Spectra Analysis

September 28, 2021

1 Initial Input

1.1 Directory with the Object Locator Files

[1]:
```
#DIRECTORY = "/mnt/basestar/UNI/Masterarbeit/data/21-09-20/"
#DIRECTORY = "../../data/21-09-20/"
DIRECTORY = "../../data/reproduction/"
```

1.2 Analysis Parameters

[2]:
```
DENOIS_WEIGHT = 0
DECONV_ITERATIONS = 0
LDA_REF_OVERRIDE = 0

LDA_MIN = 350
LDA_MAX = 800
```

2 Environment Setup

2.1 Basic Utilities

[3]:
```
import sys
import os
```

2.2 Maths

[4]:
```
import numpy as np
pi = np.pi
```

2.3 Plotting

[5]:
```
import matplotlib.pyplot as plt
%matplotlib inline

import pickle
import json
```

```
from util import plotstyle, devices, objects, tdms,␣
 ↪calibration_persistent_data_path
from util.objects import Region
```

[36]:
```
plotstyle.load('print')
```

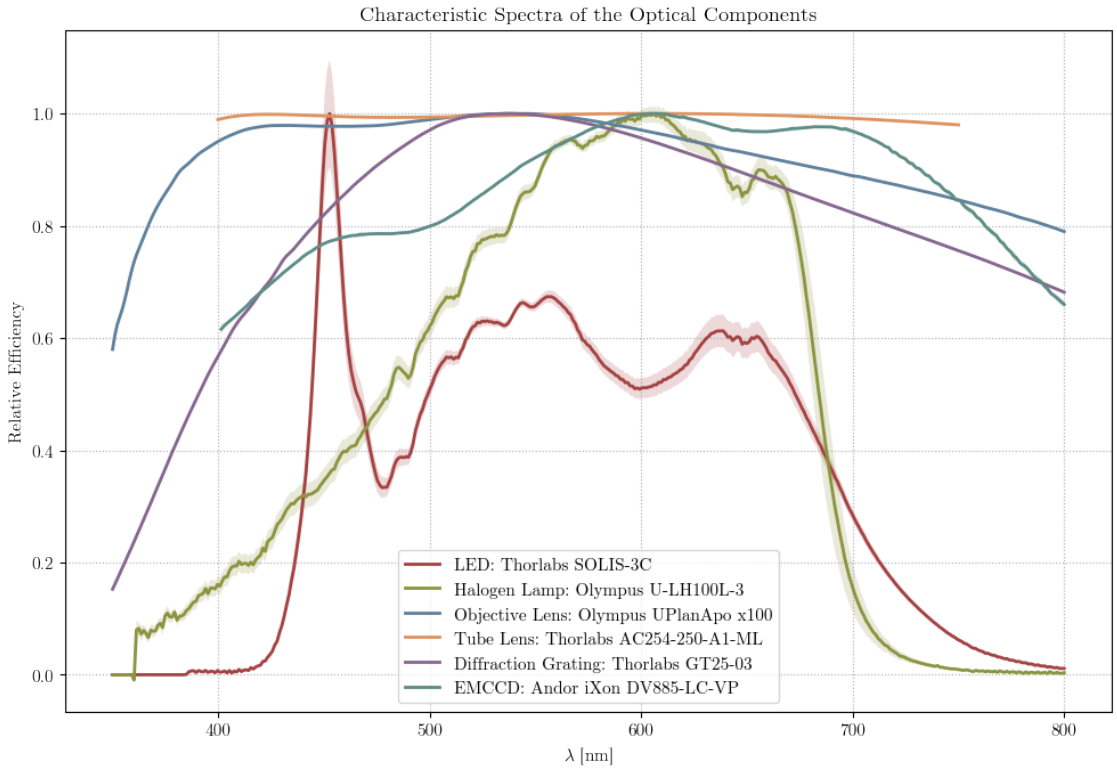
[36]:
```
True
```

3 List known Devices, show Correction Spectra

[7]:
```
DEVS = devices.load_all()
for did in DEVS:
    print(did + ": \t" + DEVS[did].descr_str())
```

```
led-specscope1:         LED: Thorlabs SOLIS-3C
halo-specscope1:        Halogen Lamp: Olympus U-LH100L-3
ol-manplt1:     Objective Lens: Olympus UPlanApo x100
tl-manplt1:     Tube Lens: Thorlabs AC254-250-A1-ML
grating-manplt1:        Diffraction Grating: Thorlabs GT25-03
emccd-manplt1:  EMCCD: Andor iXon DV885-LC-VP
```

