Biophysical Journal, Volume *115*

Supplemental Information

Tuning Length Scales of Small Domains in Cell-Derived Membranes and Synthetic Model Membranes

Caitlin E. Cornell, Allison D. Skinkle, Shushan He, Ilya Levental, Kandice R. Levental, and Sarah L. Keller

1 SUPPLEMENTARY FIGURES

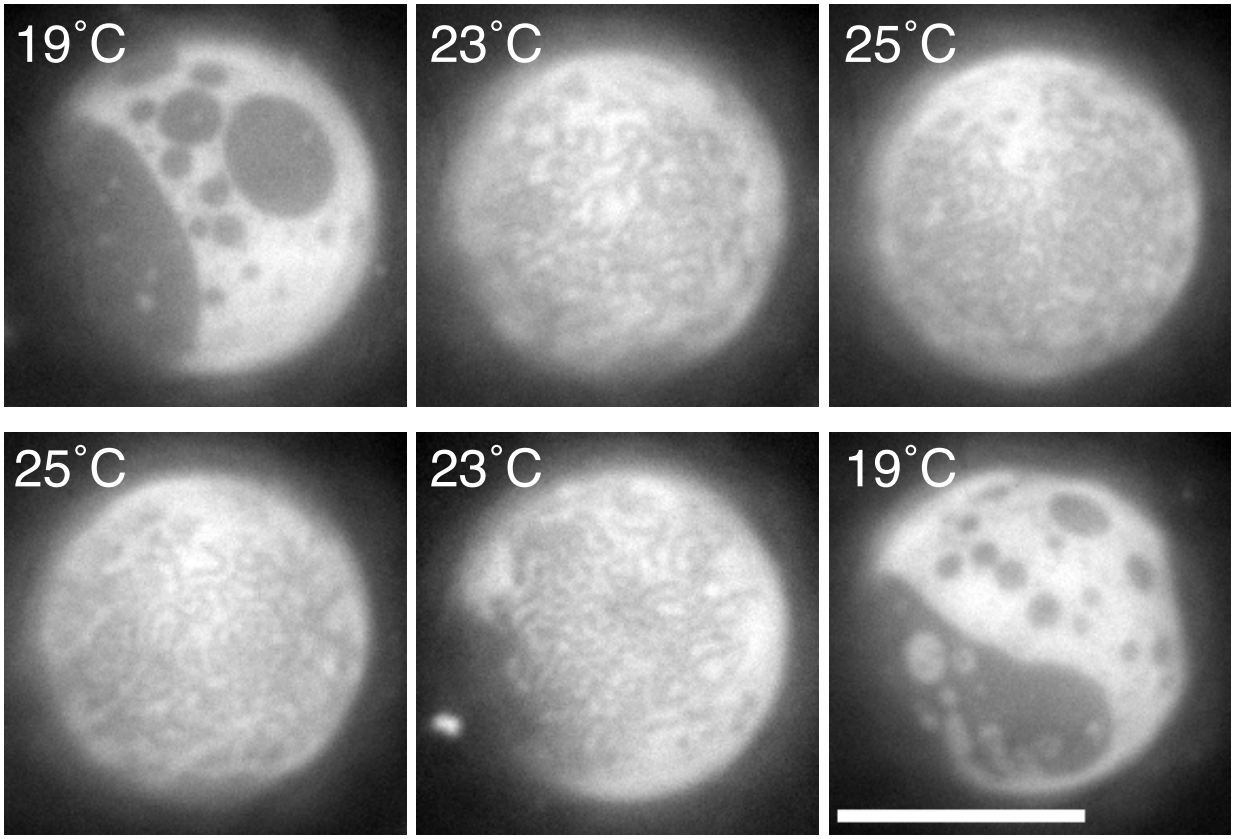

Figure S1: Small domains in a GUV do not exhibit hysteresis when temperature is cycled. The same 35/35/30 DiPhyPC/DPPC/cholesterol vesicle treated with DPPC-loaded HPαCD is shown in all panels. The scale bar is 20 μm.

