dOCT_drusen_example

November 16, 2021

0.1 Using directional OCT to understand photoreceptor visibility in AMD

0.1.1 Introduction

Investigators using adaptive optics (AO) retinal imaging instruments have reported reduced visibility of the cone photoreceptors overlying drusen as compared to those in unaffected portions of the same retinae or those in healthy retinae. Two compatible hypotheses have been offered to explain this phenomenon. Some have suggested that disease-related deformation of the photoreceptor outer segment (OS) reduces its ability to act as a wave guide, thus reducing the cell's familiar reflectance pattern. Others have suggested that drusen disorient the photoreceptors away from the eye's pupil, thus reducing the amount of reflected light that can be detected outside the eye.

In order to assess the contributions of these two potential factors to reduced photoreceptor visibility, we employed a custom research-grade OCT optical coherence tomography (OCT) system, along with a directional experimental protocol to acquire OCT images at a variety of positions in the pupil.

This repository is intended to illustrate the analytical approach that we employed.

0.1.2 Methods

Directional OCT (dOCT) was realized by translating the OCT imaging beam across the horizontal diameter of the pupil. As described below, our analytical approach did not require knowledge of the beam's position in the pupil, so we acquired images from one edge of the pupil to the other in increments of 0.64 mm. At each location, between 1200 and 1600 B-scans were acquired. These were aligned and averaged using a custom semi-rigid body algorithm. For each pupil location, at least 50 B-scans were averaged to generate a single averaged image. Pupil locations were omitted from analysis if 50 B-scans could not be identified with sufficiently high correlation (e.g., when there was too much eye movement normal to the plane of the B-scan). In the resulting average images, a semi-automated method was used to segment the inner-outer segment (IS/OS) band and fit it with a smooth curve. (The cone outer segment (COST) band was not studied because a clear boundary between COST and retinal pigmented epithelium (RPE) was rarely observed above drusen, and contamination by light scattered from RPE was unavoidable.)

At each A-scan of the average B-scan, the tangent to the IS/OS curve and the IS/OS amplitude (integrated over three pixels) was recorded. To account for the potentially confounding variable of overall image brightness (as a function of pupil position), the IS/OS amplitude was normalized by the amplitude of an overlying region in the outer nuclear layer (ONL), which was separately shown to have low directional dependence. Thus each amplitude value recorded represents the factor by which the IS/OS was more reflective than the corresponding ONL (in linear scale).

In addition, in each average B-scan, the boundaries of the drusen were determined visually, and each A-scan was labeled as ‘drusen’, ‘non-drusen’, or ‘transitional’. Transitional zones were omitted from further analysis.

Thus for each pupil entry position, an **averaged B-scan** was recorded, along with records of the following parameters, all functions of lateral position in the B-scan (i.e., A-scan location):

- IS/OS axial location
- IS/OS angle
- IS/OS amplitude
- distance to druse margin

Pupil entry position was not monitored, but the beam was stepped across the pupil. Images and quantification were labeled arbitrarily between integers $-M$ and N, with positive values designated with ‘p’ and negative values designated with ‘n’.

Visualizing data from one pupil position The following example illustrates the data recorded from a single pupil entry position.