[37]:
```
fig = plt.figure(figsize=(10,7), dpi=100)

axs = fig.add_gridspec(1, 1)

ax = fig.add_subplot(axs[0, 0])

for did in DEVS:
    dev = DEVS[did]
    LDA = np.linspace( np.maximum(dev.ldamin, LDA_MIN),
                       np.minimum(dev.ldamax, LDA_MAX),
                       400 )
    spec, err = dev.evaluate(LDA)
    ax.fill_between( LDA, spec-err, spec+err, alpha=plotstyle.err_alpha(),␣
 ↪)#color=line[1] )
    ax.plot( LDA, spec, lw=2, label=dev.descr_str() )#color=line[1] )

ax.set_title("Characteristic Spectra of the Optical Components")
ax.set_xlabel('$\lambda$ [nm]')
ax.set_ylabel('Relative Efficiency')

ax.legend()

ax.grid()

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



4 Read the Object Locator Files

[9]:
```
object_files = [ "{d}/{fn}".format(d=DIRECTORY, fn=f) for f in os.listdir(␣
↪DIRECTORY ) if f.endswith(".json") ]

object_files
```

[9]:
```
['../../data/reproduction//21-08-24-object0003.json',
 '../../data/reproduction//21-08-24-object0004.json',
 '../../data/reproduction//21-08-24-object0005.json',
 '../../data/reproduction//21-08-24-object0001+0002.json']
```

[10]:
```
object_descriptors = []

for fn in object_files:
    with open(fn, 'r') as file:
        object_descriptors.append( objects.Descriptor().from_dict( json.
↪loads(file.read()) ) )
```

4.1 Load Video Data

[11]:
```
OBJECTS = []
for d,i in zip( object_descriptors, range(len(object_descriptors)) ):
    OBJECTS.append( objects.Object() )
    OBJECTS[-1].descriptor = d
    OBJECTS[-1].video = tdms.VideoSeries().load( d.videos )
    OBJECTS[-1].index = i
```

[12]:
```
for o in OBJECTS:
    print( "{w}x{h} px,\t{f} Frames".format( w=o.video.width, h=o.video.height,␣
 ↪f=o.video.frames ) )
```

```
500x500 px,     256 Frames
1000x1000 px,   256 Frames
1000x1000 px,   256 Frames
500x500 px,     512 Frames
```

4.2 Apply the Calibration

[13]:
```
with open( "{d}/calibration.pickle".format( d=calibration_persistent_data_path␣
 ↪), 'rb' ) as importfile:
    calibration = pickle.loads( importfile.read() )
```

[14]:
```
for o in OBJECTS:
    o.descriptor.ldaref = 532.0
    o.descriptor.sref = calibration['intercept'] + (-1.0)*o.descriptor.
 ↪angle*calibration['slope']
    # TODO: divide sref by binning
    o.descriptor.sref /= o.video.binning
```

5 Setup Corrections

[15]:
```
all_devices = devices.load_all()

for o in OBJECTS:
    for dev_id in o.descriptor.devices:
        o.correction.add_device( all_devices[dev_id] )
```

6 ROI

[16]:
```
for o in OBJECTS:
    o.gen_roi( LDA_MAX, 1 )
    o.gen_streak_limit_idxs()
```

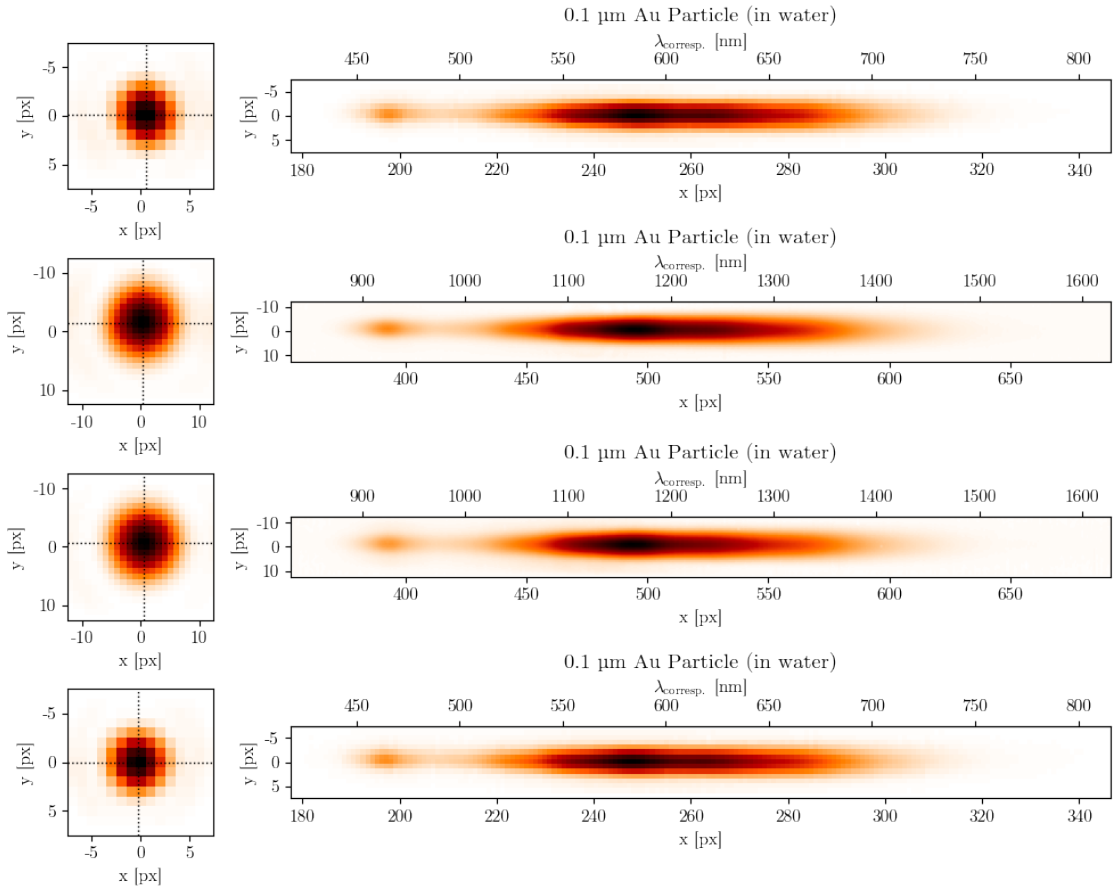
[17]:
```
#for o in OBJECTS:
#    print( o.LDA[o.streak_begin_idx] )
#    print( o.correction.ldamin )
#    print( o.streak_begin_idx )
```

[18]:
```
for o in OBJECTS:
    o.subtract_background()
```

[19]:
```
mcx = []
mcy = []