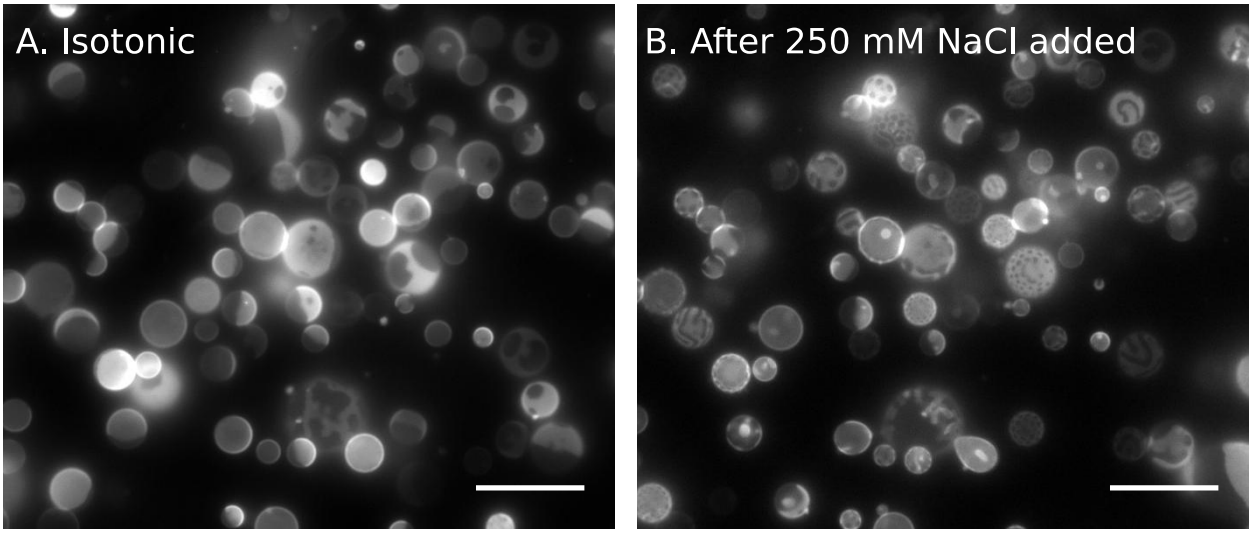

Figure S2: The fraction of GPMVs exhibiting small domains increases in hypertonic solution. **A.** A field of GPMVs derived from MDCK cells in an isotonic solution. **B.** The same field of GPMVs after addition of 250 mM NaCl to the outer solution. Images were collected ∼1 min after the addition of NaCl solution. Scale bars are 20 μm.

Ratio	DiPhyPC/DPPC/cholesterol
1	60/20/20
2	49/26/25
3	28/40/32
4	15/47/38

Table S1: Lipid compositions of the ratios in Fig. 11A-C and Fig. S4.

