In [2]:
```
import numpy as np
import scipy.io

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from IPython import display

%matplotlib inline

import importlib
#importlib.reload(driftDiffusionSimulator)
import driftDiffusionSimulator_periodicBC
importlib.reload(driftDiffusionSimulator_periodicBC)
from driftDiffusionSimulator_periodicBC import driftDiffusionSimulator
```

In [3]:
```
def getLengths(Xpos,Ypos):
    Lpos = np.sqrt((Xpos[1:]-Xpos[:-1])**2+(Ypos[1:]-Ypos[:-1])**2)

    return Lpos

def getR_EndToEnd(Xpos,Ypos,L):
    Lpos = np.insert(np.cumsum(np.sqrt((Xpos[1:]-Xpos[:-1])**2+(Ypos[1:]-Ypos
[:-1])**2)),0,0)
    Xout = np.interp(L,Lpos,Xpos)
    Yout = np.interp(L,Lpos,Ypos)
    Rout = np.sqrt(Xout**2+Yout**2)
    return Rout

def p_rGauss(r,t,D):
    return r/4/D/t*np.exp(-r**2/4/D/t)
```

Analysis of sim. files

Load in files and initialize functions

In [4]:
```
Emins = list(np.ones(20)*.9)
Emins+= list(np.ones(20)*.9)
Emins+= list(np.ones(20)*.9)
Emins+= list(np.ones(20)*.9)
Emins+= list(np.ones(20)*.95)
Emins+= list(np.ones(20)*.975)
Emins+= list(np.ones(20)*.9875)
Emins+= list(np.ones(20)*.99375)

N_particles = list(np.ones(20)*100)
N_particles+= list(np.ones(20)*200)
N_particles+= list(np.ones(20)*400)
N_particles+= list(np.ones(100)*800)
N_particles = [int(x) for x in N_particles]

mainPath = "E:\Arthur's Stanford Simulations\Hydrodynamics\Periodic Boundary C
onditions"
fnames = []
for i in range(20):
    fnames.append(mainPath+'/periodicBC_0p9E_100N/periodicBC_0p9E_100N_'+'%03d
' % i +'.npz')
for i in range(20):
    fnames.append(mainPath+'/periodicBC_0p9E_200N/periodicBC_0p9E_200N_'+'%03d
' % i +'.npz')
for i in range(20):
    fnames.append(mainPath+'/periodicBC_0p9E_400N/periodicBC_0p9E_400N_'+'%03d
' % i +'.npz')
for i in range(20):
    fnames.append(mainPath+'/periodicBC_0p9E_800N/periodicBC_0p9E_800N_'+'%03d
' % i +'.npz')
for i in range(20):
    fnames.append(mainPath+'/periodicBC_0p95E_800N/periodicBC_0p95E_800N_'+'%0
3d' % i +'.npz')
for i in range(20):
    fnames.append(mainPath+'/periodicBC_0p975E_800N/periodicBC_0p975E_800N_'+'
%03d' % i +'.npz')
for i in range(20):
    fnames.append(mainPath+'/periodicBC_0p9875E_800N/periodicBC_0p9875E_800N_'
+'%03d' % i +'.npz')
for i in range(20):
    fnames.append(mainPath+'/periodicBC_0p99375E_800N/periodicBC_0p99375E_800N
_'+'%03d' % i +'.npz')


rDiffs = []
XposMultSims = []
YposMultSims = []
Lfinals = np.zeros(len(fnames))
Lees = np.zeros(len(fnames))
diffusionConsts = np.zeros(len(fnames))


N_pathlength = 10
sample_diff_const = np.arange(0,5,.01)
```

In [5]:
```
for i,fname in enumerate(fnames):

    if N_particles[i] == 400:
        mat = np.load(fname)
        Lfinal = mat['timeCount']/100.
        Lfinals[i] = Lfinal

        traces = mat['traces']
        vX = mat['vX']
        vY = mat['vY']

        rDiff = np.zeros(N_particles[i])
        XposInSim = []
        YposInSim = []
        for j in range(N_particles[i]):

            Nscatter = round(len(traces[j]))
            Xpos = np.zeros(Nscatter+2)
            Ypos = np.zeros(Nscatter+2)
            Ltot = 0
            for k in range(Nscatter):
                P=np.array(traces[j][k][0:2])
                L=traces[j][k][2]
                E=np.sqrt(np.sum(P**2))
                vhat=P/E
                Xpos[k+1]=Xpos[k]+vhat[0]*L
                Ypos[k+1]=Ypos[k]+vhat[1]*L
                Ltot+=L
            Xpos[-1] = Xpos[-2] + vX[j]*(Lfinal-Ltot)
            Ypos[-1] = Ypos[-2] + vY[j]*(Lfinal-Ltot)

            #XposInSim and YposInSim are the sets of coordinates
            XposInSim.append(Xpos)
            YposInSim.append(Ypos)

            rDiff[j] = np.sqrt(Xpos[-1]**2+Ypos[-1]**2)
        rDiffs.append(rDiff)
        XposMultSims.append(XposInSim)
        YposMultSims.append(YposInSim)



        RvsL = np.zeros((N_particles[i],N_pathlength))
        total_travel = np.floor(Lfinals[i])
        pathlengths = np.linspace(total_travel/2.,total_travel,N_pathlength+1)
[1:]
        for k in range(N_particles[i]):
            for j,path_len in enumerate(pathlengths):
                RvsL[k,j] = getR_EndToEnd(XposInSim[k],YposInSim[k],path_len)

        mu_tests = sample_diff_const
        logProb = np.zeros(np.shape(mu_tests))
        #np.log(p_rGauss(rDiff,mu))

        for k,mu in enumerate(mu_tests):
            #logProbs = np.log(p_rGauss(rDiff,mu))
```

```
            logProb[k]=0
            for j,path_len in enumerate(pathlengths):
                logProb[k]+=np.sum(np.log(p_rGauss(RvsL[:,j],path_len,mu)))

        logProb[np.where(np.isnan(logProb))]=-np.inf
        logProb-=np.max(logProb)
        plt.plot(mu_tests,np.exp(logProb))
        plt.show()
        display.clear_output(wait=True);
        diffusionConsts[i] = mu_tests[np.argmax(logProb)]

        L = np.array([])
        for k in range(N_particles[i]):
            L = np.append(L,getLengths(XposInSim[k],YposInSim[k]))
        Lees[i] = np.mean(np.array(L))

        fname_save = fname[:-4]+'_analysis.npz'
        np.savez(fname_save, RvsL = RvsL, Lee = Lees[i], diffusionConst = diff
usionConsts[i],
                 mu_tests = mu_tests, prob = np.exp(logProb),total_travel = tot
al_travel,pathlengths = pathlengths,
                 N_particles = N_particles)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-5-8783a4c42ed1> in <module>()
      2
      3     if N_particles[i] == 400:
----> 4         mat = np.load(fname)
      5         Lfinal = mat['timeCount']/100.
      6         Lfinals[i] = Lfinal

C:\ProgramData\Anaconda3\lib\site-packages\numpy\lib\npyio.py in load(file, m
map_mode, allow_pickle, fix_imports, encoding)
    370     own_fid = False
    371     if isinstance(file, basestring):
--> 372         fid = open(file, "rb")
    373         own_fid = True
    374     elif is_pathlib_path(file):

FileNotFoundError: [Errno 2] No such file or directory: "E:\\Arthur's Stanfor
d Simulations\\Hydrodynamics\\Periodic Boundary Conditions/periodicBC_0p9E_40
0N/periodicBC_0p9E_400N_000.npz"
```