for o in OBJECTS:
    zo = o.region( Region.SPOT )
    zo = zo/np.sum(zo)

    x = np.arange( zo.shape[1] ) - (o.descriptor.roi_width-1)/2
    y = np.arange( zo.shape[2] ) - (o.descriptor.roi_width-1)/2

    X, Y = np.meshgrid( x, y )

    #print( zo.shape )
    #print( X.shape )
    #print( Y.shape )
    cx = np.zeros( zo.shape[0] )
    cy = np.zeros( zo.shape[0] )
    for F in range(zo.shape[0]):
        cx[F] = np.sum( zo[F]*X )
        cy[F] = np.sum( zo[F]*Y )

    mcx.append( np.sum(cx) )
    mcy.append( np.sum(cy) )
```

[38]:
```
fig = plt.figure(figsize=(10,2*len(OBJECTS)), dpi=100)

axs = fig.add_gridspec(len(OBJECTS), 5)

for o in OBJECTS:
    ax = fig.add_subplot(axs[ o.index , :1 ])

    ax.imshow(np.mean( o.region(Region.SPOT), axis=0 ),
              extent=o.extent(Region.SPOT),
              cmap=plotstyle.cmap('m'))

    ax.plot( [ mcx[o.index], mcx[o.index] ], [ o.extent(Region.SPOT)[2], o.
 ↪extent(Region.SPOT)[3] ],
             color=plotstyle.monochrome_fg(),
```

```
            ls=':', lw=1)
    ax.plot( [ o.extent(Region.SPOT)[0], o.extent(Region.SPOT)[1] ], [ mcy[o.
 ↪index], mcy[o.index] ],
            color=plotstyle.monochrome_fg(),
            ls=':', lw=1)

    ax.set_xlabel('x [px]')
    ax.set_ylabel('y [px]')
    #ax.set_title( o.descriptor.particle.descr_str() )



    ax = fig.add_subplot(axs[ o.index , 1: ])

    ax.imshow(np.mean( o.region(Region.STREAK), axis=0 ),
              extent=o.extent(Region.STREAK),
              cmap=plotstyle.cmap('m'))

    secxax = ax.secondary_xaxis(location='top',
                                functions=(lambda x: x*o.px_to_lda(),
                                           lambda x: x*o.lda_to_px()))
    secxax.set_xlabel('$\lambda_\mathrm{corresp.}$ [nm]')

    ax.set_xlabel('x [px]')
    ax.set_ylabel('y [px]')
    ax.set_title( o.descriptor.particle.descr_str() )

