**SUPPLEMENTARY FILE 1**

**Ca2+ Image Processing Routines**

The following texts are executed as scripts within the FLIKA software package (<https://github.com/flika-org/flika>) to generate image stacks representing temporal and spatial fluctuations in Ca2+ image data.

**Temporal SD fluctuations algorithm**

#parameters#

sigma = 2 **#** *In pixels.* *Change this number to vary the strength of the gaussian blur*

sampling\_interval = 10 **#** *frame duration in ms*

q = 0.186 **#** *scalar constant to correct for photon-shot noise, derived as described in Fig. S1*

low\_cutoff = 3 **#** *low cut off for Butterworth filter in Hz*

high\_cutoff = 20 **#** *high cut off Butterworth filter in Hz*

filter\_order = 3 **#** *increasing filter order increases the steepness of the Butterworth filter*

boxcar\_width = 160 **#** *in milliseconds*

#red text denotes user specified parameters#

# run after specifying above parameters and subtracting black level #

from scipy.ndimage.filters import convolve

sampling\_rate = 1/(sampling\_interval/1000) # in Hz

try:

 assert high\_cutoff <= .5 \* sampling\_rate

except AssertionError:

 print('High Frequency Cutoff is above the Nyquist frequency. Lower your high frequency cutoff')

high\_cutoff\_scaled = high\_cutoff / (sampling\_rate/2)

low\_cutoff\_scaled = low\_cutoff / (sampling\_rate/2)

boxcar\_frames = int(np.round(boxcar\_width / sampling\_interval))

#A = np.sqrt(10) \* np.random.randn(10000, 10,10) + 10

#Window(A, 'original image')

nFrames = g.win.mt

prefilter = gaussian\_blur(sigma, keepSourceWindow=True)

postfilter = butterworth\_filter(filter\_order, low\_cutoff\_scaled, high\_cutoff\_scaled, keepSourceWindow=True)

A = prefilter.image

B = postfilter.image

mean\_A = convolve(A, weights=np.full((boxcar\_frames,1,1),1.0/boxcar\_frames))

mean\_B = convolve(B, weights=np.full((boxcar\_frames,1,1),1.0/boxcar\_frames))

variance\_B = np.zeros\_like(B)

print('Calculating variance')

for i in np.arange(nFrames):

 print('{}/{}'.format(i, nFrames))

 mean\_frame = mean\_B[i]

 i0 = int(i-boxcar\_frames/2)

 if i0<0:

 i0 = 0

 i1 = int(i+boxcar\_frames/2)

 if i1 > nFrames:

 i1 = nFrames

 cutout\_frames = B[i0:i1]

 variance\_B[i] = np.mean((cutout\_frames - mean\_frame)\*\*2, 0)

sqrt\_B = np.sqrt(variance\_B)

mean\_A[mean\_A<0] = 0 #removes negative values

sqrt\_Bf = sqrt\_B - np.sqrt(q\*mean\_A)

Window(sqrt\_Bf, 'stdev minus sqrt mean')

#Window(variance\_B - (q\*mean\_A), 'Variance minus mean')

**Spatial SD fluctuations algorithm**

subtract(500, keepSourceWindow=False) **#** *camera black level*

trim(1000, 4000, increment=1, delete=False, keepSourceWindow=False) **#** *removes frames from beginning and ending of image stack*

I1=trim(1327, 1345, increment=1, delete=True, keepSourceWindow=False) **#** *removes flash artifact*

multiply(0.11, keepSourceWindow=True) **#** *scaled raw fluorescence*

I2=sqrt(2)

I1.setAsCurrentWindow()

butterworth\_filter(3, 3, 20, 125, keepSourceWindow=True)

I3=gaussian\_blur(2, 8, keepSourceWindow=False) # *sigma values used to calculate the difference of Gaussian blur functions*

image\_calculator(I3,I3,'Multiply',keepSourceWindow=True)

I4=sqrt(2)

image\_calculator(I4,I2,'Subtract',keepSourceWindow=True)

#red text denotes user defined parameters#