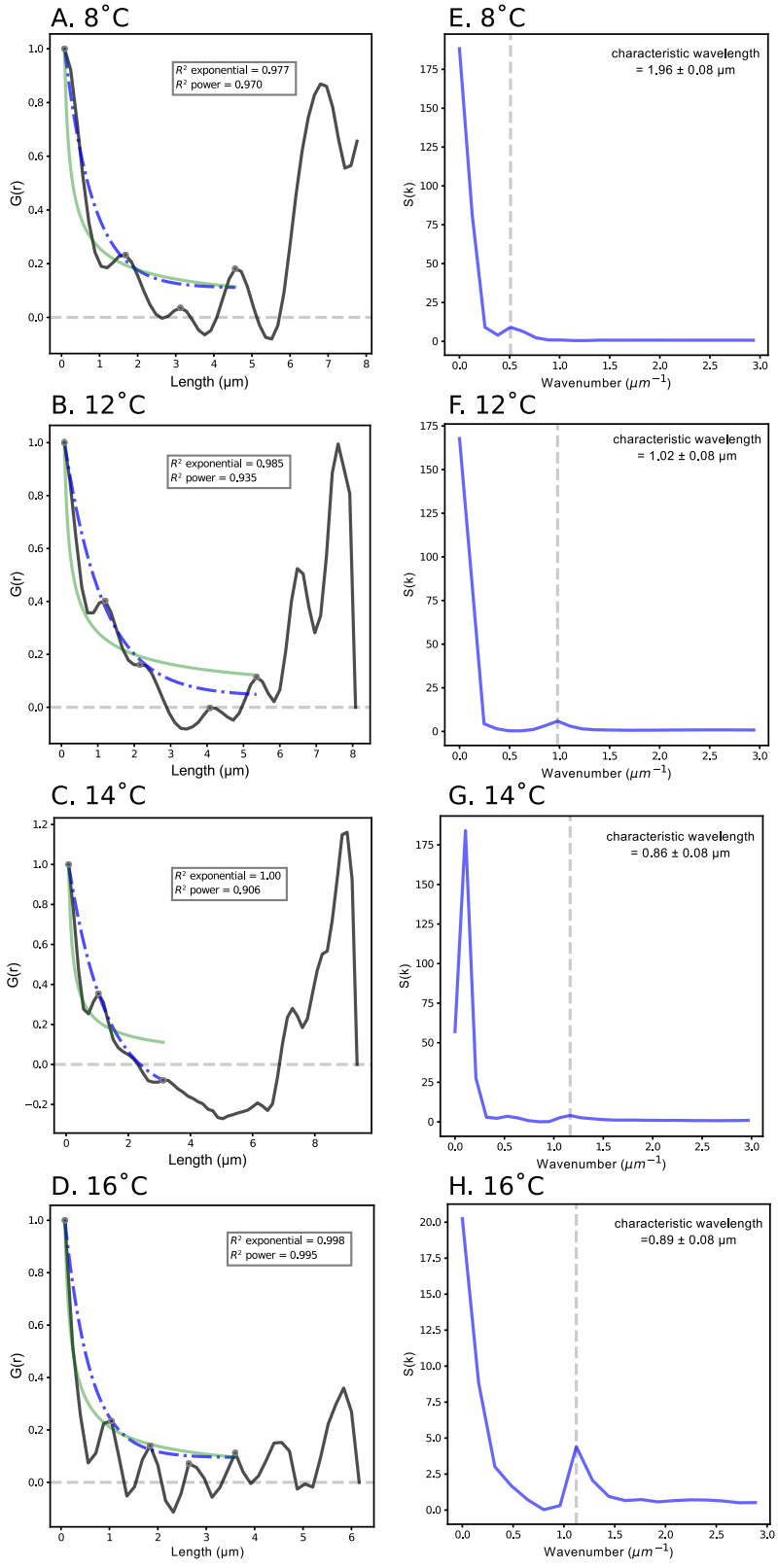

Figure S3: G(r) and S(k) for the points in Fig. 7A. In panels A-D, the grey curves are fits to a power law and the dot-dashed curves are fits to an exponential.