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



7 Line-wise extract/average Spectra

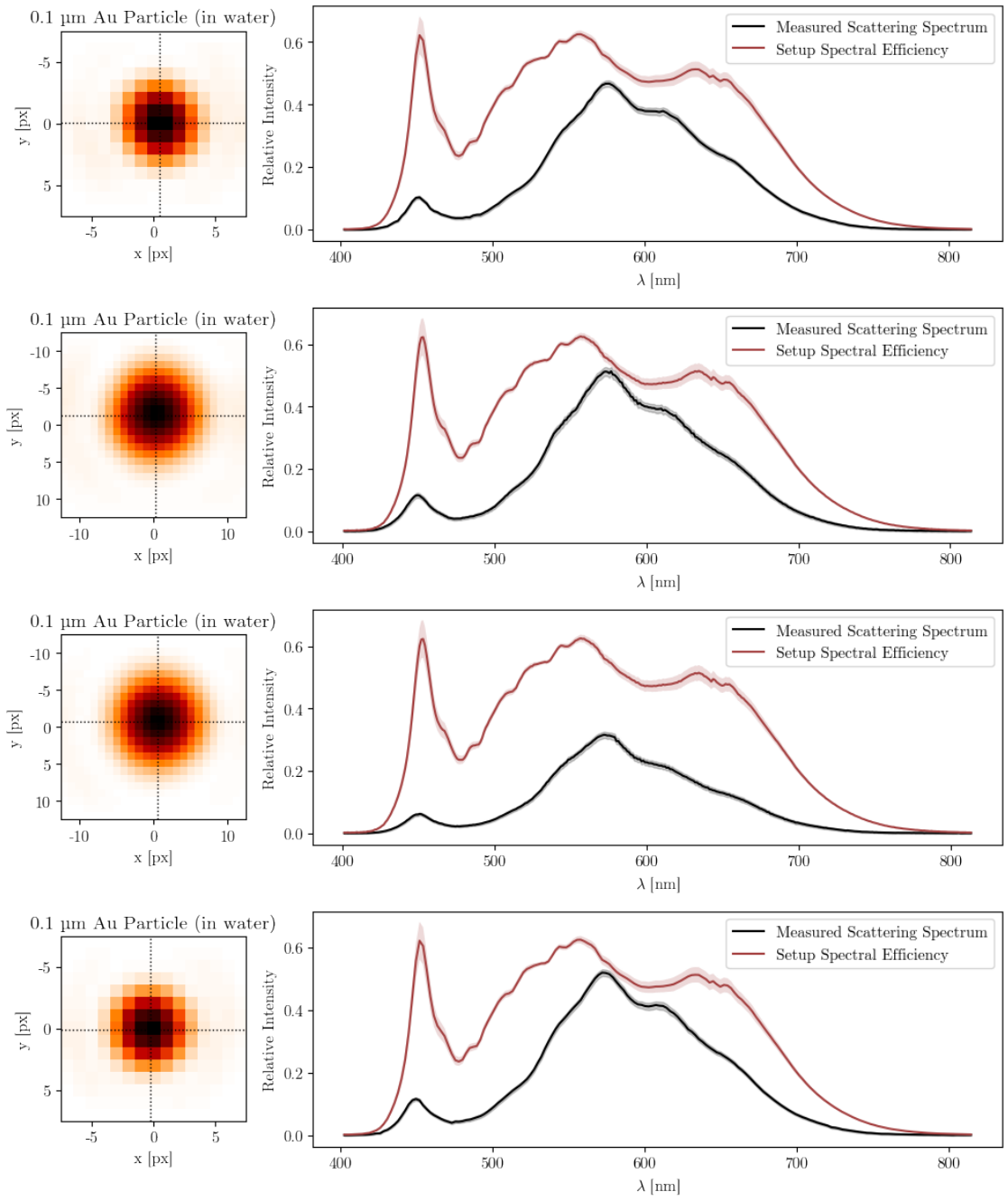
[21]:
```
LINE_WEIGHTS = [ np.empty_like( o.streak()[:,:,0] ) for o in OBJECTS ]

for o in OBJECTS:
    #print( LINE_WEIGHTS[o.index].shape )
    for T in range(o.roi().shape[0]):
        for Y in range(o.roi().shape[1]):
            LINE_WEIGHTS[o.index][T,Y] = np.mean( o.streak()[T,Y,:] ) - np.min(␣
 ↪o.streak()[T,Y,:] )
```

[22]:
```
FRAME_SPECS = [ np.empty_like(o.streak()[:,0,:]) for o in OBJECTS ]

for o in OBJECTS:
    for T in range( FRAME_SPECS[o.index].shape[0] ):
        FRAME_SPECS[o.index][T] = np.zeros( FRAME_SPECS[o.index][T].shape )
        Y = np.argmax( LINE_WEIGHTS[o.index][T,:] )
        FRAME_SPECS[o.index][T] += o.streak()[T,Y,:]
```

[23]:
```
AVG_SPECS = [ np.empty_like(o.streak()[0,0,:]) for o in OBJECTS ]
SPEC_ERRS = [ np.empty_like(o.streak()[0,0,:]) for o in OBJECTS ]

for o in OBJECTS:
    AVG_SPECS[o.index] = np.mean( FRAME_SPECS[o.index], axis=0 )

    SPEC_ERRS[o.index] = np.zeros( AVG_SPECS[o.index].shape )
    for T in range( FRAME_SPECS[o.index].shape[0] ):
        SPEC_ERRS[o.index] += np.square( FRAME_SPECS[o.index][T] - AVG_SPECS[o.
 ↪index] )
    SPEC_ERRS[o.index] /= FRAME_SPECS[o.index].shape[0]
    SPEC_ERRS[o.index] = np.sqrt( SPEC_ERRS[o.index] )
```

