Checking the particle overlaps in LF_DEM simulation data

Minimum Gap (mingap) Calculation

The **mingap** (minimum gap) reported in the data file (`data_*.dat`) and interactions file (`int_*.dat`) represents the fraction of the gap between particles with respect to the average particle radius of both particles.

The minimum gap, (g), is calculated as:

$$g = \frac{2\delta}{a_1 + a_2} = s - 2$$

where:

- $s = \frac{2r}{a_1+a_2}$
- r : distance between particle centers
- $a_1, \ a_2$: radii of particles
- $\delta = r - (a_1 + a_2)$. Gap between particle surfaces (negative for overlapping particles)

We define the **overlap percentage** as:

$$\text{overlap } \% = g \times 100$$

In the interactions file, the minimum gap for each interaction is recorded in every snapshot. In the data file, the minimum gap (g) (or maximum overlap) is reported.

We have verified the following:

1. The reported minimum gap is consistent with the actual overlap of particles calculated using particle positions.
2. The minimum gap value in the data file is the minimum gap (or maximum overlap) between particles in a snapshot.

The data shown here is at:

- **shear rate** $:\sigma = 100 \cdot \sigma_0$
- **friction coefficient** $:\mu = 100$

In [1]:

```
import numpy as np
import glob
import matplotlib.pyplot as plt
import matplotlib
import os
import random
import pandas as pd

def interactionsList(interactionFile):
    '''
    This function reads the interaction file and creates a nested-list,
    each list inside contains the array of all interaction parameters for
    that timestep.

    Input: interactionFile - the location of the interaction data file
    '''
    interFile    = open(interactionFile, 'r')
```

```python
    hashCounter = 0
    temp        = []
    contactList = [] # list with interaction parameters for each element at each time

    fileLines = interFile.readlines()[27:] # skipping the comment lines
    for line in fileLines:
        if not line.split()[0] == '#':
            lineList = [float(value) for value in line.split()]
            temp.append(lineList)
        else:
            hashCounter += 1 # checking if counter reaches 7 (7 lines of comments aft
            if hashCounter == 7:
                contactList.append(np.array(temp))
                temp        = []
                hashCounter = 0
    interFile.close()
    return contactList

def parametersList(ParametersFile):
    '''
    This function reads the parameters file and creates a nested-list,
    each list inside contains the array of all interaction parameters for
    that timestep.

    Input: ParametersFile - the location of the parameters data file
    '''

    parFile = open(ParametersFile, 'r')

    hashCounter = 0
    temp        = []
    parList     = [] # list with parameters parameters for each element at each times

    fileLines = parFile.readlines()[22:] # skipping the comment lines
    for line in fileLines:
        if not line.split()[0] == '#':
            lineList = [float(value) for value in line.split()]
            temp.append(lineList)
        else:
            hashCounter += 1 # checking if counter reaches 7 (7 lines of comments aft
            if hashCounter == 7:
                parList.append(np.array(temp))
                temp        = []
                hashCounter = 0
    parFile.close()
    return parList

############################################################
# Matplotlib rc parameters modification

plt.rcParams.update({
  "figure.max_open_warning" : 0,
  "text.usetex"             : True,
  "text.latex.preamble"     : r"\usepackage{amsmath, bm, type1cm}",  # Added \bm for 
  "figure.autolayout"       : True,
  "font.family"             : "STIXGeneral",
  "mathtext.fontset"        : "stix",
  "font.size"               : 8,
  "xtick.labelsize"         : 8,
  "ytick.labelsize"         : 8,
  "lines.linewidth"         : 1,
  "lines.markersize"        : 5,
})
#plt.rcParams['text.latex.preamble']= r"\usepackage{amsmath}"
matplotlib.rc('text', usetex=True)
```

```
matplotlib.rcParams['text.latex.preamble'] = r'\boldmath'

colors = ['#4a91b5', '#e68139', '#5da258', '#87629b', '#1b9e77']
```