[10]:
```
import numpy as np
from matplotlib import rcParams
#rcParams['font.family'] = 'serif'
#rcParams['font.serif'] = ['Times New Roman']
#rcParams['font.size'] = 10.0
from matplotlib import pyplot as plt

SMALL_SIZE = 10
MEDIUM_SIZE = 12
BIGGER_SIZE = 14
plt.rc('font', family='serif')
plt.rc('font', serif=['Times New Roman'])
plt.rc('font', size=SMALL_SIZE)          # controls default text sizes
plt.rc('axes', titlesize=SMALL_SIZE)     # fontsize of the axes title
plt.rc('axes', labelsize=MEDIUM_SIZE)    # fontsize of the x and y labels
plt.rc('xtick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
plt.rc('ytick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
plt.rc('legend', fontsize=SMALL_SIZE)    # legend fontsize
plt.rc('figure', titlesize=MEDIUM_SIZE)  # fontsize of the figure title

import os
import scipy.optimize as spo
import scipy.stats as sps

def despine(ax=None):
    """Remove the spines from a plot. (These are the lines drawn
    around the edge of the plot.)"""
    if ax is None:
        ax = plt.gca()
    ax.spines["top"].set_visible(False)
    ax.spines["right"].set_visible(False)
    ax.spines["bottom"].set_visible(False)
    ax.spines["left"].set_visible(False)

subject_folders = ['data/subject_01', 'data/subject_02', 'data/subject_03',␣
 ↪'data/subject_04']
drusen_labels = ['non-drusen','drusen']
figure_directory = '/home/rjonnal/Dropbox/apps/Overleaf/Directional OCT &␣
 ↪photoreceptor visibility over drusen/figs/'

plt.style.use('seaborn-deep')
color_cycle = plt.rcParams['axes.prop_cycle'].by_key()['color']
subject_markers = ['s','o','d','^']
print_dpi = 300
screen_dpi = 100
figure_size = (5,4)
raw_data_alpha = 0.2
raw_data_markersize = 3
plot_linewidth = 1
fitting_line = 'k--'

pupil_position = 'm01'
subject_folder = subject_folders[1]

#pupil_position = 'p01'
#subject_folder = subject_folders[0]

#pupil_position = 'm01'
#subject_folder = subject_folders[3]

bscan_arrows = {}

def savefig(pngfn,dpi):
    pdffn = pngfn.replace('.png','.pdf')
    plt.savefig(pngfn,dpi=dpi)
    plt.savefig(pdffn,dpi=dpi)

def scalebar(x_len_um,z_len_um,x0_um=150,z0_um=1380):

    x_um_per_px = 1800.0/800.0
    z_um_per_px = 0.97

    um2pxx = lambda um: um/x_um_per_px
    um2pxz = lambda um: um/z_um_per_px

    x_len = um2pxx(x_len_um)
    z_len = um2pxz(z_len_um)
    x0 = um2pxx(x0_um)
    z0 = um2pxz(z0_um)

    plt.plot([x0,x0+x_len],[z0,z0],'y-',linewidth=5,alpha=1)
    plt.plot([x0,x0],[z0,z0+z_len],'y-',linewidth=5,alpha=1)

# load B-scan and convert to dB
bscan_fn = os.path.join(subject_folder,'pupil_position_%s_average_bscan.
 ↪npy'%pupil_position)
bscan = np.load(bscan_fn)
print(bscan.shape)
sys.exit()
dB = 20*np.log10(bscan)
clim = (55,85)

# load IS/OS location
isos_location_fn = os.path.
 ↪join(subject_folder,'pupil_position_%s_isos_axial_location.
 ↪npy'%pupil_position)
isos_location = np.load(isos_location_fn)
x = np.arange(len(isos_location))

# load IS/OS angle
isos_angle_fn = os.path.join(subject_folder,'pupil_position_%s_isos_angle.
 ↪npy'%pupil_position)
isos_angle = np.load(isos_angle_fn)
isos_angle = isos_angle/2.0
#isos_angle = (isos_angle/180*np.pi)*1000.0

# load IS/OS amplitude
isos_amplitude_fn = os.path.
 ↪join(subject_folder,'pupil_position_%s_isos_amplitude.npy'%pupil_position)
isos_amplitude = np.load(isos_amplitude_fn)

x_limits = (50,750)
y_limits = (1900,1400)
print('dynamic range: %0.1f'%(dB[y_limits[1]:y_limits[0],x_limits[0]:
 ↪x_limits[1]].max()-dB[y_limits[1]:y_limits[0],x_limits[0]:x_limits[1]].
 ↪min()))



# show the B-scan and resulting IS/OS curve
plt.figure(figsize=(7,5),dpi=screen_dpi)
plt.axes([0.05,0.05,0.8,0.9])
plt.imshow(dB,clim=clim,cmap='gray',aspect='equal')
scalebar(100,100)
```

```
plt.ylim(y_limits)
plt.xlim(x_limits)
plt.xticks([])
plt.yticks([])
#plt.colorbar()

plt.plot(x,isos_location+5,label='IS/OS␣
 ↪location',alpha=1,color=color_cycle[2],linewidth=1)
plt.plot(x,isos_location-5,label='IS/OS␣
 ↪location',alpha=1,color=color_cycle[2],linewidth=1)
plt.plot(x,isos_location-35,label='IS/OS␣
 ↪location',alpha=1,color=color_cycle[4],linewidth=1)
plt.plot(x,isos_location-45,label='IS/OS␣
 ↪location',alpha=1,color=color_cycle[4],linewidth=1)
bscan_arrows[('data/subject_02','m01')] = [(220,380),(340,380)]
layer_labels = {}
layer_labels[('data/subject_02','m01')] = {'ELM':270, 'ISOS':287, 'COST':305,␣
 ↪'RPE':323}

if (subject_folder,pupil_position) in bscan_arrows.keys():
    arrows = bscan_arrows[(subject_folder,pupil_position)]
    for xa,ya in arrows:
        print(xa,ya)
        plt.arrow(xa-20,ya+1400+20,20,-20,color='w',width=5)

if (subject_folder,pupil_position) in layer_labels.keys():
    d = layer_labels[(subject_folder,pupil_position)]
    for label in d.keys():
        xl = 753
        yl = d[label]+1400
        plt.text(xl,yl,label)

#plt.legend(frameon=False)
savefig(figure_directory+'bscan_with_trace.png',dpi=print_dpi)

# plot angle and amplitude as functions of x
plt.figure(figsize=figure_size,dpi=screen_dpi)
plt.plot(x,isos_angle,label='IS/OS angle (deg)')
plt.plot(x,20*np.log10(isos_amplitude),label='IS/OS amplitude (dB)')
plt.xlim(x_limits)
plt.legend(frameon=False)
plt.xlabel('x location')
despine()
savefig(figure_directory+'amplitude_vs_location.png',dpi=print_dpi)
```

```
(2122, 858)
```

```
        ␣
↪---------------------------------------------------------------------------

       NameError                                 Traceback (most recent call␣
↪last)

       <ipython-input-10-0c6548234a3d> in <module>
        84 bscan = np.load(bscan_fn)
        85 print(bscan.shape)
   ---> 86 sys.exit()
        87 dB = 20*np.log10(bscan)
        88 clim = (55,85)


       NameError: name 'sys' is not defined
```

Data aggregated over pupil positions Data associated with the pupil-position-specific B-scans were aggregated, for drusen and non-drusen regions of the images separately. For each subject, these are stored in two files:

- `directionality_raw_data_drusen_centered.npy`
- `directionality_raw_data_nondrusen_centered.npy`