[39]:
```
fig = plt.figure(figsize=(10,3*len(OBJECTS)), dpi=100)

axs = fig.add_gridspec(len(OBJECTS),4)

for o in OBJECTS:
    ax = fig.add_subplot(axs[ o.index , :1 ])

    ax.imshow(np.mean( o.region(Region.SPOT), axis=0 ),
              extent=o.extent(Region.SPOT),
              cmap=plotstyle.cmap('m'))

    ax.plot( [ mcx[o.index], mcx[o.index] ], [ o.extent(Region.SPOT)[2], o.
 ↪extent(Region.SPOT)[3] ],
             color=plotstyle.monochrome_fg(),
             ls=':', lw=1)
    ax.plot( [ o.extent(Region.SPOT)[0], o.extent(Region.SPOT)[1] ], [ mcy[o.
 ↪index], mcy[o.index] ],
             color=plotstyle.monochrome_fg(),
             ls=':', lw=1)

    ax.set_xlabel('x [px]')
    ax.set_ylabel('y [px]')
    ax.set_title( o.descriptor.particle.descr_str() )



    ax = fig.add_subplot(axs[ o.index , 1: ])

    ax.fill_between( o.lda(Region.STREAK),
                     AVG_SPECS[o.index] - SPEC_ERRS[o.index],
                     AVG_SPECS[o.index] + SPEC_ERRS[o.index],
```

```
                    color=plotstyle.monochrome_fg(),
                    alpha=plotstyle.err_alpha() )
    ax.plot( o.lda(Region.STREAK),
             AVG_SPECS[o.index],
             color=plotstyle.monochrome_fg(),
             label="Measured Scattering Spectrum" )

    #CORR_LDA = np.clip( o.lda(Region.STREAK), o.correction.ldamin, o.
↪correction.ldamax )
    CORR_LDA = o.lda(Region.STREAK)
    corr = o.correction.evaluate( CORR_LDA )
    ax.fill_between( CORR_LDA,
                    corr[0] - corr[1],
                    corr[0] + corr[1],
                    alpha=plotstyle.err_alpha() )
    ax.plot( CORR_LDA,
             corr[0],
             label="Setup Spectral Efficiency" )


    ax.legend()

    ax.set_xlabel('$\lambda$ [nm]')
    ax.set_ylabel('Relative Intensity')



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

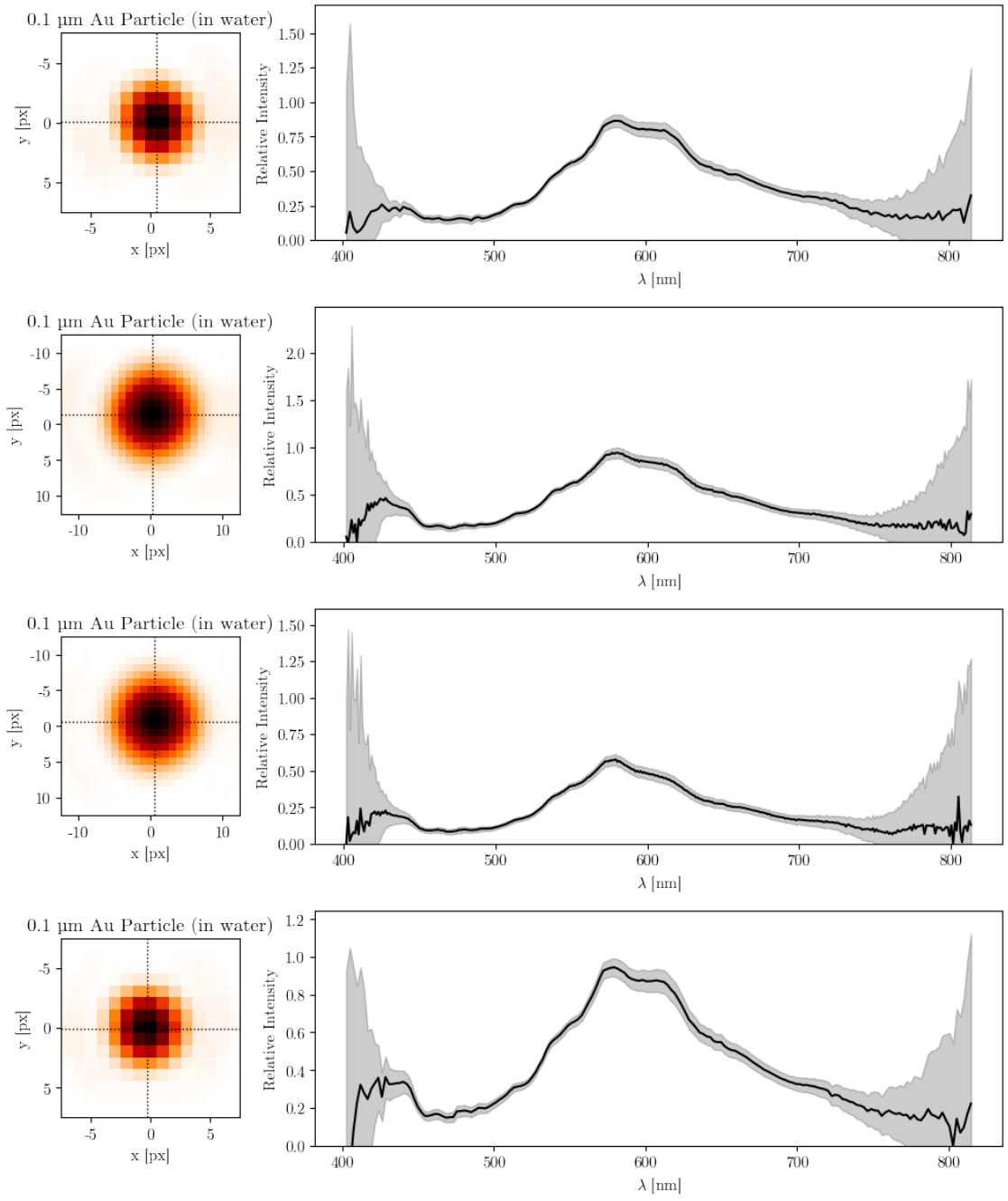


[25]:
```
CORRECTED_SPECS = [ np.empty_like(o.streak()[0,0,:]) for o in OBJECTS ]
CORRECTED_ERRS = [ np.empty_like(o.streak()[0,0,:]) for o in OBJECTS ]

