: 472
NaN @ i: 221, j: 473
NaN @ i: 221, j: 474
NaN @ i: 221, j: 475
NaN @ i: 221, j: 476
NaN @ i: 221, j: 477
NaN @ i: 221, j: 478
NaN @ i: 221, j: 479
NaN @ i: 221, j: 480
NaN @ i: 221, j: 481
```

```
NaN @ i: 221, j: 482
NaN @ i: 221, j: 483
NaN @ i: 221, j: 484
NaN @ i: 221, j: 485
NaN @ i: 221, j: 486
NaN @ i: 221, j: 487
NaN @ i: 221, j: 488
NaN @ i: 221, j: 489
NaN @ i: 221, j: 490
NaN @ i: 221, j: 491
NaN @ i: 221, j: 492
NaN @ i: 221, j: 493
NaN @ i: 221, j: 494
NaN @ i: 221, j: 495
NaN @ i: 221, j: 496
NaN @ i: 221, j: 497
NaN @ i: 221, j: 498
NaN @ i: 221, j: 499
   Time needed:  0:02:54.984393
Analyzing map no:  091  which is:  s5_planview_HR_large
   Time needed:  0:02:51.370981
Analyzing map no:  092  which is:  s5_planview_HR_small
   Time needed:  0:02:56.499833
Analyzing map no:  160  which is:  s1_planview_LR
   Time needed:  0:01:54.632721
Analyzing map no:  176  which is:  s1_planview_HR
   Time needed:  0:01:49.175232
Analyzing map no:  185  which is:  s15_planview_LR
   Time needed:  0:02:20.989238
Analyzing map no:  186  which is:  s15_planview_HR_large
   Time needed:  0:02:45.258093
Analyzing map no:  187  which is:  s15_planview_HR_small
   Time needed:  0:02:14.211781
Analyzing map no:  196  which is:  s5c_planview_LR
   Time needed:  0:02:21.658193
Analyzing map no:  197  which is:  s5c_planview_HR_large
   Time needed:  0:02:55.783449
Analyzing map no:  198  which is:  s5c_planview_HR_small
   Time needed:  0:02:25.961606
Analyzing map no:  209  which is:  s2.5_planview_LR
   Time needed:  0:02:18.389280
Analyzing map no:  210  which is:  s2.5_planview_HR_large
   Time needed:  0:02:32.811123
Analyzing map no:  212  which is:  s2.5_planview_HR_small
   Time needed:  0:02:13.426518
Analyzing map no:  221  which is:  s10_planview_LR
   Time needed:  0:02:30.965478
Analyzing map no:  222  which is:  s10_planview_HR_large
```

```
   Time needed:  0:02:29.743281
Analyzing map no:  223  which is:  s10_planview_HR_small
   Time needed:  0:02:18.112226
Analyzing map no:  291  which is:  s1_crosssection_HR_b
   Time needed:  0:01:48.062535
Analyzing map no:  301  which is:  s10_crosssection_HR_a
   Time needed:  0:02:42.797960
```

```
In [ ]:
```