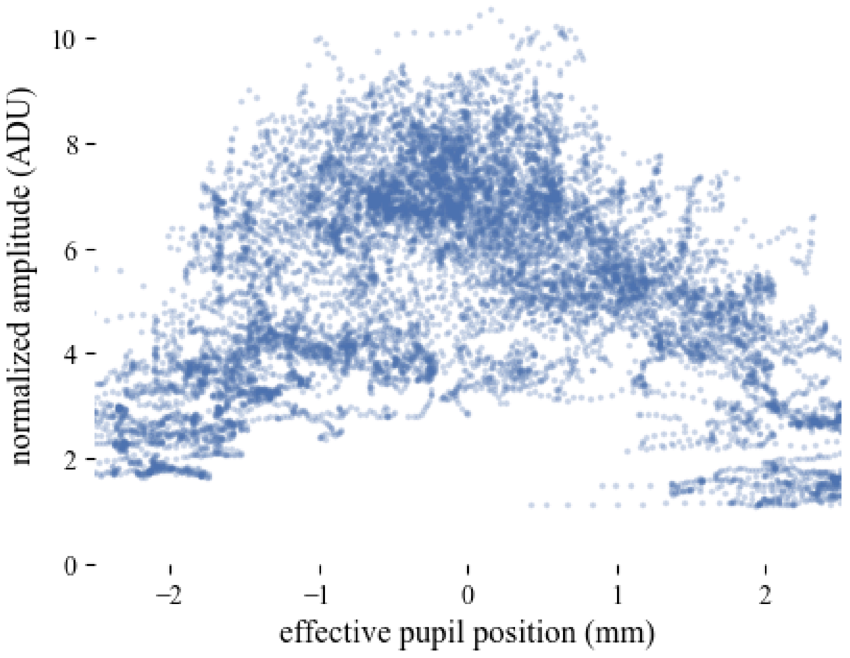
Each contains an Nx2 matrix, where N represents the number of data points aggregated from all pupil positions. The two columns represent the IS/OS curve angle and normalized amplitude. Because the displacement of the Stiles-Crawford peak from the pupil center is not of interest, and because the pupil position was not tracked, angles for each subject were zero-centered by subtracting the average angle.

Visualization of aggregated data The plot below illustrates the data aggregated from one subject's non-drusen IS/OS. In order to calculate the angle correctly, the x and z sampling densities of the OCT image must be considered. The width of the OCT scan on the retina was $2mm$, sampled with 800 A-scans, resulting in a pixel width of $2.5\mu m$. The depth corresponding to a single pixel was $5\mu m$, which requires a scaling of the angle by a factor of 2. Assuming an eye focal length f of $16.7mm$, and a tangent angle θ, the effective pupil position x is given by:

$$x = \tan(\theta)f$$

```
[2]: def filename_to_x_mm_amplitude(fn):
         angle_amplitude = np.load(fn)
         # divide the angle by 2 to account for x-z sampling anisotropy
         angle = angle_amplitude[:,0]/2.0
         angle = (angle/180*np.pi)
         x_mm = np.tan(angle)*16.67
         amplitude = angle_amplitude[:,1]
```

```
    return x_mm,amplitude

nondrusen_aggregate_fn = os.path.
 ↪join(subject_folder,'directionality_raw_data_nondrusen_centered.npy')
x_mm,amplitude = filename_to_x_mm_amplitude(nondrusen_aggregate_fn)

plt.figure(figsize=figure_size,dpi=screen_dpi)
plt.plot(x_mm,amplitude,'.',alpha=raw_data_alpha,markersize=raw_data_markersize)
plt.gca().set_ylim(bottom=0)
plt.xlim((-2.5,2.5))

plt.ylabel('normalized amplitude (ADU)')
plt.xlabel('effective pupil position (mm)')
despine()
savefig(figure_directory+'amplitude_vs_angle.png',dpi=print_dpi)
plt.show()
```



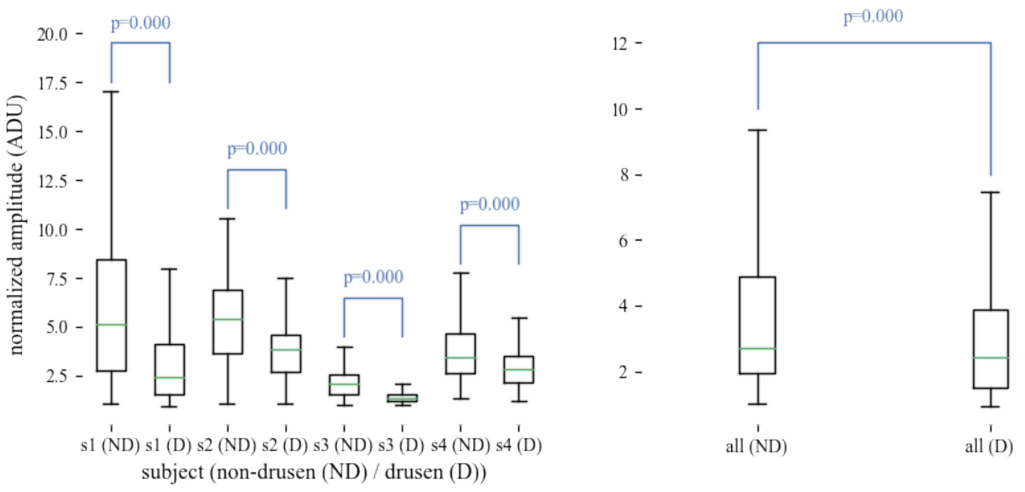
Testing for gross differences between nondrusen and drusen reflectance Before we begin modeling the directionality of the IS/OS band, we sought to see if there were any statistically significiant differences in the IS/OS amplitude, between nondrusen and drusen regions of the scan. We did this separately for each subject, and then together.