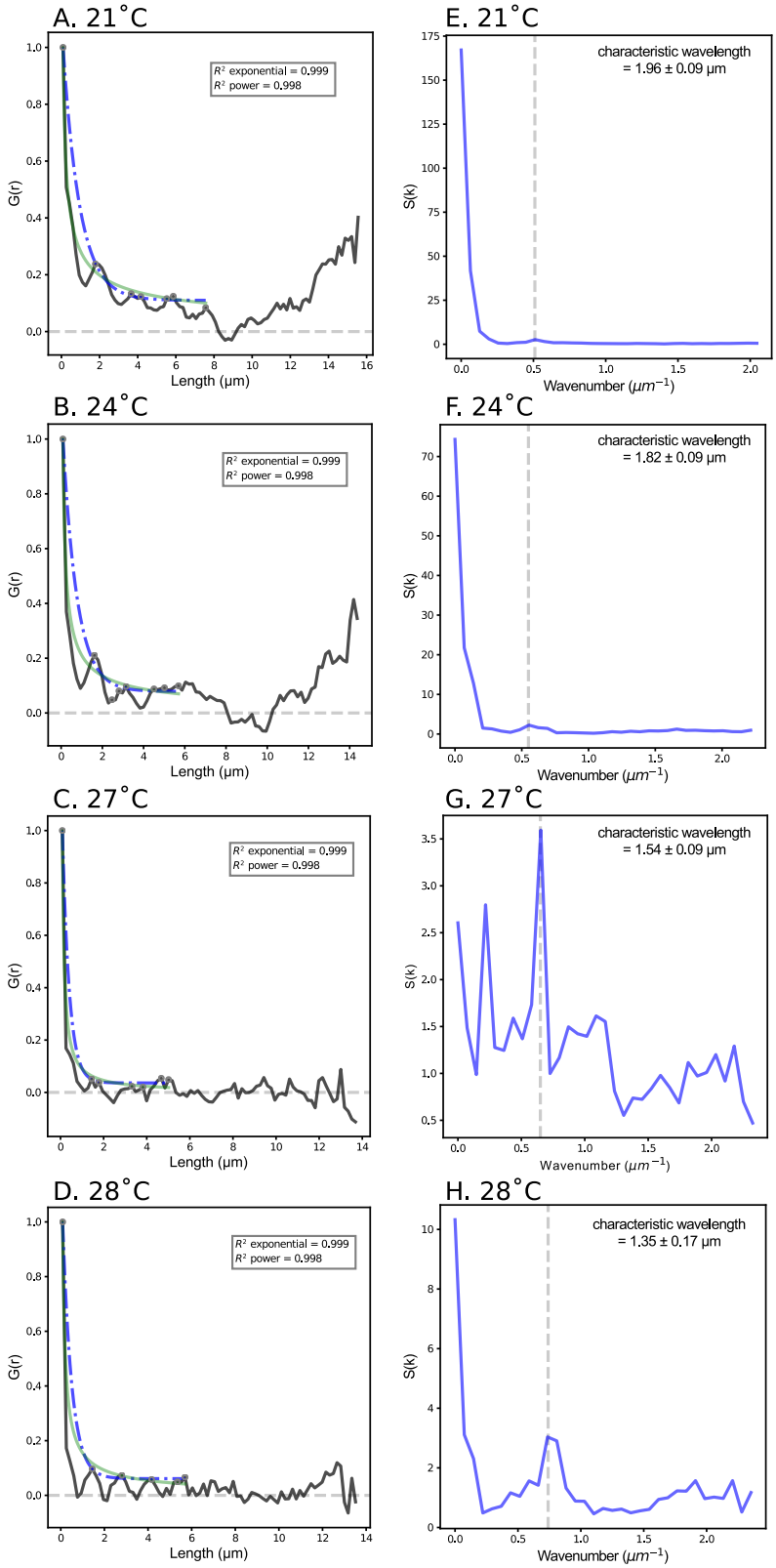

Figure S4: G(r) and S(k) for the points in Fig. 7B. In panels A-D, the grey curves are fits to a power law and the dot-dashed curves are fits to an exponential.