for o in OBJECTS:
    LDA = o.lda( Region.STREAK )
    CORR = o.correction.evaluate( LDA )
    CORRECTED_SPECS[o.index] = AVG_SPECS[o.index] / CORR[0]
```

```
    ERR = SPEC_ERRS[o.index] + AVG_SPECS[o.index]/CORR[0]*CORR[1]
    CORRECTED_ERRS[o.index] = ERR / CORR[0]
```

[40]:
```
fig = plt.figure(figsize=(10,3*len(OBJECTS)), dpi=100)

axs = fig.add_gridspec(len(OBJECTS),4)

for o in OBJECTS:
    ax = fig.add_subplot(axs[ o.index , :1 ])

    ax.imshow(np.mean( o.region(Region.SPOT), axis=0 ),
              extent=o.extent(Region.SPOT),
              cmap=plotstyle.cmap('m'))

    ax.plot( [ mcx[o.index], mcx[o.index] ], [ o.extent(Region.SPOT)[2], o.
 ↪extent(Region.SPOT)[3] ],
             color=plotstyle.monochrome_fg(),
             ls=':', lw=1)
    ax.plot( [ o.extent(Region.SPOT)[0], o.extent(Region.SPOT)[1] ], [ mcy[o.
 ↪index], mcy[o.index] ],
             color=plotstyle.monochrome_fg(),
             ls=':', lw=1)

    ax.set_xlabel('x [px]')
    ax.set_ylabel('y [px]')
    ax.set_title( o.descriptor.particle.descr_str() )



    ax = fig.add_subplot(axs[ o.index , 1: ])

    ax.fill_between( o.lda(Region.STREAK),
                     CORRECTED_SPECS[o.index] - CORRECTED_ERRS[o.index],
                     CORRECTED_SPECS[o.index] + CORRECTED_ERRS[o.index],
                     color=plotstyle.monochrome_fg(),
                     alpha=plotstyle.err_alpha() )
    ax.plot( o.lda(Region.STREAK),
             CORRECTED_SPECS[o.index],
             color=plotstyle.monochrome_fg() )

    ax.set_ylim( bottom=0 )