[3]:
```
nondrusen_amplitudes = []
drusen_amplitudes = []
```

```
data_grid = []
xtl = []
ps = []
data_grid_all = [[],[]]

for subject_folder in subject_folders:
    nondrusen_aggregate_fn = os.path.
 ↪join(subject_folder,'directionality_raw_data_nondrusen_centered.npy')
    drusen_aggregate_fn = os.path.
 ↪join(subject_folder,'directionality_raw_data_drusen_centered.npy')

    _,nondrusen_amplitude = filename_to_x_mm_amplitude(nondrusen_aggregate_fn)
    _,drusen_amplitude = filename_to_x_mm_amplitude(drusen_aggregate_fn)
    p = sps.ttest_ind(nondrusen_amplitude,drusen_amplitude).pvalue
    ps.append(p)
    print('nondrusen v drusen independent t-test for %s, p=%0.5f'%
            (subject_folder,p))

    nondrusen_amplitudes.append(nondrusen_amplitude)
    drusen_amplitudes.append(drusen_amplitude)
    data_grid.append(nondrusen_amplitude)
    xtl.append('s%s (ND)'%subject_folder[-1])
    xtl.append('s%s (D)'%subject_folder[-1])
    data_grid.append(drusen_amplitude)
    data_grid_all[0]+=list(nondrusen_amplitude)
    data_grid_all[1]+=list(drusen_amplitude)


p = sps.ttest_ind(data_grid_all[0],data_grid_all[1]).pvalue
print('nondrusen v drusen independent t-test for all subjects, p=%0.5f'%p)
def sig(t1,t2,y1,y2,p=1.0,height=2,padding=0.5):
    plt.
 ↪plot([t1,t1,t2,t2],[y1+padding,y1+height+padding,y1+height+padding,y2+padding],color=color_
    plt.text((t1+t2)/2.0,y1+2*padding+height,'p=%0.
 ↪3f'%p,ha='center',va='bottom',color=color_cycle[0])

plt.figure(figsize=(figure_size[0]*2,figure_size[1]),dpi=screen_dpi)

plt.subplot(1,2,1)
ph = plt.boxplot(data_grid,showfliers=False,labels=xtl,notch=False)
whiskers = ph['whiskers']
maxes = np.zeros(8)
for w in whiskers:
    d = w.get_data()
    if d[1].max()>maxes[int(d[0][0])-1]:
        maxes[int(d[0][0])-1] = d[1].max()
```

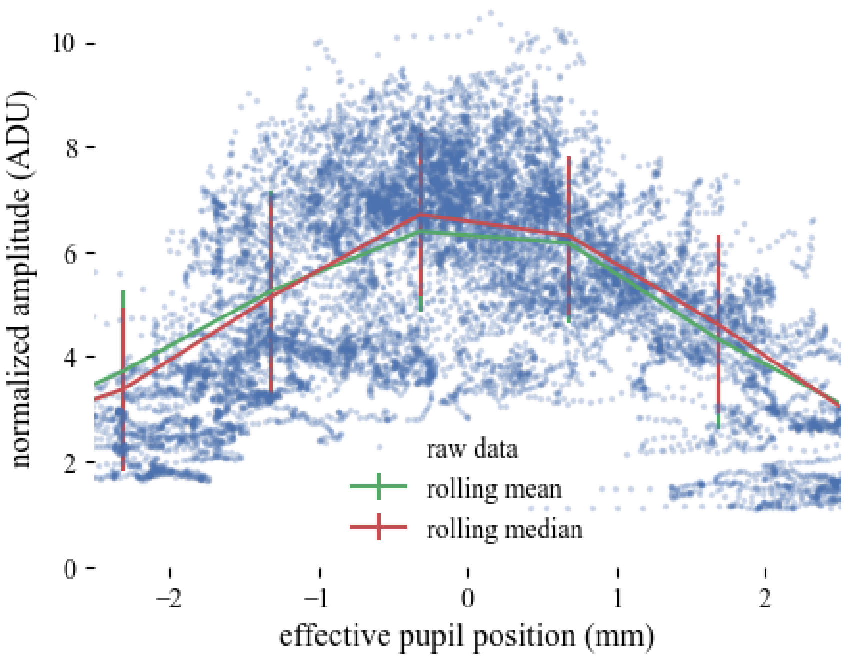
```
for pair_idx in range(4):
    t1 = pair_idx*2+0
    t2 = pair_idx*2+1
    y1 = maxes[t1]
    y2 = maxes[t2]
    sig(t1+1,t2+1,y1,y1,p=ps[pair_idx])
plt.ylabel('normalized amplitude (ADU)')
plt.xlabel('subject (non-drusen (ND) / drusen (D))')
despine()



plt.subplot(1,2,2)
ph = plt.boxplot(data_grid_all,showfliers=False,labels=['all (ND)','all␣
 ↪(D)'],notch=False)
sig(1,2,9.5,7.5,p=p)
despine()
savefig(figure_directory+'gross_comparisons.png',dpi=print_dpi)
plt.show()
```

```
nondrusen v drusen independent t-test for data/subject_01, p=0.00000
nondrusen v drusen independent t-test for data/subject_02, p=0.00000
nondrusen v drusen independent t-test for data/subject_03, p=0.00000
nondrusen v drusen independent t-test for data/subject_04, p=0.00000
nondrusen v drusen independent t-test for all subjects, p=0.00000
```