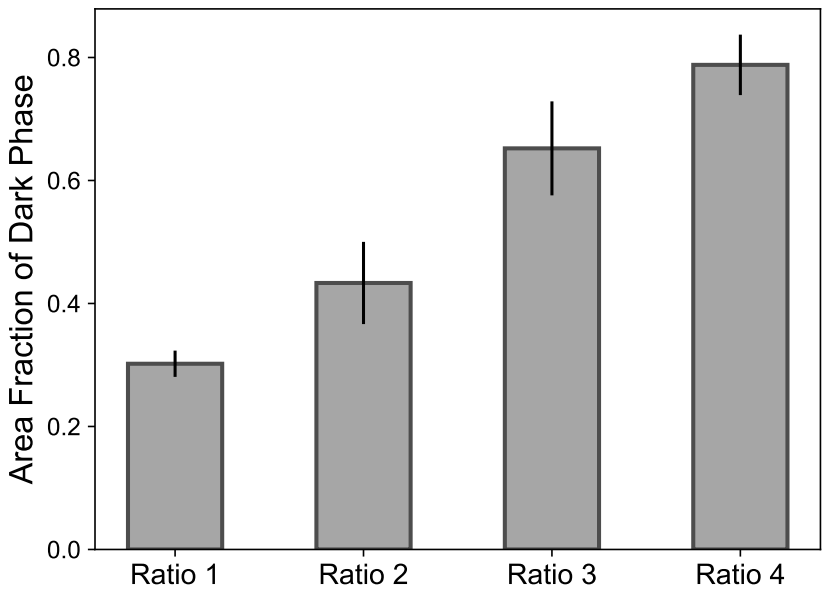

Figure S5: The fraction of area that appears dark increases monotonically from lipid ratio 1 to 4 (Fig. 11A) for a population of vesicles. The bars represent the average area fraction for each lipid ratio. The numbers of vesicles analyzed for lipid ratios $1-4$ were 6, 10, 9, and 6, respectively. The error bars represent the standard deviation.

2 Methods

2.1 Area Fraction

For membranes exhibiting small-scale domains, images of vesicles were collected such that the top, spherical cap of the vesicle lay within the $< 5\mu$m depth of field of the microscope objective; the remainder of the vesicle appeared as a bright ring. Square areas were drawn (edge length 15−60 μm) that included only areas in focus. Pixel intensities were thresholded to yield only white and black areas. Images within the 2D squares were projected onto 3D spherical surfaces using MATLAB code by Sarah Veatch.[S1] The area fraction of the dark regions was the 3D projected area of all black pixels divided by the projected area of all pixels in the image.

For vesicles exhibiting large-scale coexisting Lo-Ld coexistence, images were collected when domains aligned roughly perpendicular to the field of view. Area fractions were then assessed geometrically by evaluating the surface area of spherical caps relative to the surface area of an entire sphere.

2.2 Radial Distribution Function and Structure Factor

We define the two dimensional radial distribution function (RDF) for the image as:

$$g\left(\boldsymbol{r}\right) = \frac{\langle \delta\rho(\boldsymbol{r'}+\boldsymbol{r})\delta\rho(\boldsymbol{r'})\rangle}{\langle \delta\rho(\boldsymbol{r})\rangle^2} \tag{1}$$

where $\delta\rho(\boldsymbol{r})$ is the contrast between the two dimensional image grey value vector and the image average grey value:

$$\delta\rho(\boldsymbol{r}) = \rho(\boldsymbol{r}) - \bar{\rho} \tag{2}$$

and

$$\bar{\rho} = \frac{\int \mathrm{d}\boldsymbol{r}\rho(\boldsymbol{r})}{\int \mathrm{d}\boldsymbol{r}}. \tag{3}$$

To implement the above definition for our pixelated image, we put the grey value vector into matrix representation:

$$\rho(\boldsymbol{r}) = \rho_{i,j} \tag{4}$$

where $\boldsymbol{r}$ is the position vector represented by a matrix element as a pixel at i-th row and j-th column. The average density can be calculated as:

$$\bar{\rho} = \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{\rho_{i,j}}{mn} \tag{5}$$

and the contrast:

$$\delta\rho(\boldsymbol{r}) = \delta\rho_{ij} = \rho_{i,j} - \bar{\rho} \tag{6}$$

Thus, the 2-D $g(\boldsymbol{r})$ becomes:

$$g(\boldsymbol{r}) = g(r_{i,j}) = \frac{\left(\sum_{k,l=1}^{m,n} \delta\rho_{k,l} \cdot \delta\rho_{k+i,l+j} \right)}{\left(\sum_{k,l=1}^{m,n} \delta\rho_{k,l} \right)^2} \tag{7}$$

for all $i \in (1, m)$ and $j \in (1, n)$. The one dimensional RDF, $g(r)$, is the two dimensional function that reports modulation in the direction perpendicular to the stripes and is averaged over the direction parallel to the stripes in the image, assuming translational invariance:

$$g(r) = g(r_i) = \sum_{j=1}^{n} \frac{g(r_{i,j})}{n} \tag{8}$$

Due to the finite size of the image, the calculation yields fewer data points as the separation between two correlated pixels increases and approaches the image size, thus the quality of $g(r)$ degrades as r increases.