    ax.set_xlabel('$\lambda$ [nm]')
    ax.set_ylabel('Relative Intensity')
```

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



[27]:
```
NORMS = [ np.sqrt( np.mean( np.square( AVG_SPECS[o.index] ) ) ) for o in␣
↪OBJECTS ]
```

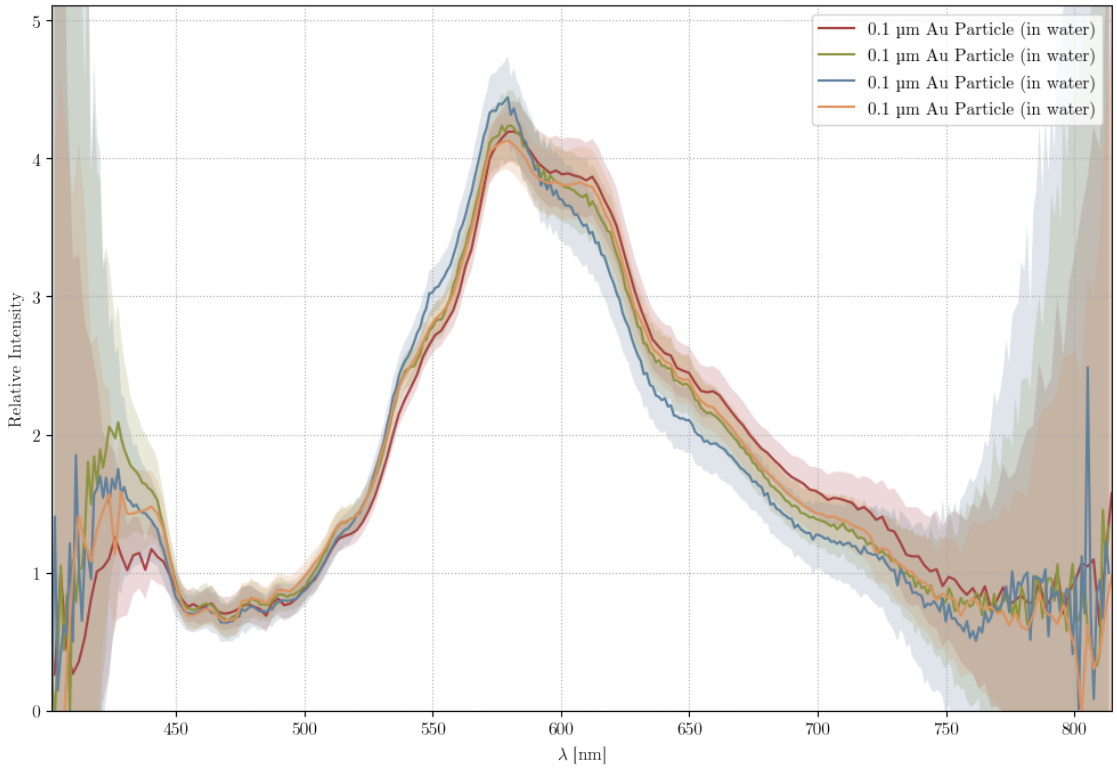
```
NORMS
```

[27]:
```
[0.2060841861264745,
 0.2220237703877113,
 0.13009501212756872,
 0.22882716465983594]
```

[28]:
```
lim_left = np.min( np.array([ o.correction.ldamin for o in OBJECTS ]) )
#lim_right = np.max( np.array([ o.correction.ldamax for o in OBJECTS ]) )
lim_right = np.max( np.array([ np.max( o.lda(Region.STREAK) ) for o in OBJECTS␣
↪]) )

lim_upper = np.max( np.array([ np.max( CORRECTED_SPECS[o.index]/NORMS[o.index]␣
↪) for o in OBJECTS ]) )*1.15
```

[41]:
```
fig = plt.figure(figsize=(10,7), dpi=100)

axs = fig.add_gridspec(1,1)

ax = fig.add_subplot(axs[ : , : ])

for o in OBJECTS:
    ax.fill_between( o.lda(Region.STREAK),
                     (CORRECTED_SPECS[o.index] - CORRECTED_ERRS[o.index])/
↪NORMS[o.index],
                     (CORRECTED_SPECS[o.index] + CORRECTED_ERRS[o.index])/
↪NORMS[o.index],
                     #color=plotstyle.monochrome_fg(),
                     alpha=plotstyle.err_alpha() )
    ax.plot( o.lda(Region.STREAK),
             CORRECTED_SPECS[o.index]/NORMS[o.index],
             #color=plotstyle.monochrome_fg(),
             label=o.descriptor.particle.descr_str() )


ax.grid()

ax.set_ylim( bottom=0, top=lim_upper )
ax.set_xlim( left=lim_left, right=lim_right )

ax.legend()
ax.set_xlabel('$\lambda$ [nm]')
ax.set_ylabel('Relative Intensity')