Improving data visualization using rolling averaging While a clear dependence of IS/OS amplitude on angle of illumination is visible in the figure above, substantial variance at all angles makes the relationship difficult to appreciate visually. To improve visualization, a rolling average technique was employed. A window of width 5.0° was stepped across the range of angles present in a subject's measurements, in increments of 2.0°, and the average amplitude and standard deviation of amplitude were recorded at each location. The result of this averaging is shown below.

[4]:
```
def rolling_median(pupil_position,amp,window_width=2.0,step_size=1.
↪0,diagnostics=False):

    # find the start and end angles
    t_start = np.min(pupil_position)
    t_end = np.max(pupil_position)
    #print(t_start,t_end)
    window_centers = []
    amplitude_mean = []
    amplitude_median = []
    amplitude_std = []
    amplitude_sem = []

    n_points = []
    for t in np.arange(t_start,t_end+step_size,step_size):

        # find he indices in the angle array where the angle falls in our window
        idx = np.where(np.
↪logical_and(pupil_position>=t,pupil_position<t+window_width))[0]

        if len(idx)>0:
            window_centers.append(t+window_width/2.0)
            amplitude_mean.append(np.mean(amp[idx]))
            amplitude_median.append(np.median(amp[idx]))
            amplitude_std.append(np.std(amp[idx]))
            amplitude_sem.append(np.std(amp[idx])/np.sqrt(float(len(idx))))
            n_points.append(len(idx))

    return np.array(window_centers),np.array(amplitude_mean),np.
↪array(amplitude_median),np.array(amplitude_std),np.array(amplitude_sem)

window_centers, amplitude_mean, amplitude_median, amplitude_std, amplitude_sem␣
↪= rolling_median(x_mm,amplitude)
plt.figure(figsize=figure_size,dpi=screen_dpi)
# plot the raw data with low alpha
plt.plot(x_mm,amplitude,'.
↪',alpha=raw_data_alpha,markersize=raw_data_markersize,label='raw data')
# plot the average with standard deviation bars
plt.errorbar(window_centers,amplitude_mean,amplitude_std,label='rolling mean')
plt.errorbar(window_centers,amplitude_median,amplitude_std,label='rolling␣
↪median')
plt.xlabel('effective pupil position (mm)')
plt.ylabel('normalized amplitude (ADU)')
plt.gca().set_ylim(bottom=0)
plt.xlim((-2.5,2.5))
plt.legend(frameon=False)
despine()
savefig(figure_directory+'rolling_median.png',dpi=print_dpi)
plt.show()
```



A model for directionality The amplitude distribution in the pupil of light backscattered by the retina is typically described as a sum of a constant (diffuse) component and a Gaussian (directional) component:

$$a(x, y) = B + A \times 10^{-\rho[(x-x_0)^2+(y-y_0)^2]},$$

where B represents the diffuse component, A represents the height of the Gaussian component, ρ represents the directionality coefficient, x (y) represents the horizontal (vertical) position in the pupil, and x_0 (y_0) the location of the Stiles-Crawford peak. x, x_0, y, and y_0 are typically expressed in mm, which results in ρ having units of mm^{-2}. If, as in the present work, only the horizontal dimension of the directionality function is modeled, the equation simplifies to:

$$a(x) = B + A \times 10^{-\rho[(x-x_0)^2]}$$

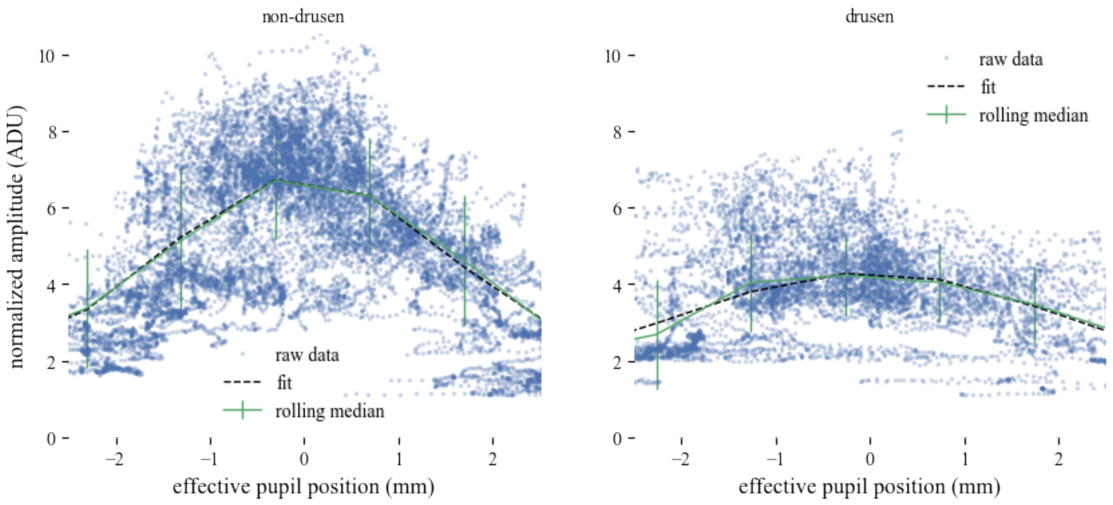
The width of Gaussian distributions is more commonly described by standard deviation σ, and from that definition it is apparent that $\rho = \frac{1}{2\sigma^2}$, and that wider distributions (higher σ) have lower directionality coefficients ρ, and narrower ones have higher ρ.