We can also calculate the one dimensional structure factor as a Fourier transform of the

1-D $g(r)$ of the image using the generic definition[S2].:

$$S(k) = 1 + \bar{\rho} \int g_{\alpha\beta}(r) \mathrm{e}^{-i\boldsymbol{k} \cdot \boldsymbol{r}} \mathrm{d}\boldsymbol{r} \tag{9}$$

The actual Fouier transform is carried out using the fast Fourier transform function provided in the NumPy v 1.14 Python software package[S3,S4].

SUPPLEMENTARY PYTHON CODE

""" This script will take an 8-bit image and compute a pair correlation function, G(r), of the pixels in the image. First, a pair correlation is calculated along the x-direction. This generates a 2D pair correlation with correlations between particles in X and Y. The code then plots one slice of the G(r) in the x-direction against the radius.

The pair correlation function will be normalized around 0 and this code can also calculate an envelope function of the G(r). The code fits either an exponential or a power law function to the G(r) or envelope and gives error bounds. This version truncates the envelope function after the first several peaks of the G(r)

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.misc import imread
from scipy.optimize import curve_fit


def calcrdf(phi):
    ''' This function calculates the radial distribution function
    of an image composed of an array of pixels. It returns a 2D
    G(r) integrated over the average contrast intensity '''
```

```
    ## Set up the size of the array
    Lx = phi.shape[0]
    Ly = phi.shape[1]

    ## Define field as contrast from average
    ## pixel intensity (can be negative)
    dphi = phi - np.average(phi)

    ## Calculate the average contrast intensity
    diphiavg2 = np.average(dphi**2)

    ## Define the array for a 2D G(r)
    rho2 = np.zeros((Lx+1, Ly+1))

    ## Calculate the 2D G(r)
    for dx in np.arange(Lx):
        for dy in np.arange(Ly):
            phi2 = dphi
            rho2[dx,dy] = np.average(phi2[0:Lx-dx,0:Ly-dy] * phi2[dx:Lx,dy:Ly])

    return rho2/diphiavg2

def exponential(x, a, b, c):
    return a * np.exp(-b * x) + c

def power(x, a, b, c):
    return a * (x**(-b)) + c

def power2(x, a, b):
```

```
    return a * (x**(-b))

if __name__ == "__main__":

    ## Load in the image
    im = imread("GUV_28 deg.tif")
    phi = im

    ## Calculate the 2D G(r) using the RDF function
    dgr_2d = calcrdf(phi)

    ## Slice the 2D G(r) in the x-direction, where
    ## dy=0 to obtain a 1D G(r)
    dgr_1d = dgr_2d[0]

    ## Calculate the numerical derivative of G(r)
    ## to find the maxima to fit to an exponential
    ## or power law curve
    gr_deriv = np.diff(dgr_1d)

    ## Roll the derivative over one to see where
    ## it changes sign
    gr_derivposition = np.array(gr_deriv > 0, dtype=int)
    gr_derivposition2 = np.roll(gr_derivposition, 1)
    np.array(gr_derivposition - gr_derivposition2 == -1, dtype=int)
    maxX = np.array(gr_derivposition-gr_derivposition2 == -1, dtype=int)

    ## Define the array in units of pixels
    ## (will need the max number of pixels)
```

```
r = np.arange(80)

## Define the array in units of microns
## (will need the resolution of the image)
r = r*0.17

## Redefine r to shift by half a pixel length.
## This is due to an error in fitting a power law
## when there are 0's present in r. The
## uncertainty of the self correlation is on the
## order of the size of the pixel (0.17 Âţm)
r = r+0.17/2

## Redefine the bounds of the 1D G(r)
g_r = dgr_1d[:80]

## For plotting, set the maxX at 0 equal to 1
## and find the x values where maxX is 1
## and the y values where x is 1 multiplied
## by the 1D G(r)
maxX[0] = 1
envelopeX = r[maxX==1]
envelopeY = maxX[maxX==1]*g_r[maxX==1]

## Truncate the envelope at ~6 Âţm
## and delete erroneous point at 1 pixel noise
envelopeX = envelopeX[:7]
envelopeY = envelopeY[:7]
print(envelopeX)
```

```
envelopeX = np.delete(envelopeX,4)
envelopeY = np.delete(envelopeY,4)

## Fit the data to an exponential curve or power law
poptE, pcovE = curve_fit(exponential, envelopeX, envelopeY)#,
    p0=[1,1],maxfev=10000)
fitE = exponential(envelopeX, *poptE)
poptP, pcovP = curve_fit(power2, envelopeX, envelopeY, p0 =[1,1])
fitP = power2(envelopeX, *poptP)
#print(popt)
#print(pcov)

# Calculate the error based on a taylor expansion
# and the most weighted parameters (a and c)
perrE = np.sqrt(np.diag(pcovE))
upperE = fitE + (perrE[0]+perrE[2])
lowerE = fitE - (perrE[0]+perrE[2])

perrP = np.sqrt(np.diag(pcovP))
upperP = fitP + (perrP[0])
lowerP = fitP - (perrP[0])

# Calculate the residual and the variance to obtain
# the R-squared value for both fits

residualE = np.sum((envelopeY - fitE)**2)
varianceE = np.sum((envelopeY- envelopeY.mean())**2)
r_squaredE = 1 - (residualE/varianceE)
print(r_squaredE)
```

```
residualP = np.sum((envelopeY - fitP)**2)
varianceP = np.sum((envelopeY - envelopeY.mean())**2)
r_squaredP = 1 - (residualP/varianceP)
print(r_squaredP)

### Plot the G(r)

# Make a plot with space for subplots
fig, axes = plt.subplots(1,1,sharex=True, sharey=True, figsize=(6,4))

# Plot the G(r) and the fit with error bounds
x = np.linspace(envelopeX.min(),envelopeX.max(),200)
axes.axhline(0, color = "black", linestyle ='--', alpha=0.2 )
axes.plot(r, g_r, color = "black", alpha = 0.7, label = 'data')
axes.plot(x, exponential(x,*poptE), color = "blue", alpha = 0.7, linestyle =
    '-.', label = 'exponential fit')
axes.plot(x, power2(x,*poptP), color = "green", alpha = 0.4, label = 'power law
    fit')
axes.plot(envelopeX,envelopeY, marker = "o", linestyle = "None", markersize =
    5, markeredgecolor = "grey", markerfacecolor = "None")
axes.set_ylabel("G(r)", fontname="Arial", fontsize=15)
axes.set_xlabel("Radius (Âţm)", fontname="Arial", fontsize=15)
axes.text(8, 0.85, '$R^2$ exponential = 0.999 \n$R^2$ power = 0.998',
    style='normal',
     bbox={'facecolor':'None', 'alpha':0.5, 'pad':3})

# Save the figure as a PDF
plt.savefig("GUV_28deg_combined_truncate.pdf")
```

```
   plt.show()


## Extra code to compute the S(k)


## Calculate the S(k) by taking a Fourier
   ## Transform of the magnitude squared of the
   ## 1D G(r)
   rho_k = np.fft.fft(dgr_1d)
   s_k = np.real(rho_k)**2 + np.imag(rho_k)**2



   ## Truncate the s_k to cut off the
   ## mirrored part of the plot
   s_k = s_k[0:19]
```

References

(S1) Veatch, S. L.; Soubias, O.; Keller, S. L.; Gawrisch, K. Critical Fluctuations in Domain-Forming Lipid Mixtures. *Proc. Natl. Acad. Sci. U.S.A.* **2013**, *104*, 17650–17655.

(S2) Hansen, J. P.; McDonald, I. R. *Theory of Simple Liquids (Fourth Edition)*, 4th ed.; Academic Press: Oxford, 2013.

(S3) Oliphant, T. E. Python for Scientific Computing. *Comput. Sci. Eng.* **2007**, *9*, 10–20.

(S4) Jones, E.; Oliphant, T.; Peterson, P.; et al., SciPy: Open source scientific tools for Python. 2001–; `http://www.scipy.org/`, [Online; accessed 2018-01-22].