plt.tight_layout()
```

```
plt.show()
```



[30]:
```
from util import mie_theory
from scipy.interpolate import interp1d
```

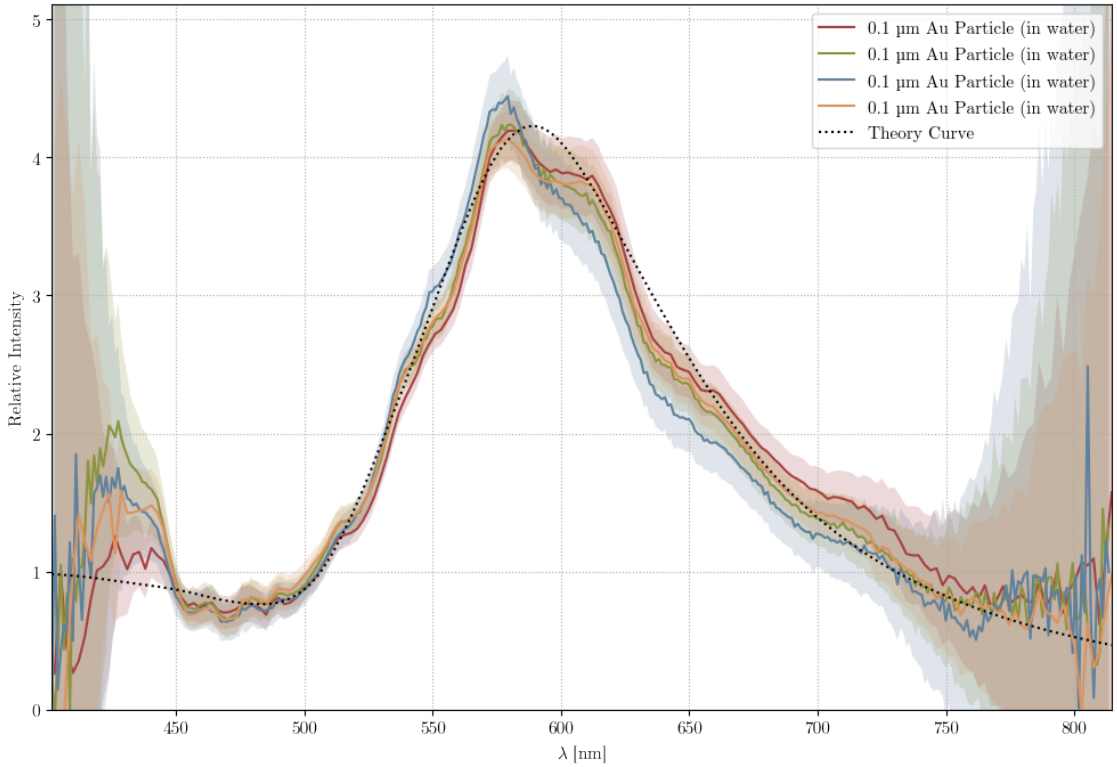
[31]:
```
read_lda, read_n, read_k = np.transpose(np.loadtxt('data/refractive-indices/
↪Au_nk.txt', skiprows=1))
n_Au = interp1d( read_lda*1e-6, read_n + 1j*read_k, kind=2 )
```

[32]:
```
TH_LDA = np.linspace( lim_left, lim_right, 100 )*1e-9
TH_SPEC_WATER = mie_theory.sigma_sca( TH_LDA, 50e-9, 1.33, n_Au, 100 )
TH_SPEC_GLASS = mie_theory.sigma_sca( TH_LDA, 50e-9, 1.45, n_Au, 100 )
TH_SPEC = 0.5*( TH_SPEC_WATER + TH_SPEC_GLASS )
```

[33]:
```
#TH_LDA *= 1e9
#TH_SPEC *= 1e12 # µm²
TH_NORM = np.sqrt( np.mean( np.square( TH_SPEC ) ) )
TH_NORM /= 2.1
```

[42]:
```
fig = plt.figure(figsize=(10,7), dpi=100)

axs = fig.add_gridspec(1,1)
```

```
ax = fig.add_subplot(axs[ : , : ])

for o in OBJECTS:
    ax.fill_between( o.lda(Region.STREAK),
                     (CORRECTED_SPECS[o.index] - CORRECTED_ERRS[o.index])/
 ↪NORMS[o.index],
                     (CORRECTED_SPECS[o.index] + CORRECTED_ERRS[o.index])/
 ↪NORMS[o.index],
                     #color=plotstyle.monochrome_fg(),
                     alpha=plotstyle.err_alpha() )
    ax.plot( o.lda(Region.STREAK),
             CORRECTED_SPECS[o.index]/NORMS[o.index],
             #color=plotstyle.monochrome_fg(),
             label=o.descriptor.particle.descr_str() )


ax.plot( TH_LDA*1e9,
         TH_SPEC/TH_NORM,
         ls=':',
         color=plotstyle.monochrome_fg(),
         label='Theory Curve')

ax.grid()

ax.set_ylim( bottom=0, top=lim_upper )
ax.set_xlim( left=lim_left, right=lim_right )

ax.legend()
ax.set_xlabel('$\lambda$ [nm]')
ax.set_ylabel('Relative Intensity')



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



[43]:
```
fig.savefig("{d}/scattering-spectra-vs-theory.pdf".format(d=DIRECTORY),␣
↪bbox_inches='tight', dpi=100)
fig.savefig("{d}/scattering-spectra-vs-theory.png".format(d=DIRECTORY),␣
↪bbox_inches='tight', dpi=100)
```

[ ]:

[ ]: