通过trackpy库实现粒子追踪及数据处理示例

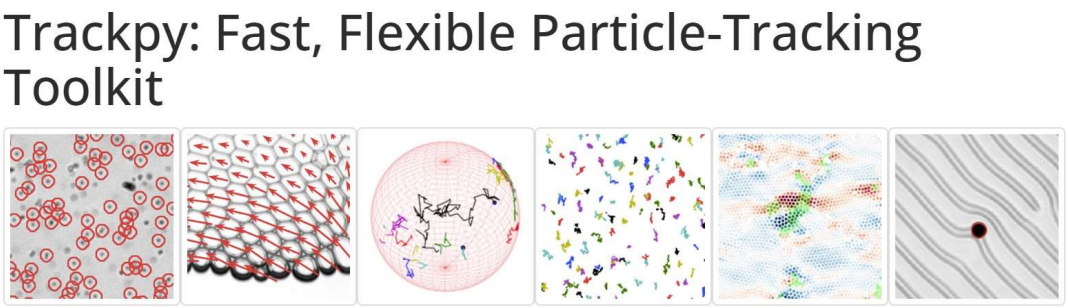
credit by 谢昀城，张世范

(python verion 3.8.5)

trackpy 是一个强大的开源 Python 库，能够方便的实现粒子追踪和轨迹分析。对于粒子msd的计算也提供了封装好的函数，

以下将介绍如何使用trackpy对实验中拍摄得到的颗粒布朗运动的连续帧序列进行粒子追踪和相关的数据处理

对于其更多功能感兴趣可以参考trackpy的官网:http://soft-matter.github.io/trackpy/dev/index.html



可以通过运行以下代码安装trackpy库

In [ ]:
```
# ! pip install trackpy
```

In [ ]:
```
import trackpy as tp #导入trackpy

#导入必要的数据处理库
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
from PIL import Image

#使matplotlib能够正常显示中文和负号
plt.rcParams['font.family'] = 'SimHei'
plt.rcParams['axes.unicode_minus'] = False
```

安装图片编号读取example_T=18.1C°文件夹下的图片将其转化为灰度图并以数组的形式储存在frames中

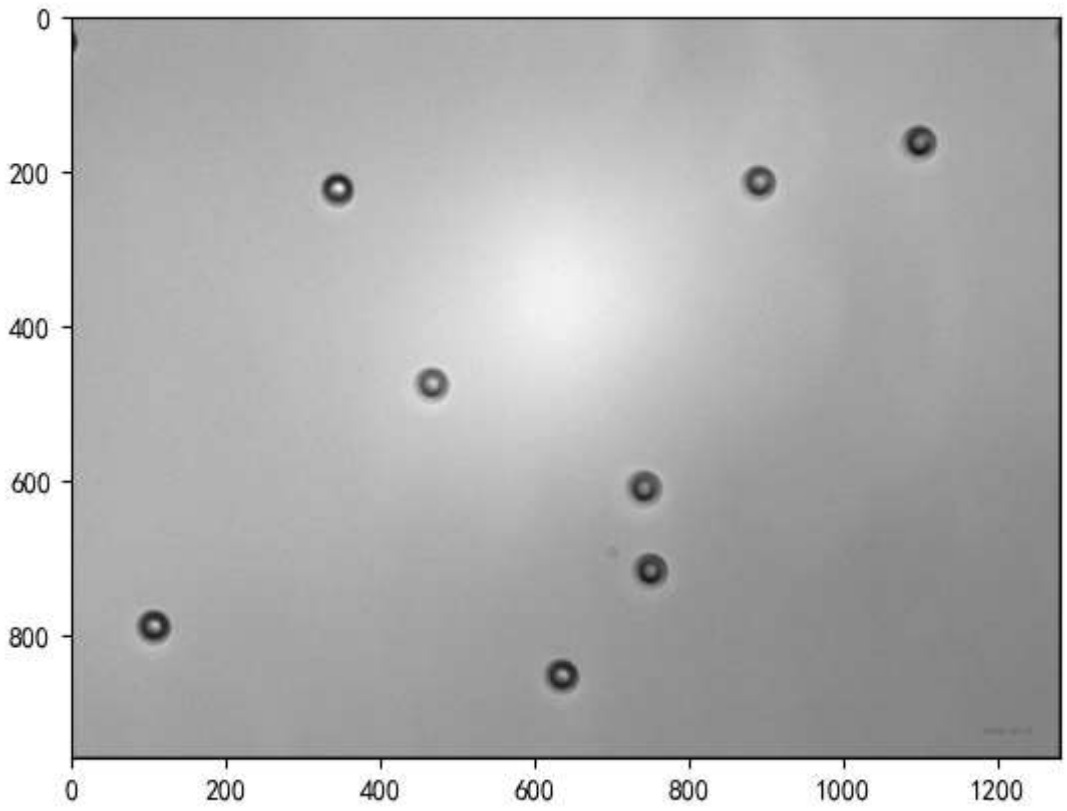
In [ ]:
```
cd=os.getcwd()
image_folder="example_T=18.1C° \\"
image_paths=os.listdir(image_folder) #获取图片路径列表

image_paths_id=[[image_path,int(image_path[6:-4:])] for image_path in image_paths]
image_paths_sorted=(np.array(sorted(image_paths_id,key=lambda x:x[1])))[:,0]  #将图

#读取图片
```

```
frames=[]
for fp in image_paths_sorted:
    image = Image.open(image_folder+fp)


    gray_image = image.convert("L") # 转换为灰度图

    frames.append(np.array(gray_image))#以数组形式存入frames

    image.close()
    gray_image.close()# 关闭图像
```

用matplotlib.pyplot.imshow显示其中一张图片

In [ ]:
```
i=0
plt.imshow(frames[i],cmap="gray") #cmap="gray"表示以灰度图显示
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x1cc366abd90>
```



根据单位像素的长度和颗粒直径估算识别物尺寸的大小,*注意：trackpy要求尺寸须为奇数*

In [ ]:
```
micron_per_pixel = 0.098
feature_diameter = 3.3 # um
radius = int(np.round(feature_diameter/2.0/micron_per_pixel))
if radius % 2 == 0:
    radius += 1
print('Using a radius of {:d} px'.format(radius))
```

```
Using a radius of 17 px
```

这里给出了对于3.3μm颗粒，其半径约为17个像素

根据估计得到的粒子特征对于第1帧，用trackpy.locate函数识别图片中的特征点

(注意该函数识别的是相对于周围较亮的部分，故若实际颗粒较暗则可设置invert参数为True将图片反转)

In [ ]:
```
i=0
inv=False
f1 = tp.locate(frames[i],radius,invert=inv)
f1 #返回dataframe对象，包含粒子坐标，总亮度(mass),高斯轮廓的半径size),偏心率(ecc)等特
```

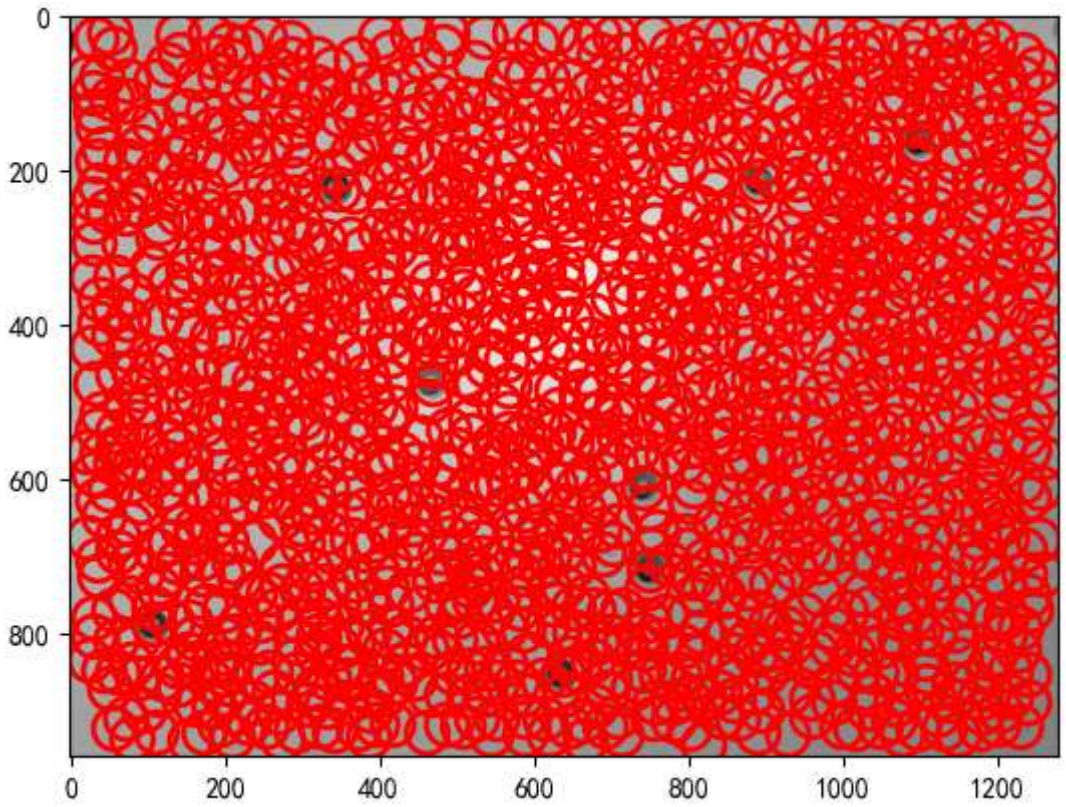
Out[ ]:

	y	x	mass	size	ecc	signal	raw_mass	ep
0	19.994681	486.884752	198.965406	4.047151	0.192277	5.644409	30435.0	NaN
1	21.060523	626.522696	256.467819	5.010855	0.373769	4.586082	30959.0	NaN
2	20.404537	951.950851	186.618262	3.731286	0.168896	4.586082	29950.0	NaN
3	21.494253	50.440066	214.840305	4.885050	0.211548	4.233307	28899.0	NaN
4	26.368217	753.550388	182.032180	5.246631	0.148232	3.527755	31199.0	NaN
...	...	...	...	...	...	...	...	...
1275	935.116059	79.260459	261.406677	4.614271	0.168380	5.291633	28258.0	NaN
1276	933.623552	207.370656	182.737731	4.841304	0.410487	3.174980	27668.0	NaN
1277	936.966981	666.613208	149.576830	5.600708	0.172334	3.527755	26186.0	NaN
1278	937.618128	153.338782	237.417940	5.186134	0.086442	3.527755	27871.0	NaN
1279	941.975155	818.041925	227.187449	5.048369	0.125480	3.174980	24911.0	NaN

1280 rows × 8 columns

用tp.annotate在图中标出特征点，可以看到除了目标颗粒外，识别到了许多错误的特征，他们可能是转瞬即逝的亮度峰值，实际上不是粒子

In [ ]:
```
style = {'markersize':radius, 'markeredgewidth': 2, 'markeredgecolor': 'r', 'markerf
tp.annotate(f1, frames[i])
```



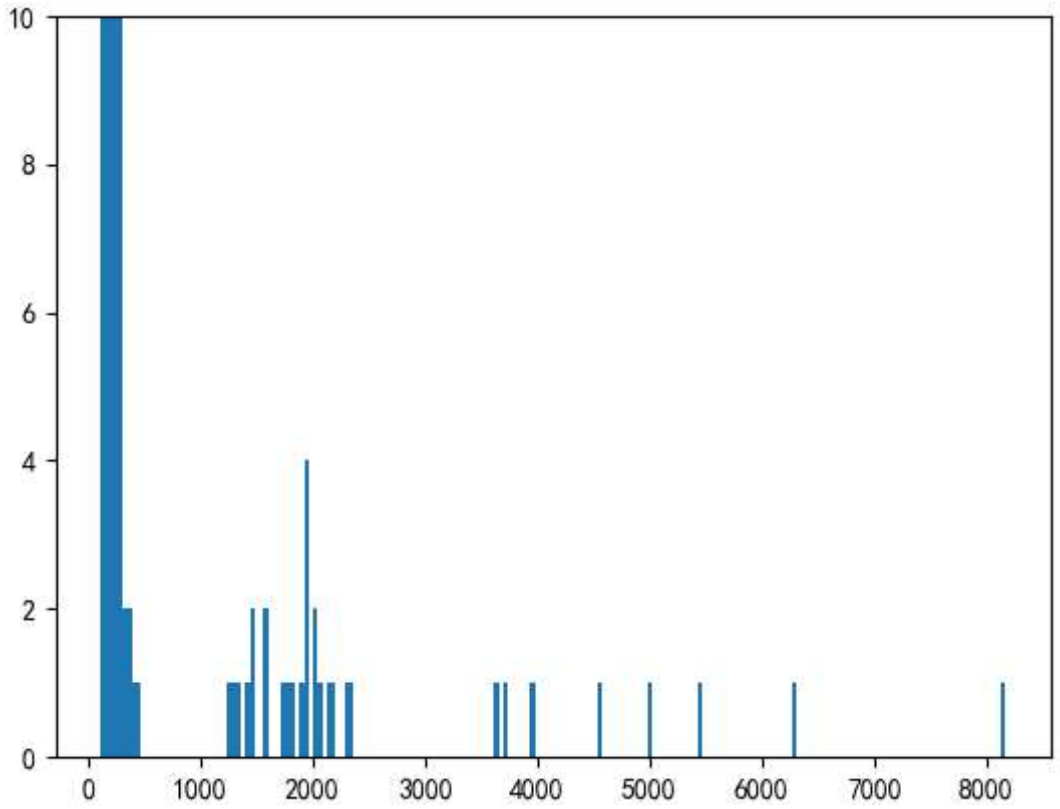
Out[ ]: <AxesSubplot: >

有许多方法可以区分真实粒子和虚假粒子。最重要的方法是查看总亮度（"mass"）。

绘出特征亮度的分布，可以看到亮度1000以上颗粒较少，因此我们设置tp.locate中minmass参数为1000

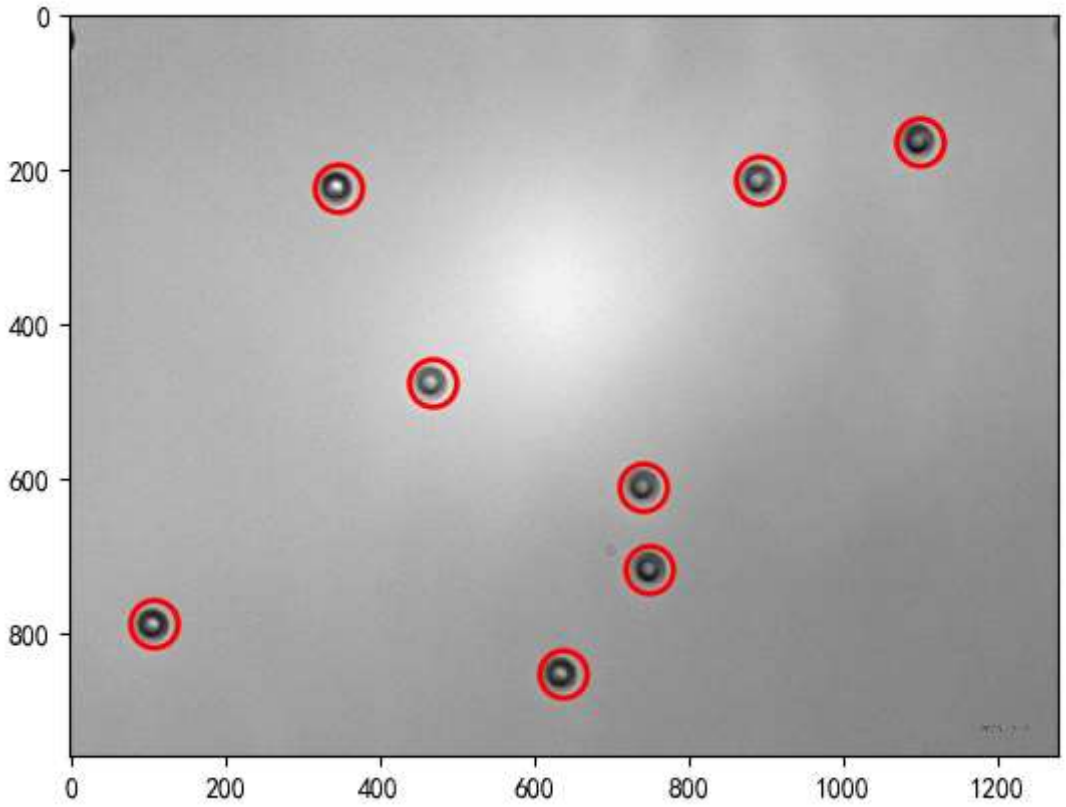
In [ ]:
```
hist=plt.hist(f1['mass'], bins=200)
plt.ylim(0,10)
```

Out[ ]: (0.0, 10.0)



可以看到在修改参数后对颗粒识别效果较好

In [ ]:
```
minm=1000
inv=False
sep=2*radius+1
f1 = tp.locate(frames[i],radius,invert=inv,minmass=minm,separation=sep)
#separation表示两个特征之间的最小间隔，设置其为2*radius+1 可以有效防止将一个过亮粒子
style = {'markersize':radius, 'markeredgewidth': 2, 'markeredgecolor': 'r', 'markerf
tp.annotate(f1, frames[i],plot_style=style)
```