In [18]:
```
# overlap check

#topDir = '/media/rahul/Rahul_2TB/high_bidispersity'
topDir = '/Volumes/Rahul_2TB/high_bidispersity'

phi    = [0.70, 0.72, 0.75, 0.79]
ar     = [1.4, 2.0, 4.0]
runs   = 4
off    = 100

for i, phii in enumerate(phi):
    for j, arj in enumerate(ar):
        delta_g = []
        phir    = '{:.3f}'.format(phii) if len(str(phii).split('.')[1])>2 else '{:.2f
        Dir     = f'{topDir}/NP_1000/phi_{phir}/ar_{ar[j]}'
        if not os.path.exists(Dir):
            continue
        for l in range(runs):
            workDir = f'{topDir}/NP_1000/phi_{phir}/ar_{ar[j]}/Vr_0.5/run_{l+1}'

            intFile = glob.glob(f'{workDir}/int_*.dat')[0]
            intList = interactionsList(intFile)

            parFile = glob.glob(f'{workDir}/par_*.dat')[0]
            parList = parametersList(parFile)

            dataFile = glob.glob(f'{workDir}/data_*.dat')[0]
            data     = np.loadtxt(dataFile)
            time, gamma, shearrate = data[:, 0], data[:, 1], data[:, 2]

            lx = float(open(dataFile).readlines()[3].strip().split()[2])
            ly = float(open(dataFile).readlines()[5].strip().split()[2])

            for k, sampleList in enumerate(intList[off:]):
                for i in range (sampleList.shape[0]):
                    if int(sampleList[i,10]) == 2:
                        index1, index2 = map(int, sampleList[i, :2])
                        snapshot       = off + k
                        r1, x1, y1     = parList[snapshot][index1, 1:4]
                        r2, x2, y2     = parList[snapshot][index2, 1:4]
                        xdist, ydist   = x1-x2, y1-y2
                        rsum           = r1 + r2

                        # Lees-Edwards BC
                        #xdist -= shearrate[snapshot] * ydist * np.round(ydist / ly) 
                        xdist -= gamma[snapshot]%1 * ly * np.round(ydist/ly)

                        # Periodic BC
                        ydist -= ly * np.round(ydist/ly)
                        xdist -= lx * np.round(xdist/lx)

                        dist   = np.sqrt(xdist**2 + ydist**2)
                        g_calc = (2 * dist/(rsum)) - 2 # Calculated dimensionless gap
                        g_sim  = sampleList[i,5] # Gap listed in the int* file
                        delg   = g_calc - g_sim  # difference in calculated and
                        delta_g.append(np.round(delg))

        bold_red = "\033[1;31m"
        reset    = "\033[0m"
        print(f'phi = {phir}, ar = {arj}: {bold_red}{np.abs(np.max(delta_g))}{reset},
        del parList, intList, gamma, time, shearrate, delta_g
```

```
phi = 0.70, ar = 1.4: 0.0, 0.0
phi = 0.70, ar = 2.0: 0.0, 0.0
phi = 0.70, ar = 4.0: 0.0, 0.0
phi = 0.72, ar = 1.4: 0.0, 0.0
phi = 0.72, ar = 2.0: 0.0, 0.0
phi = 0.72, ar = 4.0: 0.0, 0.0
phi = 0.75, ar = 1.4: 0.0, 0.0
phi = 0.75, ar = 2.0: 0.0, 0.0
phi = 0.75, ar = 4.0: 0.0, 0.0
phi = 0.79, ar = 4.0: 0.0, 0.0
```

The results above show the the min and max from the delta_g list which is the difference of calculated g and g reported in the interactions file. We see that, for the cases studied the max and min of the difference is zero, hence verifying our first objective:

1. The reported dimensionaless gap is the fraction of gap between contacting particles and is consistent with the actual overlap calculated from particle position

In [ ]:

In [23]:

```
# 2nd objective

topDir = '/media/rahul/Rahul_2TB/high_bidispersity'
phi    = [0.70, 0.72, 0.75, 0.79]
ar     = [1.4, 2.0, 4.0]
runs   = 4
off    = 100

for i, phii in enumerate(phi):
    for j, arj in enumerate(ar):
        mingap_delta = []
        phir = '{:.3f}'.format(phii) if len(str(phii).split('.')[1])>2 else '{:.2f}'.
        Dir = f'{topDir}/NP_1000/phi_{phir}/ar_{ar[j]}'
        if not os.path.exists(Dir):
            continue
        for l in range(runs):
            workDir = f'{topDir}/NP_1000/phi_{phir}/ar_{ar[j]}/Vr_0.5/run_{l+1}'

            intFile     = glob.glob(f'{workDir}/int_*.dat')[0]
            intList     = interactionsList(intFile)

            dataFile    = glob.glob(f'{workDir}/data_*.dat')[0]
            data        = np.loadtxt(dataFile)
            mingap_data = data[off:, 13]

            mingap_int = []
            for k, sampleList in enumerate(intList[off:]):
                g_sim  = sampleList[:,5] # Gap listed in the int* file
                mingap_int.append(np.min(g_sim))

            mingap_diff = [a-b for a,b in zip(mingap_data, mingap_int)]
            mingap_delta.append(mingap_diff)
        bold_red = "\033[1;31m"
        reset    = "\033[0m"
        print(f'phi = {phir}, ar = {arj}: {bold_red}{np.abs(np.max(mingap_delta))}{re
```

```
phi = 0.70, ar = 1.4: 0.0, 0.0
phi = 0.70, ar = 2.0: 0.0, 0.0
phi = 0.70, ar = 4.0: 0.0, 0.0
phi = 0.72, ar = 1.4: 0.0, 0.0
phi = 0.72, ar = 2.0: 0.0, 0.0
phi = 0.72, ar = 4.0: 0.0, 0.0
phi = 0.75, ar = 1.4: 0.0, 0.0
phi = 0.75, ar = 2.0: 0.0, 0.0
phi = 0.75, ar = 4.0: 0.0, 0.0
phi = 0.79, ar = 4.0: 0.0, 0.0
```

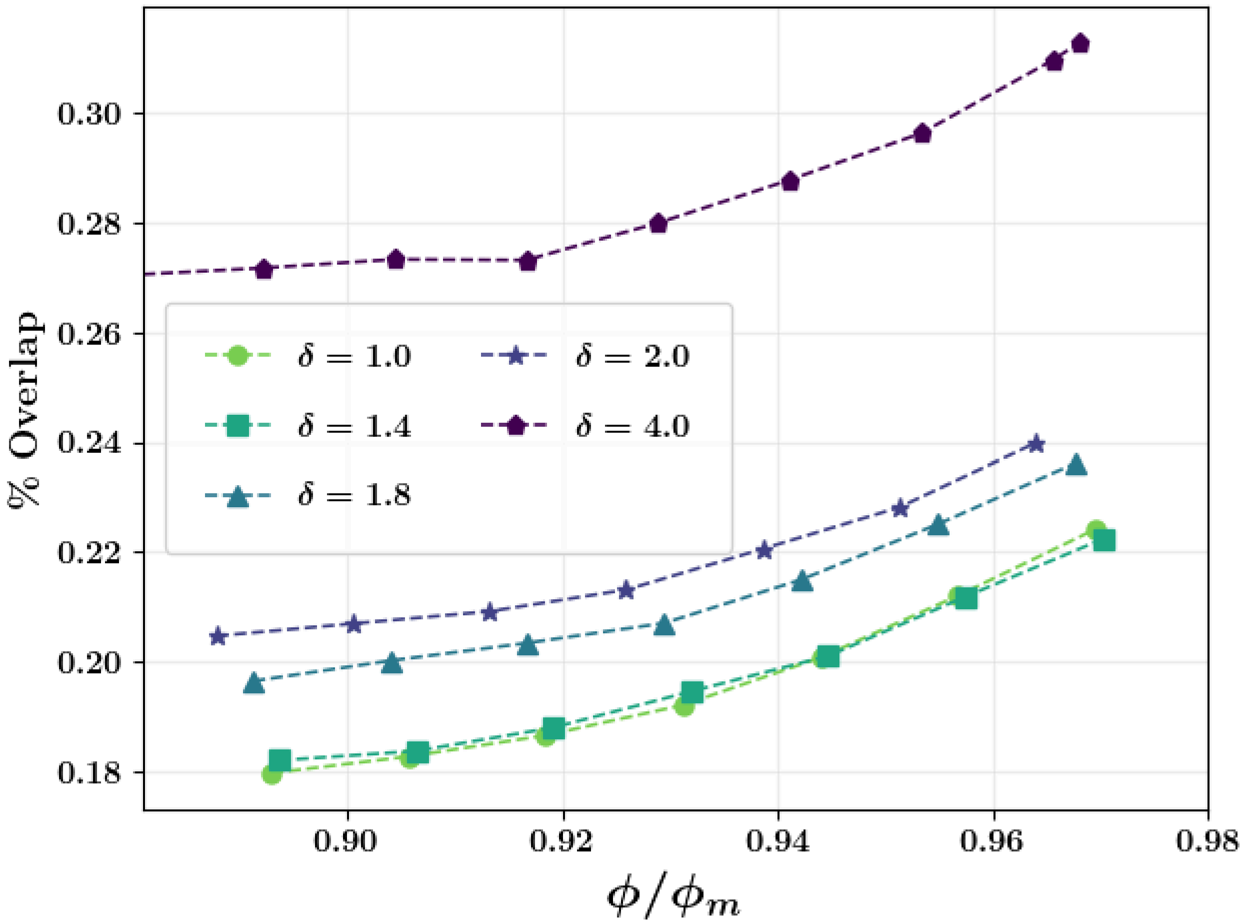
The results above show the the min and max from the mingap_diff list which is the difference of reported g value in the data file and the min of g value in the interactions file for a snaphot. We see that, for the cases studied the max and min of the difference is zero, hence verifying our second objective:

2. The reported dimensionaless gap is in the data file is the minimum gap of the respective snapshot

In [22]:

```
# min gap plot all data

phi = [0.70, 0.71, 0.72, 0.73, 0.74, 0.75, 0.76, 0.77, 0.78, 0.79, 0.792]
ar  = [1.0, 1.4, 1.8, 2.0, 4.0]

npp           = 1000
cmap          = matplotlib.colormaps['viridis_r'] #color scheme
topDir        = '/media/rahul/Rahul_2TB/high_bidispersity'
fig_save_path = "/home/rahul/Dropbox (City College)/CUNY/Research/Bidisperse Project/

phim         = [0.7839319645227314,0.783317545762567,0.7854135245320721,0.78841492538
off          = 100
runs         = 4
line_markers = ['o', 's', '^', '*', 'p']

datax = []
datay = []

for j, arj in enumerate(ar):
    gaplist = []
    for i, phii in enumerate(phi):
        phir = '{:.3f}'.format(phii) if len(str(phii).split('.')[1])>2 else '{:.2f}'.
        Dir = f'{topDir}/NP_1000/phi_{phir}/ar_{ar[j]}'
        if not os.path.exists(Dir):
            continue
        mingap = []
        for l in range(runs):
            workDir     = f'{topDir}/NP_1000/phi_{phir}/ar_{ar[j]}/Vr_0.5/run_{l+1}'
            dataFile    = glob.glob(f'{workDir}/data_*.dat')[0]
            data        = np.loadtxt(dataFile)
            mingap_data = data[off:, 13]
            mingap.append(-np.mean(mingap_data)*100)
        gaplist.append(np.mean(mingap))
    phif     = [x/phim[j] for x, g in zip(phi, gaplist) if not np.isnan(g)]
    gaplistf = [x for x in gaplist if not np.isnan(x)]

    datax.append(phif)
    datay.append(gaplistf)
    plt.plot(phif, gaplistf, linestyle='--',marker = line_markers[j], label=r'$\delta

plt.grid(which='Both', alpha=0.2)
plt.xlabel(r'$\phi/\phi_{m}$', fontsize=18,fontstyle='italic', fontweight='bold')
plt.ylabel(r'\textbf{\% Overlap}', fontsize=14,fontstyle='italic')

plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
```

```
plt.xlim(0.881, 0.98)
plt.legend(loc=(0.02, 0.32), fontsize=12, labelspacing=1.2,borderpad=1.2, ncol=2)
plt.tight_layout()

figsave=False

if figsave:
    figFormat = ".svg"
    plt.savefig(fig_save_path+ "mean_max_gap_" +str(npp)+figFormat, bbox_inches="tigh

plt.show()
```



In [3]:
```
# Effect of normal contact spring constant (kn) on overlap

topDir = '/Users/rahul/Documents/Simulations/test_kn'
phi    = [0.75, 0.79]
ar     = [1.4, 4.0]
runs   = 1
off    = 100
tests  = ['kn1', 'kn2', 'kn3']
kn     = [1884.95, 188.49, 4712.39]

for t in range(len(tests)):
    for i, phii in enumerate(phi):
        for j, arj in enumerate(ar):
            phir   = '{:.3f}'.format(phii) if len(str(phii).split('.')[1])>2 else '{:
            Dir = f'{topDir}/{tests[t]}/NP_1000/phi_{phir}/ar_{ar[j]}'
            if not os.path.exists(Dir):
                    continue
            mingap = []
            for l in range(runs):
                workDir = f'{topDir}/{tests[t]}/NP_1000/phi_{phir}/ar_{ar[j]}/Vr_0.5/
                dataFile     = glob.glob(f'{workDir}/data_*.dat')[0]
                data         = np.loadtxt(dataFile)
                mingap_data = data[off:, 13]
                mingap.append(np.mean(mingap_data))
```

```
        bold_red = "\033[1;31m"
        reset    = "\033[0m"
        overlap  = -np.mean(mingap) * 100
        print(f'phi = {phir}, ar = {arj}, kn = {kn[t]}: {bold_red}{np.round(overl
```

```
phi = 0.75, ar = 1.4, kn = 1884.95: 2.057 %
phi = 0.75, ar = 1.4, kn = 188.49: 16.155 %
phi = 0.79, ar = 4.0, kn = 4712.39: 1.209 %
```

In [17]:

```
# Check if gamma = del_time*shear rate

#workDir  = '/media/rahul/Rahul_2TB/high_bidispersity/NP_1000/phi_0.74/ar_1.8/Vr_0.5/
workDir  = '/Volumes/Rahul_2TB/high_bidispersity/NP_1000/phi_0.74/ar_2.0/Vr_0.5/run_1

dataFile = glob.glob(f'{workDir}/data_*.dat')[0]
data     = np.loadtxt(dataFile)
time, gamma, shearrate = data[:, 0], data[:, 1], data[:, 2]

lx = float(open(dataFile).readlines()[3].strip().split()[2])
ly = float(open(dataFile).readlines()[5].strip().split()[2])

delta_t   = np.diff(time)
cal_gamma = []
calg = 0
for i in range(len(delta_t)):
    cg = delta_t[i]*shearrate[i]
    calg += np.round(cg,2)
    cal_gamma.append(calg)

df = pd.DataFrame({r'gamma': gamma[1:], 'dt * sr': cal_gamma})

print(df)
```

```
      gamma  dt * sr
0      0.01     0.01
1      0.02     0.02
2      0.03     0.03
3      0.04     0.04
4      0.05     0.05
...     ...      ...
3495  34.96    34.96
3496  34.97    34.97
3497  34.98    34.98
3498  34.99    34.99
3499  35.00    35.00

[3500 rows x 2 columns]
```