Calculate diffusion const

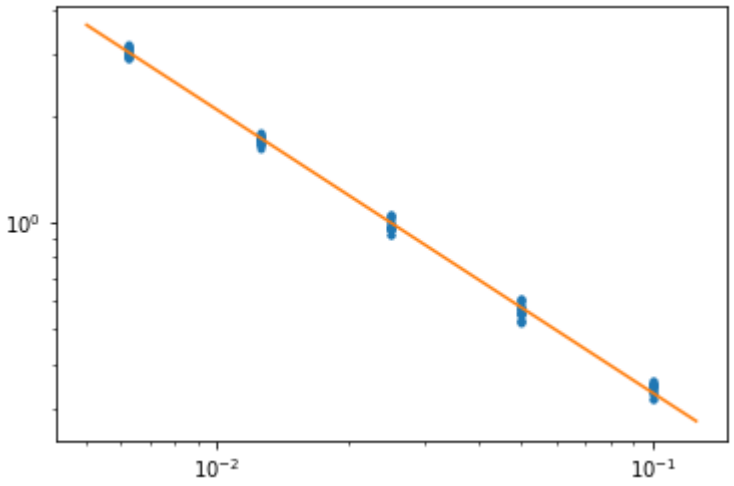
Emperical fit of diffusion const. vs. temperature

In [142]:
```
diffusionConsts = np.zeros(np.shape(fnames))
Lees = np.zeros(np.shape(fnames))

for i,fname in enumerate(fnames):
    mat = np.load(fname[:-4]+'_analysis.npz')
    diffusionConsts[i] = mat['diffusionConst']
    Lees[i] = mat['Lee']
```

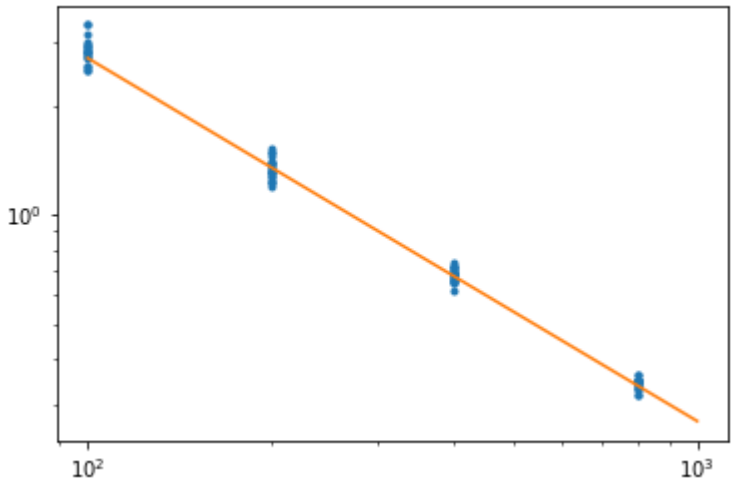
In [196]:
```
r1 = [int(i) for i,x in enumerate(N_particles) if x == 800]
Emins1 = [Emins[i] for i in r1]
diffusionConsts1 = [diffusionConsts[i] for i in r1]

plt.loglog(1-np.array(Emins1),np.array(diffusionConsts1),'.')
x = np.logspace(-2.3,-.9,1000)
plt.plot(x,42*x**-.8/800)
plt.show()k
```



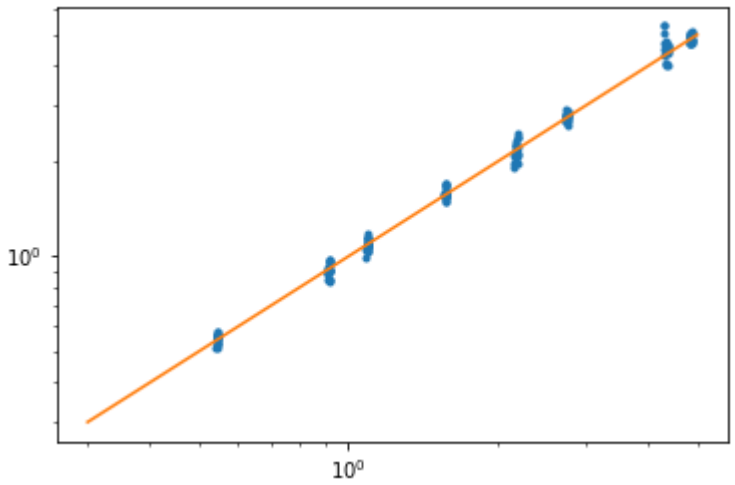
In [191]:
```
r1 = [int(i) for i,x in enumerate(Emins) if x == 0.9]
N_particles1 = [N_particles[i] for i in r1]
diffusionConsts1 = [diffusionConsts[i] for i in r1]


plt.loglog(N_particles1,np.array(diffusionConsts1),'.')
x = np.logspace(2,3,1000)
plt.plot(x,270*x**-1)
plt.show()
```



ℓ_{ee} vs. D/v_f

In [192]:
```
plt.loglog(Lees,np.array(diffusionConsts)*1.6,'.')
plt.plot([.3,5],[.3,5])
plt.show()
```



Based on these data, our current understanding is that the diffusion constant goes as:

$$D = \frac{42}{N(\Delta E)^{0.8}}$$

Where N is the number of particles (in a $3\mu m$ square and ΔE is the effective temperature $(1 - E_{min})$. This yields:

$$D = \frac{4.7}{n(\Delta E)^{0.8}}$$

where n is the particle density in μm^{-2}. We also find that the $\ell_{ee} \approx 1.6D/v_f$. So, this means that the mean free path goes as:

$$\ell_{ee} = \frac{7.5}{n(\Delta E)^{0.8}}$$

in units of microns.

The variables n and ΔE are input parameters to the simulation, however in a physical system, these are linked. Both are approximately proportional to temperature. So, we could select them independently, or tie them together based on a simulation condition. Alternatively, we could try to mimic the behavior for a particular quasiparticle density at a particular temperature. For now, we'll go with the former.

Based on particular simulation conditions ($N = 800$ and $\Delta E = 0.1$) we get $\ell_{ee} = 0.53\ \mu m$. This scattering length is achieved at a density of $10^{12}\ cm^{-2}$ at roughly $120\ K$.

So, we could assert that $N = 6.67T$ and $\Delta E = 8.33 \times 10^{-4} T$.

One variable not yet tested is the "overlap radius". Based on scaling arguments, I believe the scattering length is inversely related to this variable, as long as the system is still "dilute" in the sense that the thickness in the direction of travel is small compared to ℓ_{ee}. In other words, the volume traversed by a given particle in a given time is proportional to its width, and not its thickness.

Currently, we have an "overlap radius" as $r_{over} = 0.05\ \mu m$. This means the scattering length should actually go as:

$$\ell_{ee} = \frac{150}{n r_{over}(\Delta E)^{0.8}}$$

In [198]:
```
150/(800/42.1)/.05/(.1)**.8
```

Out[198]:
```
996.1239075981051
```

Theoretical mean free path for bulk scattering

In [260]:
```
p_scatter = 0.001
N_test = int(1E8)

A = np.random.rand(N_test)

scatters = np.array([i for i,x in enumerate(A) if x<p_scatter])

print(np.mean(scatters[1:]-scatters[:-1]))
```

```
996.4385475417024
```

This shows that the mean free path with a probability $p_{scatter}$ of scattering per timestep is:

$$\ell_{scatter} = \frac{v_f \Delta t}{p_{scatter}}$$

where Δt is the timestep and v_f is the Fermi velocity.

Given that the timestep we traditionally use is 0.01 and $v_f = 1$, This means $\ell_{scatter} = 1\ \mu m$ corresponds to $p_{scatter} = 0.01$

In [ ]:
```

```