Out[ ]: <AxesSubplot: >

以上我们完成了对一张图片中的粒子识别，接着可以根据上面调试好的参数使用tp.batch对所有图片序列进行识别

In [ ]:
```
f = tp.batch(frames, radius,invert=inv,minmass=minm,separation=sep)
```

```
Frame 998: 9 features
```

In [ ]:
```
f#返回了每一帧的粒子位置
```

Out[ ]:

	y	x	mass	size	ecc	signal	raw_mass	ep
0	162.997827	1098.703420	3733.776339	4.913923	0.070297	34.572003	19392.0	NaN
1	213.552719	891.730582	4573.734905	4.810252	0.059248	42.685841	24964.0	NaN
2	222.891149	346.425554	8167.106579	4.585167	0.134541	89.957763	31219.0	NaN
3	475.366260	467.939180	4999.887760	4.740340	0.094783	49.388576	29219.0	NaN
4	610.978707	741.986096	3628.296452	4.775138	0.065530	33.866452	20519.0	NaN
...	...	...	...	...	...	...	...	...
8846	459.435592	541.444593	8851.216413	5.121719	0.082100	73.355891	39809.0	0.029499
8847	519.096001	1120.672313	6449.796962	4.676492	0.176367	67.834479	24926.0	0.062600
8848	592.314903	454.568663	10307.291403	5.092340	0.064848	83.609940	38304.0	0.031165
8849	676.877453	285.807732	11415.123105	5.107286	0.054417	93.469603	37966.0	0.031566
8850	870.145547	17.614239	12231.503178	4.986692	0.032295	100.568560	36721.0	0.033134

8851 rows × 9 columns

得到每一帧的粒子位置后，用tp.link()将其链接成轨迹

其中search_range参数表示相邻两帧间同一个粒子最大的移动距离

memory表示对于颗粒的记忆,如memory=2,则粒子在消失2帧后若在附近位置出现会被认为是同一粒子，可以用于图片序列中存在若干不清晰图片的情况

In [ ]:
```
t = tp.link(f,search_range=8*radius, memory=0)
```

```
Frame 998: 9 trajectories present.
```

In [ ]:
```
t#这是链接后的轨迹数据，仍然按照frame顺序排列，其中particle列表明轨迹的序号
```

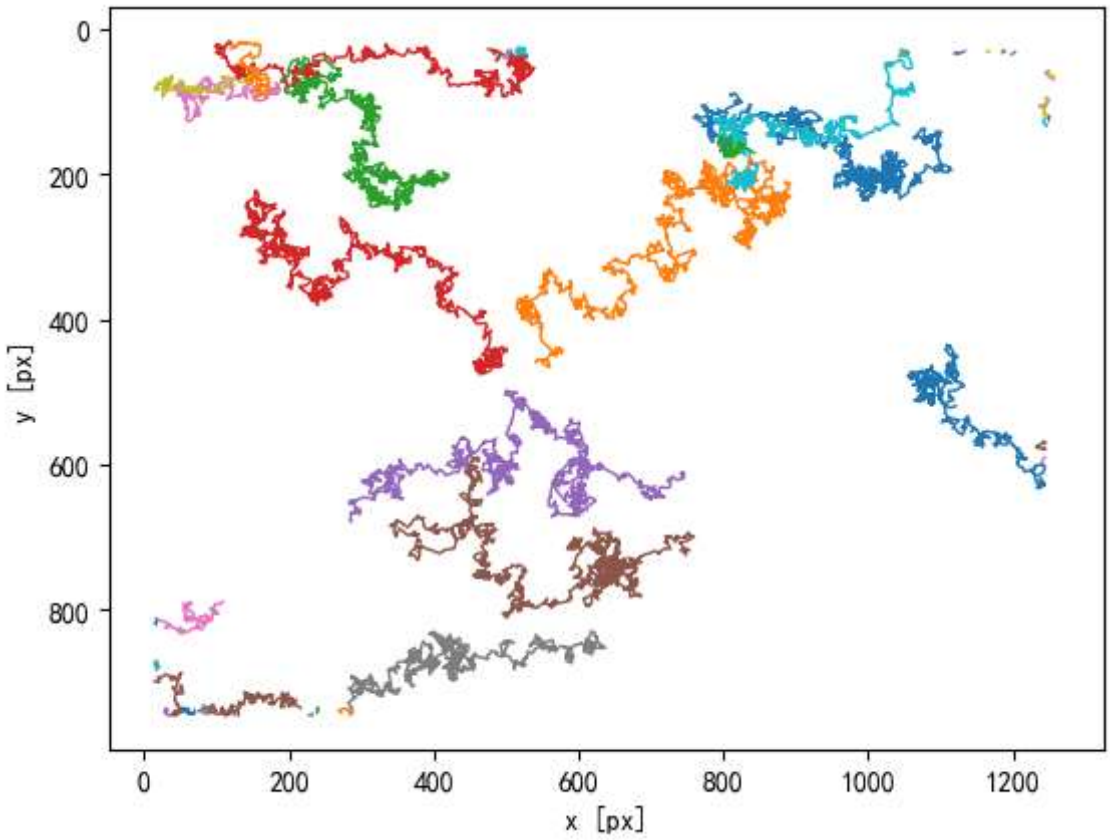
Out[ ]:

	y	x	mass	size	ecc	signal	raw_mass	ep
0	162.997827	1098.703420	3733.776339	4.913923	0.070297	34.572003	19392.0	NaN
1	213.552719	891.730582	4573.734905	4.810252	0.059248	42.685841	24964.0	NaN
2	222.891149	346.425554	8167.106579	4.585167	0.134541	89.957763	31219.0	NaN
3	475.366260	467.939180	4999.887760	4.740340	0.094783	49.388576	29219.0	NaN
4	610.978707	741.986096	3628.296452	4.775138	0.065530	33.866452	20519.0	NaN
...	...	...	...	...	...	...	...	...
8845	280.033111	190.618070	11851.314584	5.147579	0.011437	94.652762	38472.0	0.030970
8846	459.435592	541.444593	8851.216413	5.121719	0.082100	73.355891	39809.0	0.029499
8847	519.096001	1120.672313	6449.796962	4.676492	0.176367	67.834479	24926.0	0.062600
8848	592.314903	454.568663	10307.291403	5.092340	0.064848	83.609940	38304.0	0.031165
8850	870.145547	17.614239	12231.503178	4.986692	0.032295	100.568560	36721.0	0.033134

8851 rows × 10 columns

用tp.plot_traj可以直接绘制粒子轨迹

In [ ]:
```
tp.plot_traj(t)
```



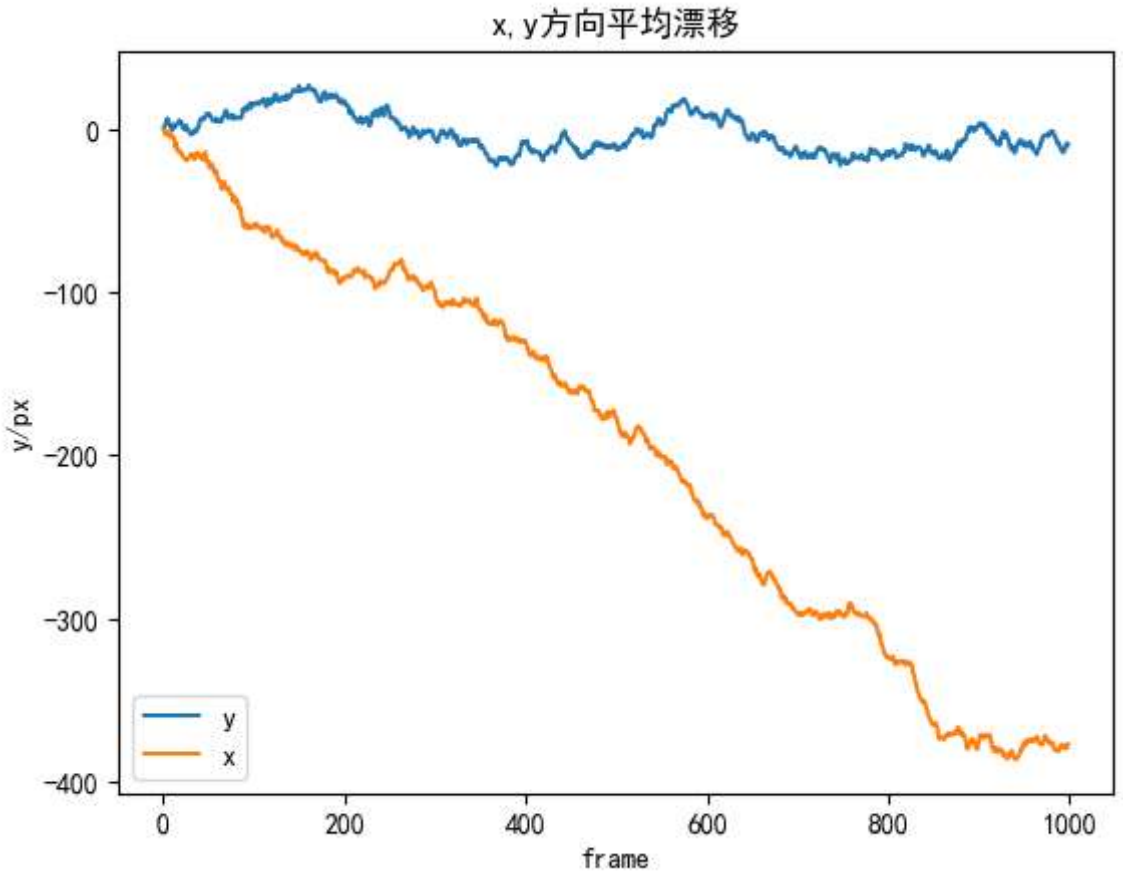
Out[ ]: <AxesSubplot: xlabel='x [px]', ylabel='y [px]'>