Estimating B, A, and ρ To estimate these three parameters, the smoothed functions described above were fit with a four parameter model. (x_0 was used to optimize the fits, but was not considered in further analysis.) In the next step, the median-filtered data from one subject's non-drusen and drusen regions are fit separately, and the results of the fits visualized side by side.

[5]:
```python
def gaussian(x_mm, B, A, rho, x0_mm):
    return B + A*(10**(-rho*(x_mm-x0_mm)**2))

amax = None

def fit_data_from_file(fn):
    x_mm,amp = filename_to_x_mm_amplitude(fn)
    wc,amean,amed,std,sem = rolling_median(x_mm,amp)
    fit_params = spo.curve_fit(gaussian,wc,amed)[0]
    return fit_params, x_mm, amp, wc, amed, std

subject_folder='data/subject_02'

plt.figure(figsize=(figure_size[0]*2,figure_size[1]),dpi=screen_dpi)
titles = ['non-drusen','drusen']
print(subject_folder)
for idx,fn in enumerate(['directionality_raw_data_nondrusen_centered.
 ↪npy','directionality_raw_data_drusen_centered.npy']):
    ffn = os.path.join(subject_folder,fn)
    plt.subplot(1,2,idx+1)
    fit_params, x_mm, amp, wc, amed, std = fit_data_from_file(ffn)
    if idx==0:
        amax = np.max(amp)
    afit = gaussian(wc,*fit_params)

    x_mm0 = x_mm-fit_params[-1]
    wc0 = wc-fit_params[-1]

    plt.plot(x_mm0,amp,'.
 ↪',alpha=raw_data_alpha,markersize=raw_data_markersize,label='raw data')
    # plot the average with standard deviation bars
    plt.errorbar(wc0,amed,std,label='rolling median',linewidth=plot_linewidth)
    plt.plot(wc0,afit,fitting_line,label='fit',linewidth=plot_linewidth)
    plt.xlabel('effective pupil position (mm)')
    if idx==0:
        plt.ylabel('normalized amplitude (ADU)')
    plt.gca().set_ylim(bottom=0)
    plt.gca().set_ylim(top=amax)
    plt.xlim((-2.5,2.5))
    plt.legend(frameon=False)
    plt.title(titles[idx])
    despine()
```

```
savefig(figure_directory+'fitting_normal_drusen.png',dpi=print_dpi)
```

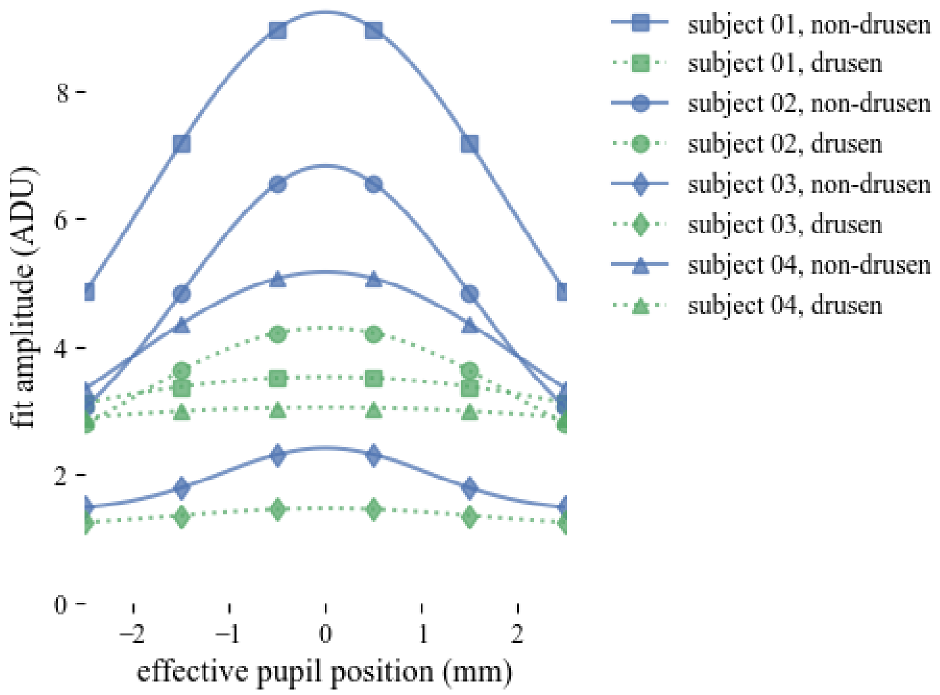
```
data/subject_02

<ipython-input-5-e57b5b72c74a>:2: RuntimeWarning: overflow encountered in power
  return B + A*(10**(-rho*(x_mm-x0_mm)**2))
```



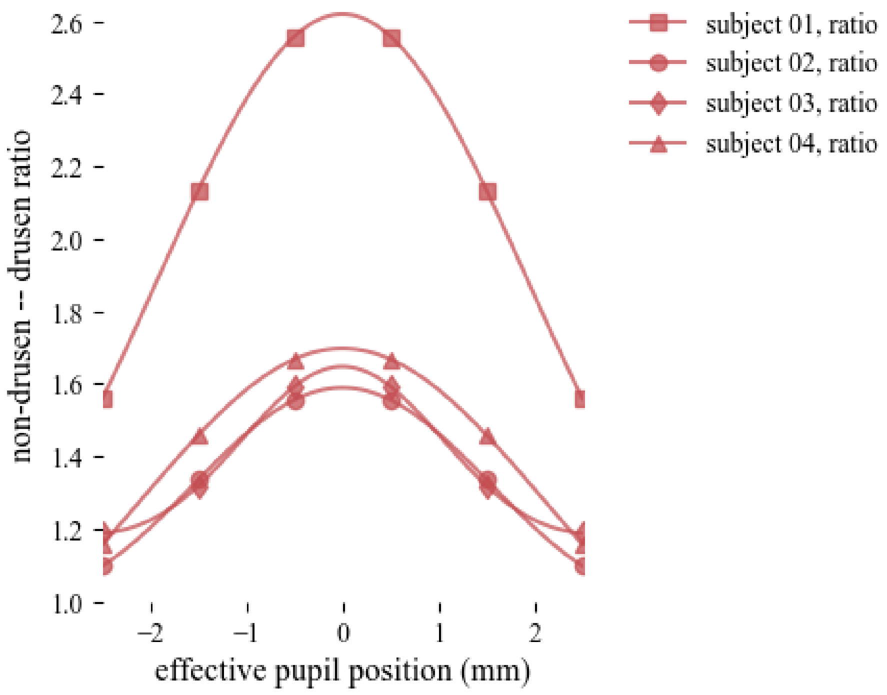
Comparing B, A, and ρ between non-drusen and drusen regions

[6]:
```
# positions to visualize fits:
x_mm_fit = np.arange(-2.5,2.55,0.05)

linestyles = ['-',':']
marker_alpha = 0.75

B_grid = np.zeros((len(subject_folders),len(drusen_labels)))
A_grid = np.zeros((len(subject_folders),len(drusen_labels)))
rho_grid = np.zeros((len(subject_folders),len(drusen_labels)))
fig1 = plt.figure(figsize=(figure_size[0],figure_size[1]),dpi=screen_dpi)
fig2 = plt.figure(figsize=(figure_size[0],figure_size[1]),dpi=screen_dpi)
ax1 = fig1.add_axes([.1,.1,.5,.8])
ax2 = fig2.add_axes([.1,.1,.5,.8])

for subject_idx,subject_folder in enumerate(subject_folders):
    subject_label = os.path.split(subject_folder)[1].replace('_',' ')
    for drusen_idx,fn in enumerate(['directionality_raw_data_nondrusen_centered.
 ↪npy','directionality_raw_data_drusen_centered.npy']):
        drusen_label = drusen_labels[drusen_idx]
        full_fn = os.path.join(subject_folder,fn)
        fit_params, x_mm, amp, wc, amed, std = fit_data_from_file(full_fn)
        B, A, rho, x0_mm = fit_params
        y_fit = gaussian(x_mm_fit, B, A, rho, 0.0)
        ax1.
 ↪plot(x_mm_fit,y_fit,marker=subject_markers[subject_idx],alpha=marker_alpha,markevery=20,col
 ↪label='%s, %s'%(subject_label,drusen_label))
        if drusen_idx==0:
            nondrusen_fit = y_fit
        else:
            ratio = nondrusen_fit/y_fit
            ax2.
 ↪plot(x_mm_fit,ratio,marker=subject_markers[subject_idx],alpha=marker_alpha,markevery=20,col
 ↪ratio'%subject_label)

        B_grid[subject_idx,drusen_idx] = B
        A_grid[subject_idx,drusen_idx] = A
        rho_grid[subject_idx,drusen_idx] = rho

plt.figure(fig1.number)
plt.gca().set_ylim(bottom=0)
plt.xlim((-2.5,2.5))
plt.xlabel('effective pupil position (mm)')
plt.ylabel('fit amplitude (ADU)')
plt.legend(frameon=False,bbox_to_anchor=(1.05, 1), loc='upper left')
despine()
savefig(figure_directory+'fits_all_subjects.png',dpi=print_dpi)

plt.figure(fig2.number)
plt.gca().set_ylim(bottom=1)
plt.xlim((-2.5,2.5))
plt.xlabel('effective pupil position (mm)')
plt.ylabel('non-drusen -- drusen ratio')
plt.legend(frameon=False,bbox_to_anchor=(1.05, 1), loc='upper left')
despine()
savefig(figure_directory+'fit_ratios_all_subjects.png',dpi=print_dpi)
plt.show()
```

```
<ipython-input-5-e57b5b72c74a>:2: RuntimeWarning: overflow encountered in power
  return B + A*(10**(-rho*(x_mm-x0_mm)**2))
<ipython-input-5-e57b5b72c74a>:2: RuntimeWarning: overflow encountered in power
  return B + A*(10**(-rho*(x_mm-x0_mm)**2))
<ipython-input-5-e57b5b72c74a>:2: RuntimeWarning: overflow encountered in power
  return B + A*(10**(-rho*(x_mm-x0_mm)**2))
<ipython-input-5-e57b5b72c74a>:2: RuntimeWarning: overflow encountered in power
  return B + A*(10**(-rho*(x_mm-x0_mm)**2))
```



subject 01, non-drusen
subject 01, drusen
subject 02, non-drusen
subject 02, drusen
subject 03, non-drusen
subject 03, drusen
subject 04, non-drusen
subject 04, drusen
fit amplitude (ADU)
effective pupil position (mm)
0
2
4
6
8
−2
−1
0
1
2

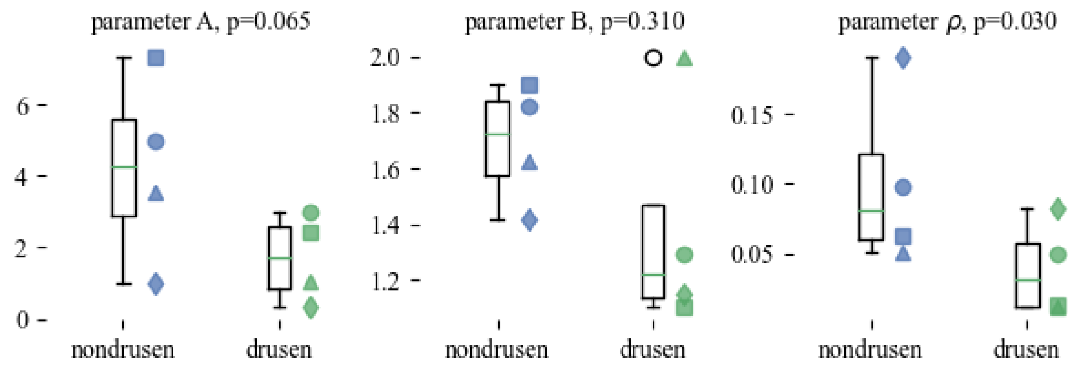


subject 01, ratio
subject 02, ratio
subject 03, ratio
subject 04, ratio
non-drusen -- drusen ratio
effective pupil position (mm)
1.0
1.2
1.4
1.6
1.8
2.0
2.2
2.4
2.6
−2
−1
0
1
2

[7]:
```
grids = [A_grid,B_grid,rho_grid]
labels = ['A','B',r'$\rho$']
xtl = ['nondrusen','drusen']
markeralpha = 0.75

plt.figure(figsize=(figure_size[0]*1.5,figure_size[1]*.5),dpi=screen_dpi)
for idx,(grid,label) in enumerate(zip(grids,labels)):

    print(label)
    print('mean')
    print(np.mean(grid,axis=0))
    print('std')
    print(np.std(grid,axis=0))

    tres = sps.ttest_rel(grid[:,0],grid[:,1])
    p = tres.pvalue
    print(p)

    if label=='B':
        # do a second t-test excluding the outlier
        tres = sps.ttest_rel(grid[:3,0],grid[:3,1])
        p_nooutlier = tres.pvalue
        print(p_nooutlier)

    plt.subplot(1,3,idx+1)
    plt.boxplot(grid,labels=xtl,notch=False)

    for idx in range(len(subject_folders)):
        plt.plot(1.
 ↪2,grid[idx,0],subject_markers[idx],color=color_cycle[0],alpha=markeralpha)
        plt.plot(2.
 ↪2,grid[idx,1],subject_markers[idx],color=color_cycle[1],alpha=markeralpha)

    plt.title('parameter %s, p=%0.3f'%(label,p))
    despine()

savefig(figure_directory+'parameters_all_subjects.png',dpi=print_dpi)
plt.show()
```

```
A
mean
[4.22066502 1.69779524]
std
[2.30118318 1.06990656]
0.06478700447883119
B
mean
```

```
[1.69128098 1.38570597]
std
[0.18684985 0.35896143]
0.31015640967256397
0.07480797749210936
$\rho$
mean
[0.10040399 0.03871941]
std
[0.05480807 0.02914497]
0.029984353846298824
```



[8]:
```
for subject_idx,subject_folder in enumerate(subject_folders):
    subject_label = os.path.split(subject_folder)[1].replace('_',' ')
    x_mm_both = []
    amp_both = []
    for drusen_idx,fn in enumerate(['directionality_raw_data_nondrusen_centered.
 ↪npy','directionality_raw_data_drusen_centered.npy']):
        drusen_label = drusen_labels[drusen_idx]
        full_fn = os.path.join(subject_folder,fn)
        fit_params, x_mm, amp, wc, amed, std = fit_data_from_file(full_fn)
        x_mm_both = x_mm_both + list(x_mm)
        amp_both = amp_both + list(amp)
```

```
<ipython-input-5-e57b5b72c74a>:2: RuntimeWarning: overflow encountered in power
  return B + A*(10**(-rho*(x_mm-x0_mm)**2))
```

[9]:
```
# Other calculations

# sampling density in z

points_per_spectrum = 2048
```

```
wavelength_spectrum = np.polyval([4.1e-11,8.01e-7],np.
↪arange(points_per_spectrum))
# k_in is just 2*pi/wavlength_spectrum, and k_out is a linearized version.
k_in = 2.0*np.pi/wavelength_spectrum
k_out = np.linspace(k_in[0],k_in[-1],points_per_spectrum)
k_range = k_out[0]-k_out[-1]
print(1/k_range/1.38)
```

```
9.740456428983476e-07
```