In [ ]:
```
t1 = tp.filter_stubs(t,10) #长度过短的粒子可能不可靠，可以用tp.filter_stubs函数进行过

# Compare the number of particles in the unfiltered and filtered data.
print('Before:', t['particle'].nunique())
print('After:', t1['particle'].nunique())
```

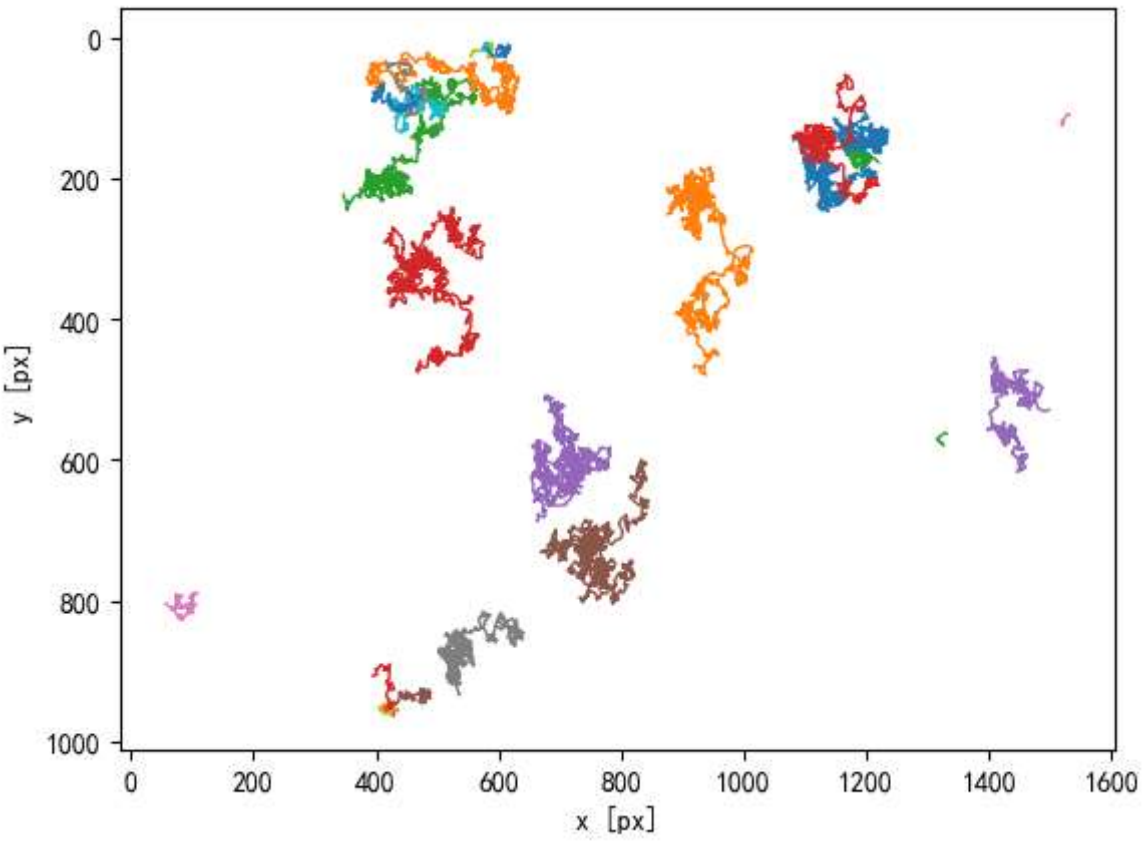
```
Before: 62
After: 24
```

由于毛细管中的液体可能存在整体流动，故粒子轨迹需要减去其整体漂移

In [ ]:
```
d = tp.compute_drift(t1)  #tp.compute_drift计算得到粒子的平均漂移
d.plot()

plt.title("x,y方向平均漂移")
plt.ylabel("y/px")
plt.show()#漂移速度
```



In [ ]:
```
tm = tp.subtract_drift(t1.copy(), d)
#减去整体漂移运动后，我们再次绘制轨迹。我们观察到很好的随机漫步。

ax = tp.plot_traj(tm)

plt.show()
```



trackpy提供了直接计算单条轨迹的平均平方位移，即MSD(Mean Square Displacement）或多条轨迹平均MSD的函数:imsd,emsd

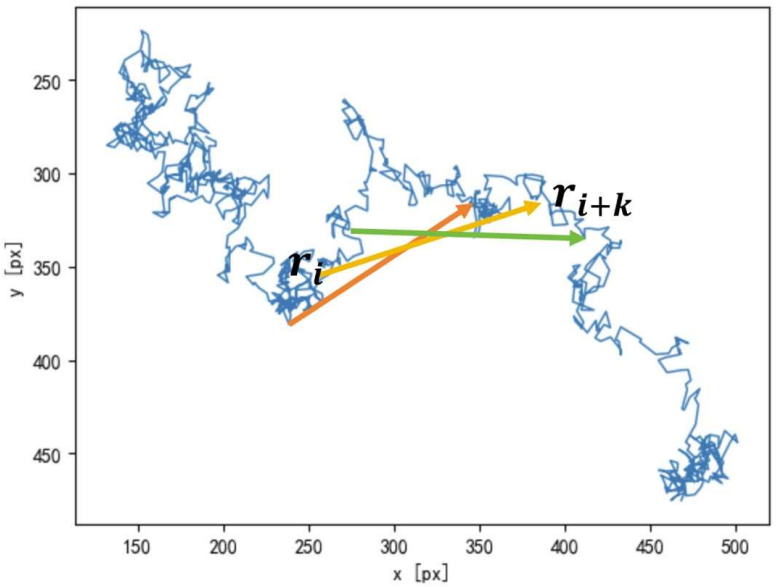
imsd(traj, mpp, fps, max_lagtime=100, statistic='msd', pos_columns=None):

imsd函数会计算每条轨迹的平均平方位移

对于单条轨迹，其为:

$$\rho_k = MSD(k\Delta t) =< r(k\Delta t) - r(0)^2 >= \frac{1}{N-k-1}\sum_{i=0}^{N-k}(r_{k+i} - r_i)^2$$

其中N为轨迹总长度，k为延迟帧数，Δt为相邻图片拍摄时间间隔



其中traj为上一步我们得到的包含轨迹的DateFrame

mmp为单位像素的长度，在本示例中mmp=micron_per_pixel=$0.098\mu m$

fps为拍摄帧率，这里为1/s

max_lagtime为其计算的最大的延迟时间，默认为100帧，可以将其设置为一个很大的值，那么就会计算到轨迹允许的最长时间

statistic为你需要计算的物理量，默认即为msd，也可以设置为'msd','<x> ','<y> ', '<x^2>', '<y^2>'

pos_columns为粒子位置数据在DateFrame中的位置，如果你是用trackpy追踪得到的轨迹，那么直接使用默认值['x', 'y']即可

感兴趣的同学可以尝试自己实现msd的计算

注:对于不存在间断的轨迹，trackpy会通过更为高效的傅里叶变换的方法进行(复杂度O(nlogn))

详情见:https://stackoverflow.com/questions/34222272/computing-mean-square-displacement-using-python-and-fft

V. Calandrini,E. Pellegrini,nMoldyn - Interfacing spectroscopic experiments, molecular dynamics simulations and models for time correlation functions.https://doi.org/10.1051/sfn/201112010

In [ ]:
```
fps=1 #1/s
im=tp.imsd(tm,micron_per_pixel,fps,max_lagtime=1000)
im
```

Out[ ]:

lag time [s]	0	1	2	3	4	5	6	7
1.0	0.292961	0.321070	0.288798	0.292882	0.282114	0.294387	0.321156	0.270766
2.0	0.599310	0.683300	0.585055	0.600883	0.582712	0.623116	0.694252	0.540752
3.0	0.898890	1.047820	0.860649	0.909136	0.891030	0.933761	1.097401	0.811550
4.0	1.213790	1.411972	1.150468	1.207731	1.199031	1.250175	1.449746	1.081788
5.0	1.518240	1.772681	1.429537	1.520787	1.505659	1.593321	1.761763	1.337727
...	...	...	...	...	...	...	...	...
994.0	120.491509	637.885317	NaN	408.877565	111.070077	191.761270	NaN	NaN
995.0	118.249058	639.210735	NaN	417.092732	113.003212	192.086953	NaN	NaN
996.0	114.787886	635.489502	NaN	419.295246	110.958581	190.111323	NaN	NaN
997.0	111.318483	635.164907	NaN	419.804993	112.980896	192.345420	NaN	NaN
998.0	111.822323	633.614749	NaN	426.618399	115.060857	192.719194	NaN	NaN

998 rows × 24 columns

emsd(traj, mpp, fps, max_lagtime=100, detail=False, pos_columns=None)

emsd返回的为所有轨迹的平均msd，参数的输入与imsd相同 其中detail参数若设置为True，则其除平均msd外还会返回一些其它信息

In [ ]:
```
em=tp.emsd(tm,micron_per_pixel,fps,max_lagtime=1000,detail=True)
```

```
c:\Users\xyc\AppData\Local\Programs\Python\Python38\lib\site-packages\trackpy\motion.py:235: FutureWarning: Using the level keyword in DataFrame and Series aggregations is deprecated and will be removed in a future version. Use groupby instead. df.median(level=1) should use df.groupby(level=1).median().
  results = msds.mul(msds['N'], axis=0).mean(level=1)  # weighted average
c:\Users\xyc\AppData\Local\Programs\Python\Python38\lib\site-packages\trackpy\motion.py:236: FutureWarning: Using the level keyword in DataFrame and Series aggregations is deprecated and will be removed in a future version. Use groupby instead. df.median(level=1) should use df.groupby(level=1).median().
  results = results.div(msds['N'].mean(level=1), axis=0)  # weights normalized
c:\Users\xyc\AppData\Local\Programs\Python\Python38\lib\site-packages\trackpy\motion.py:242: FutureWarning: Using the level keyword in DataFrame and Series aggregations is deprecated and will be removed in a future version. Use groupby instead. df.sum(level=1) should use df.groupby(level=1).sum().
  results['N'] = msds['N'].sum(level=1)
```

In [ ]:
```
em
```

Out[ ]:

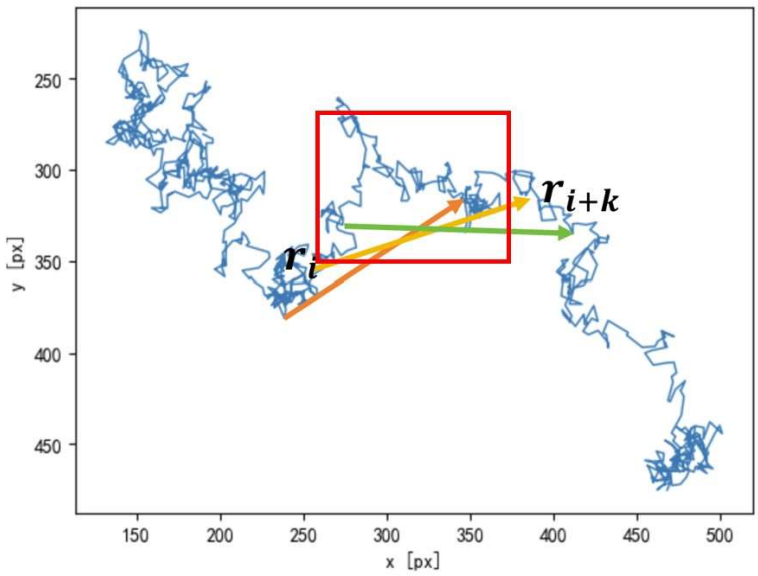
frame	<x>	<y>	<x^2>	<y^2>	msd	N	lagt
1	-2.214069e-17	-1.224370e-18	0.145016	0.146208	0.291224	8705.000000	1.0
2	3.475240e-04	-1.887452e-04	0.292924	0.305242	0.598166	5792.711636	2.0
3	6.402885e-04	-3.460451e-04	0.442991	0.463306	0.906297	4107.992130	3.0
4	6.273510e-04	-4.170217e-04	0.588998	0.622973	1.211971	3147.438063	4.0
5	5.438132e-04	-7.501159e-04	0.736231	0.778765	1.514996	2540.732366	5.0
...	...	...	...	...	...	...	...
994	4.671618e+00	-4.584121e-01	67.348224	226.668923	294.017148	5.016128	994.0
995	4.621239e+00	-5.289331e-01	66.607559	229.320979	295.928538	5.012582	995.0
996	4.495094e+00	-5.687176e-01	63.781629	230.346879	294.128508	5.008934	996.0
997	4.349180e+00	-5.867879e-01	61.279476	233.043464	294.322940	5.005018	997.0
998	4.237967e+00	-6.562909e-01	60.032323	235.934781	295.967104	5.000000	998.0

998 rows × 7 columns

lagt为延迟时间，N为有效数据量

对于某一条轨迹 $N = \begin{cases} \frac{6k(N_0-k)^2}{(4(N_0-k)k^2+2(N_0-k)+k-k^3)}, k \geq N_0/2 \\ [1+[(N_0-k)^3-4(N_0-k)^2k+4k-N_0+k)]]^{-1}, k \leq N_0/2 \end{cases}$ (N_0为轨迹长度，k为延迟帧数)

需要注意的时，N并不为N-k，这是由于在相邻两步位移独立，且k>1时，$(r(k\Delta t)-r(0))^2$间并不独立，故有效数据量小于N-k



部分 (r(k Δt)-r(0))^2有共同部分

对于不同长度的轨迹，计算得到的单条轨迹也应该以此为权重进行加权平均

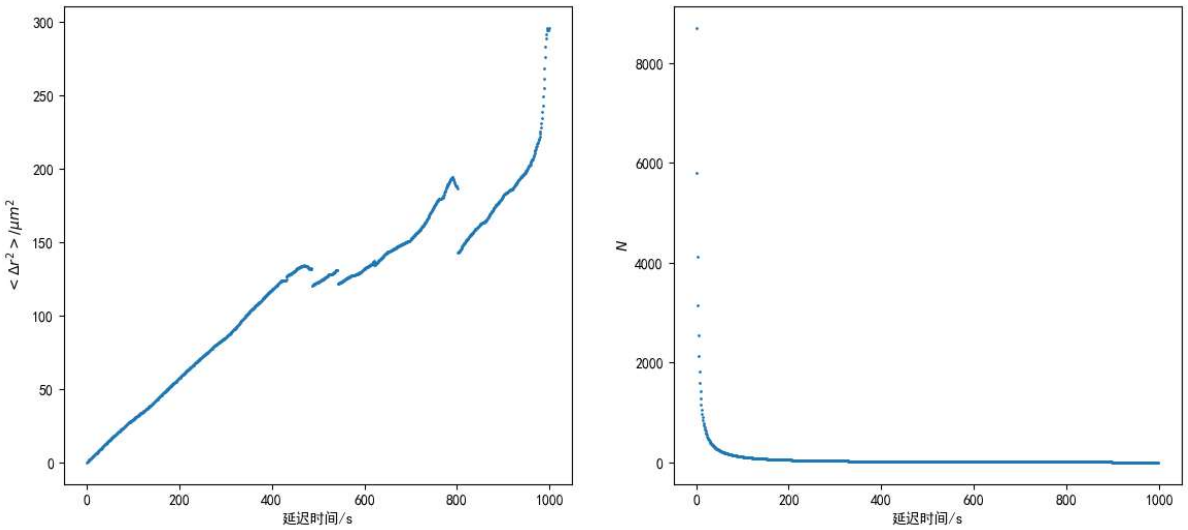
此外，还有MSD的相对偏差的期望值 $\frac{\Delta\rho_k\Delta\rho_k}{<\rho_k>^2} = \frac{1}{N}$,这可以帮助我们判断得到的统计量的有效性

以上详细推导见文献:Qian, Hong, Michael P. Sheetz, and Elliot L. Elson. "Single particle tracking. Analysis of diffusion and flow in two-dimensional systems."Biophysical journal 60.4 (1991): 910.中的Appendix B

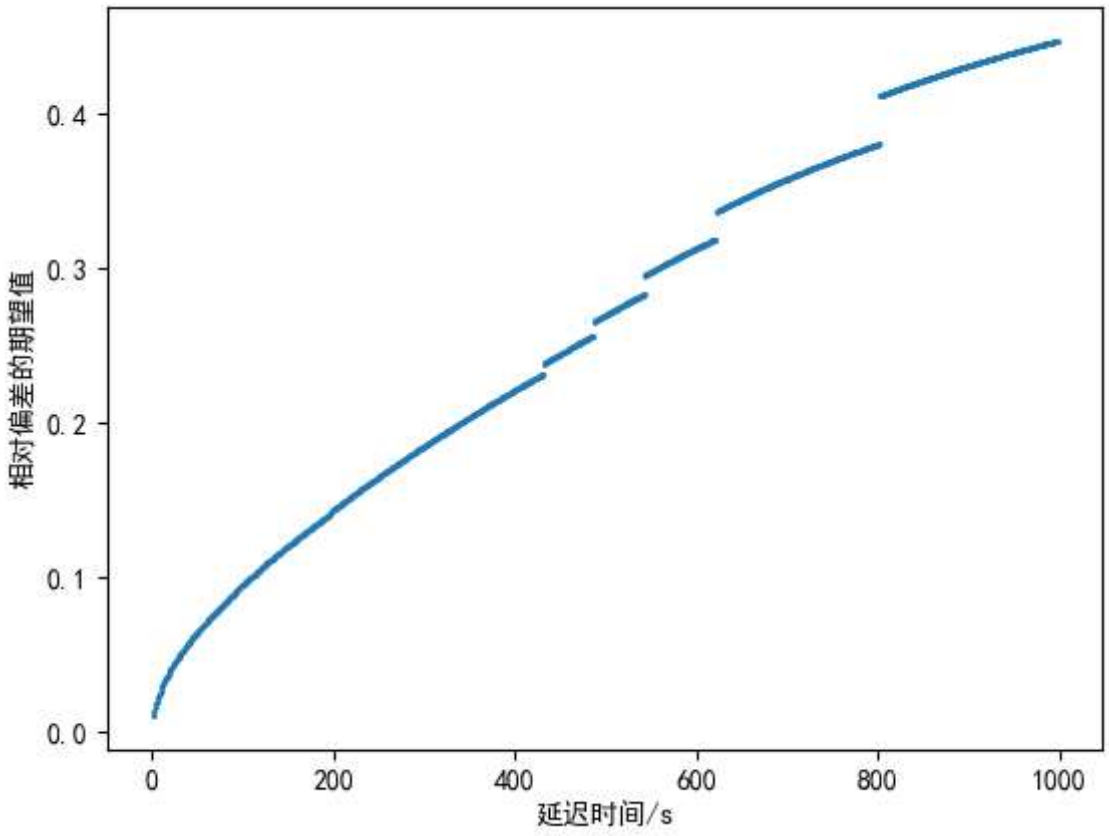
可以看到，emsd在推迟时间短时保持较好的线性，随着推迟时间的延长，有效数据的下降，明显偏离线性

In [ ]:
```
plt.figure(figsize=[14,6])
plt.subplot(121)
plt.scatter(em["lagt"],em["msd"],s=1)
plt.ylabel(r"$<\Delta r^2>/\mu m^2$")
plt.xlabel(r"延迟时间/s")
plt.subplot(122)
plt.scatter(em["lagt"],em["N"],s=1)
plt.ylabel(r"$N$")
plt.xlabel("延迟时间/s")
```

Out[ ]:
```
Text(0.5, 0, '延迟时间/s')
```



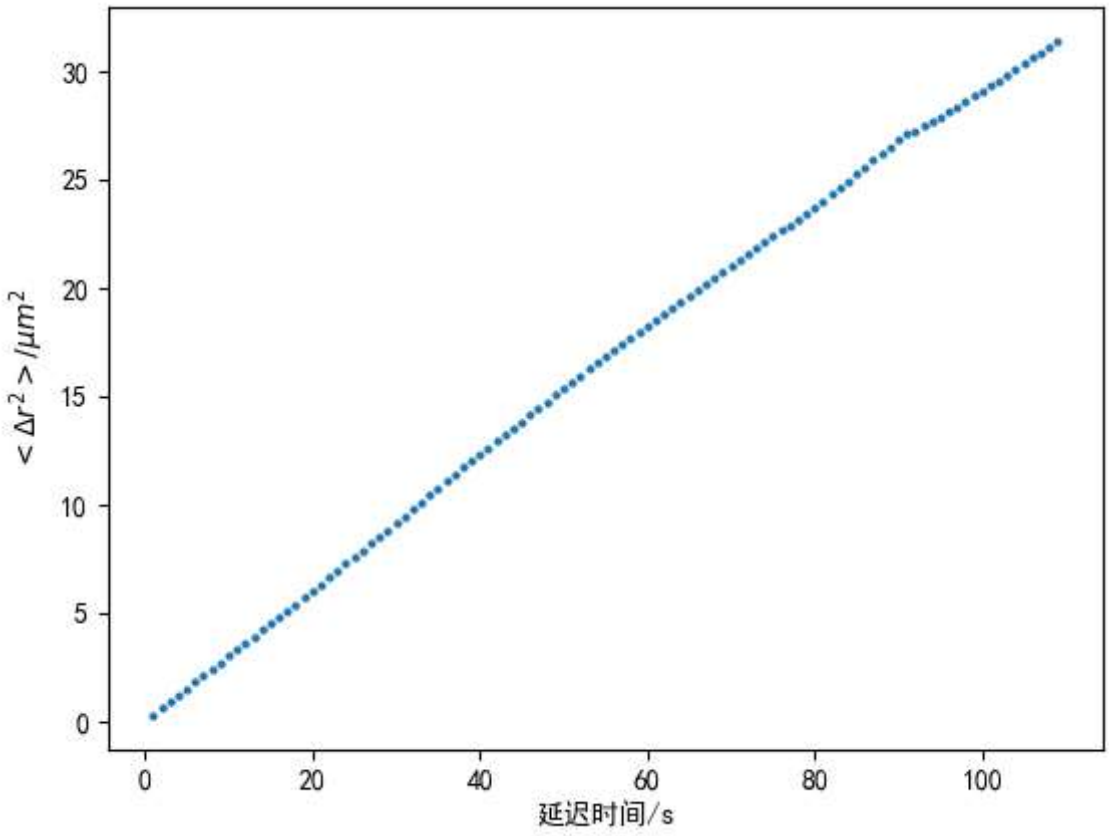
In [ ]:
```
plt.scatter(em["lagt"],1/np.sqrt(em["N"]),s=1) #相对偏差的期望值从接近0增加值50%以上
plt.ylabel("相对偏差的期望值")
plt.xlabel(r"延迟时间/s")
plt.show()
```



我们认为相对偏差<10%的数据较为可信,并对其作带误差的线性拟合

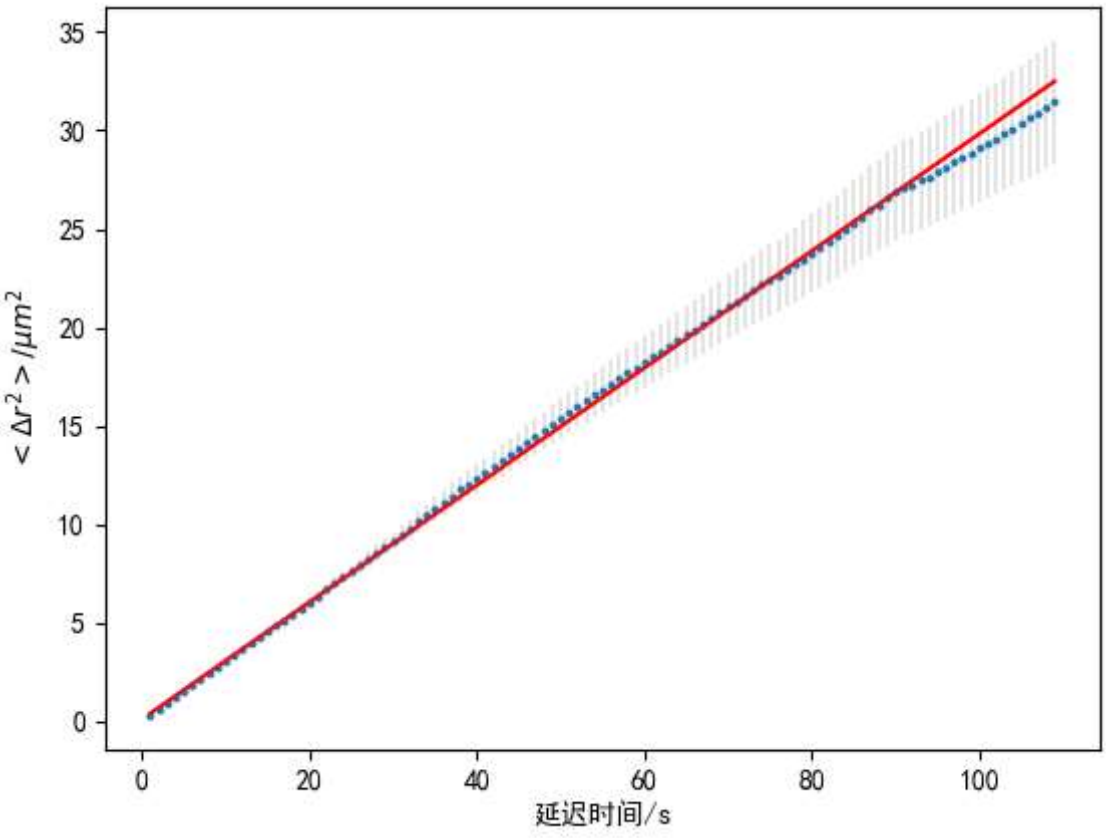
In [ ]:
```
maxlagt=np.count_nonzero(1/np.sqrt(em["N"])<0.1)
print(f"maxlagtime={maxlagt}")
plt.scatter(em["lagt"].iloc[0:maxlagt],em["msd"].iloc[0:maxlagt],s=4)
plt.ylabel(r"$<\Delta r^2>/\mu m^2$")
plt.xlabel(r"延迟时间/s")
plt.show()
#可以看到此段保持较好线性关系
```

```
maxlagtime=109
```



In [ ]:

```
x=em["lagt"].iloc[0:maxlagt]
y=em["msd"].iloc[0:maxlagt]
errors=1/np.sqrt(em["N"].iloc[0:maxlagt])
coefficients, covariance = np.polyfit(x, y, 1, w=1/errors, cov=True) #以1/相对偏差为


# 提取拟合参数和误差
slope = coefficients[0]
intercept = coefficients[1]
slope_error = np.sqrt(covariance[0, 0])
intercept_error = np.sqrt(covariance[1, 1])


# 绘制数据
plt.scatter(x, y,s=3)
plt.errorbar(x, y, yerr=y*errors, fmt='o', markersize=2,ecolor='gray', capsize=0.05,

#绘制拟合线
plt.plot(x, slope * x + intercept,c="r")
plt.ylabel(r"$<\Delta r^2>/\mu m^2$")
plt.xlabel(r"延迟时间/s")

plt.show()
print(f"slope={slope},intercept={intercept},slope_error={slope_error},intercept_erro
```

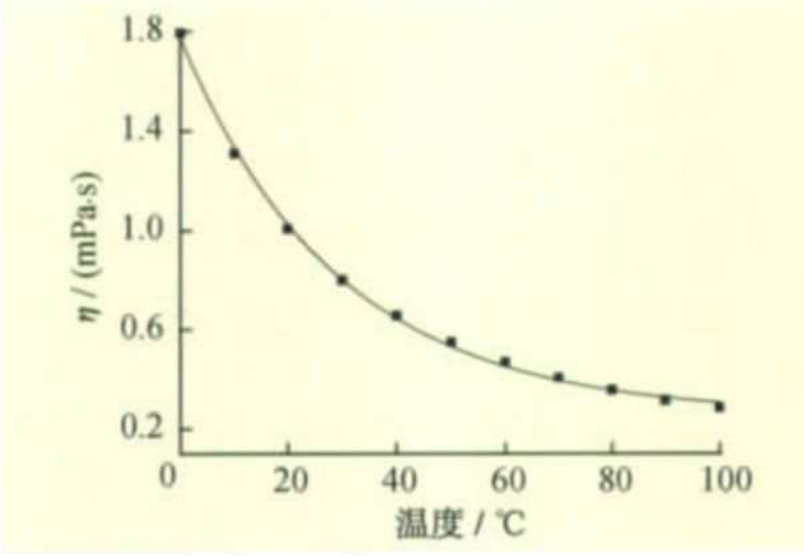


slope=0.29752437089942957, intercept=0.07012760457269701, slope_error=0.00072175822024
55167, intercept_error=0.024342625124089922

更具拟合斜率计算玻尔兹曼常数

其中粘度可根据温度用公式: $\eta = A \times 10^{\frac{B}{T-C}}$

A=2.414×10^(−5) pa·s, B=247.8K,C=140K 给出



冉诗勇.利用光学显微镜测量布朗运动[J].实验室研究与探索,2011,30(11):15-17.

In [ ]:
```
T=18.1#C°
D=slope/4
eta=2.414*10**(-5)*10**(247.8/(T+273.15-140))
kb=D*10**(-12)*3*np.pi*feature_diameter*10**(-6)*eta/(273.15+T)
print(f"kb={kb}J/K")
```

kb=8.338080058322233e-24J/K

思考

可以看到，此测量值相对标准值$kb = 1.38 \times 10^{-23} J/K$明显偏小,重复实验后，你认为这是系统误差吗?

如果是，这是由什么原因带来的?(提示：斯托克斯公式成立的条件,颗粒对液体性质的影响)

那么，你认为应该选用什么样的颗粒能够减小这样的系统误差?

聚苯乙烯的密度约为$1.05 g/cm^3$,你选择的颗粒在拍摄可能会出现怎样的问题?

换用什么材质可以避免?

推荐阅读:

C. K. Choi, C. H. Margraves, and K. D. KihmExamination of near-wall hindered Brownian diffusion of nanoparticles:Experimental comparison to theories by Brenner (1961) and Goldman(1967) Phys. Fluids 19, 103305 (2007); doi: 10.1063/1.2798811

J. Goldman, R. G. Cox, and H. Brenner, "Slow viscous motion of a sphere parallel to a plane—I: Motion through a quiescent fluid," Chem.Eng. Sci. 22, 637(1967)

H. Brenner, "The slow motion of a sphere through a viscous fluid towards a plane surface," Chem. Eng. Sci. 16, 242 (1961).

K. D. Kihm, A. Banerjee, C. K. Choi, and T. Takagi, "Near-wall hindered Brownian diffusion of nanoparticles examined by three-dimensional ratio-metric total internal reflection fluorescence microscopy (3D-TIRFM),Exp. Fluids 37, 811 (2004)

居永. 物理流体力学. 西安 : 西安交通大学出版社, 2022 362-367 悬浮液动力学

思考:

分别计算x,y方向的MSD,其关于t的斜率相同吗？重复实验后,你认为这是系统误差吗？是由什么因素导致的?

根据测量得到的D，验证其单步位移是否满足高斯分布:

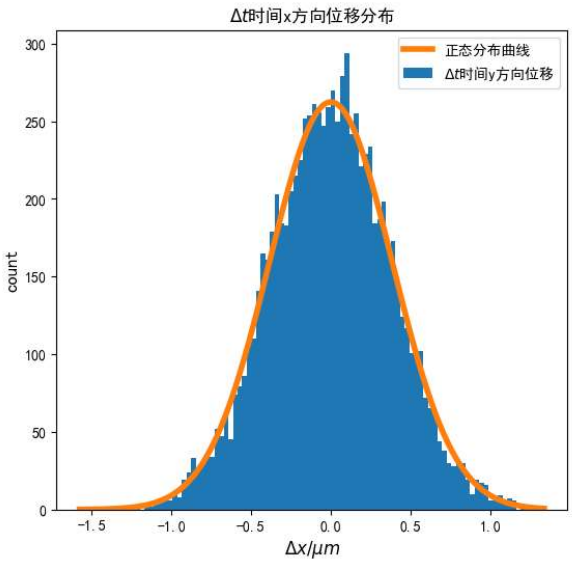
$$\rho(x, \Delta t) = \frac{1}{\sqrt{4\pi D \Delta t}} exp(-x^2/4D\Delta t)$$

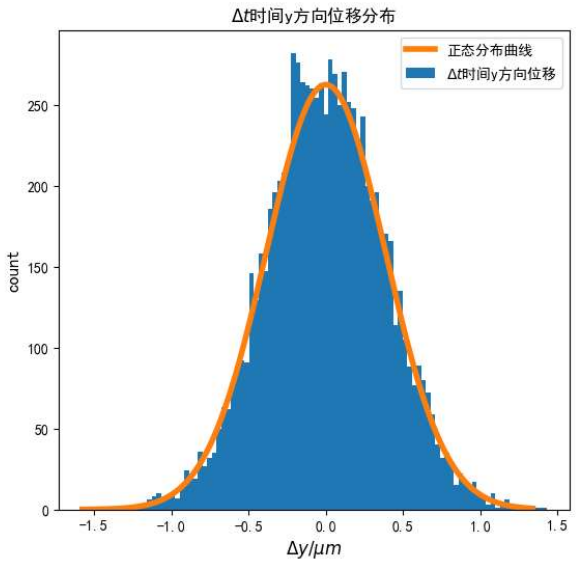
In [ ]:
```
def get_dxy(tm,pid,mmp):

    yp=np.array((tm[tm["particle"]==pid])["y"])
    xp=np.array((tm[tm["particle"]==pid])["x"])
    dxp=xp[1:]-xp[:-1]
    dyp=yp[1:]-yp[:-1]

    return np.array(dxp)*mmp,np.array(dyp)*mmp
def get_allp_dxy(tm,mmp):
    particle=tm["particle"]
    particle_unique=np.unique(particle)
    dx=np.array([])
    dy=np.array([])
    for pid in particle_unique:
        dxp,dyp=get_dxy(tm,pid,mmp)
```

```
        dx=np.hstack([dx,dxp])
        dy=np.hstack([dy,dyp])
    return dx,dy
```

In [ ]:

```
dxp,dyp=get_allp_dxy(tm,micron_per_pixel)

#绘制x方向位移分布图
plt.figure(figsize=[14,6])
plt.subplot(121)
hx=plt.hist(dxp,100)

sigma=np.sqrt(2*D/fps) #由D得到正态分布的sigma

#绘制正态曲线
x=np.linspace(min(hx[1]),max(hx[1]),1000)
y=1/np.sqrt(2*np.pi)*np.e**(-(x)**2/(2*sigma**2))/sigma
plt.plot(x,y*sum(hx[0])*(hx[1][1]-hx[1][0]),linewidth=4)
plt.xlabel("$\Delta x$"+"$/\mu m$",size=12)
plt.ylabel("count",size=12)
plt.title("$\Delta t$时间x方向位移分布")
plt.legend(["正态分布曲线","$\Delta t$时间y方向位移"])



plt.subplot(122)

#绘制y方向位移分布图
hy=plt.hist(dyp,100)

#绘制正态曲线
x=np.linspace(min(hx[1]),max(hx[1]),1000)
y=1/np.sqrt(2*np.pi)*np.e**(-(x)**2/(2*sigma**2))/sigma
plt.title("$\Delta t$时间y方向位移分布")
plt.plot(x,y*sum(hx[0])*(hx[1][1]-hx[1][0]),linewidth=4)
plt.xlabel("$\Delta y$"+"$/\mu m$",size=12)
plt.ylabel("count",size=12)
plt.legend(["正态分布曲线","$\Delta t$时间y方向位移"])
plt.show()
```





思考：

如何验证单步位移的测量值确实来自正态分布总体?

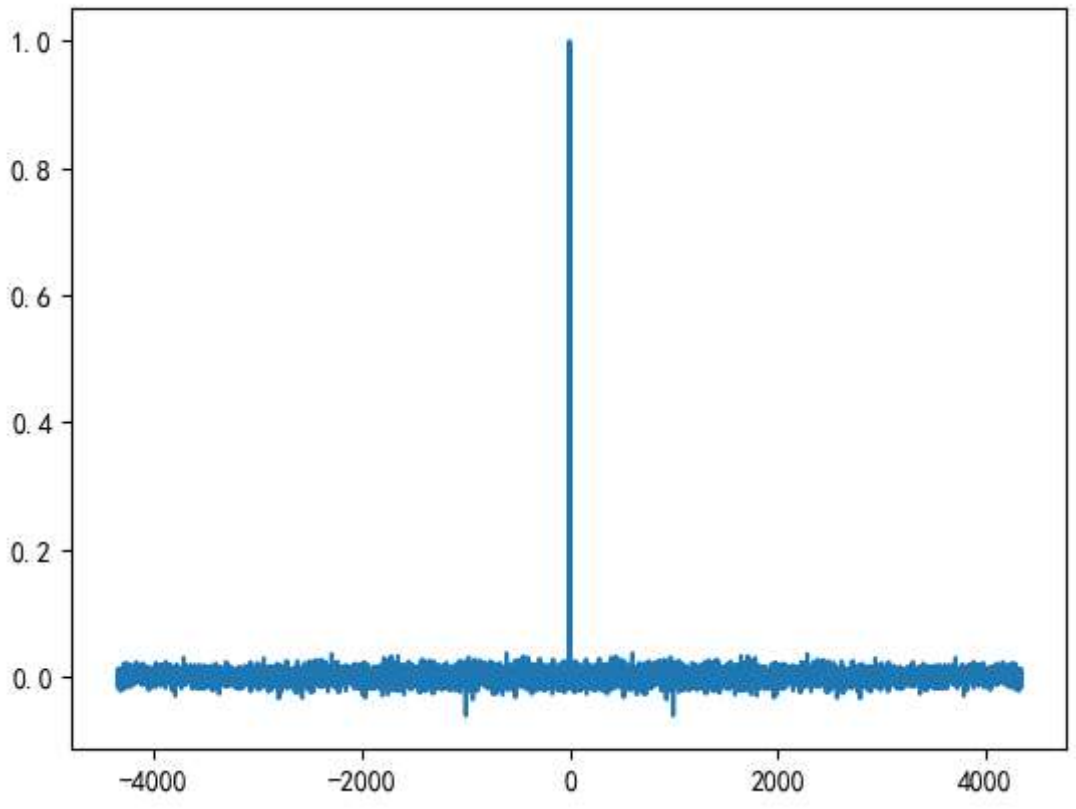
如何验证布朗运动单步位移的无记忆性?

In [ ]:
```
# 对数据作峰度，偏度检验
from scipy.stats import kurtosistest, skewtest
test=dyp  #检验y方向单步位移
# 计算峰度检验
kurtosis_stat, kurtosis_p_value = kurtosistest(test)
print(f'Kurtosis Test Statistic: {kurtosis_stat}')
print(f'P-value: {kurtosis_p_value}')

# 计算偏度检验
skewness_stat, skewness_p_value = skewtest(test)
print(f'Skewness Test Statistic: {skewness_stat}')
print(f'P-value: {skewness_p_value}')

#p值>0.05,故认为数据来自正态总体
```

```
Kurtosis Test Statistic: 1.2240435079888197
P-value: 0.2209358151073194
Skewness Test Statistic: -0.17295045414157423
P-value: 0.8626903724945251
```

In [ ]:
```
#计算x方向位移的相关性sum(dx(i+k)dx(i))，可以看到其仅在k=0时取1，其余几乎为0，这反映
test=dxp
corr = np.correlate(test, test, mode='same') /np.sum(test**2)
plt.plot(range(-len(dxp)//2,len(dxp)//2),corr)
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x27c2cb76e50>]
```



如果使用ImageJ得到颗粒的轨迹数据的csv文件，可以使用以下方法将其转换为适合trackpy的dataframe格式

In [ ]:
```
def imagej2tpy(data):
    data.rename(columns={'TRACK_ID':"particle" }, inplace=True)
    data.rename(columns={'FRAME':"frame" }, inplace=True)
```

```
    data.rename(columns={'POSITION_X':"x" }, inplace=True)
    data.rename(columns={'POSITION_Y':"y" }, inplace=True)

    data=data.drop([0,1,2])

    data["frame"]=data["frame"].astype(float)
    data["x"]=data["x"].astype(float)
    data["y"]=data["y"].astype(float)
    data["particle"]=data["particle"].astype(float)
    #更改列名称及数据类型

    data=data.sort_values(by='frame')
    #按帧序号排列

    return data
```

In [ ]:
```
data=pd.read_csv("data.csv")
data=imagej2tpy(data)
tp.emsd(data,micron_per_pixel,fps,max_lagtime=1000,detail=True)
```

```
c:\Users\xyc\anaconda3\lib\site-packages\trackpy\motion.py:235: FutureWarning: Using the level keyword in DataFrame and Series aggregations is deprecated and will be removed in a future version. Use groupby instead. df.median(level=1) should use df.groupby(level=1).median().
  results = msds.mul(msds['N'], axis=0).mean(level=1)  # weighted average
c:\Users\xyc\anaconda3\lib\site-packages\trackpy\motion.py:236: FutureWarning: Using the level keyword in DataFrame and Series aggregations is deprecated and will be removed in a future version. Use groupby instead. df.median(level=1) should use df.groupby(level=1).median().
  results = results.div(msds['N'].mean(level=1), axis=0)  # weights normalized
c:\Users\xyc\anaconda3\lib\site-packages\trackpy\motion.py:242: FutureWarning: Using the level keyword in DataFrame and Series aggregations is deprecated and will be removed in a future version. Use groupby instead. df.sum(level=1) should use df.groupby(level=1).sum().
  results['N'] = msds['N'].sum(level=1)
```

Out[ ]:

frame	<x>	<y>	<x^2>	<y^2>	msd	N	lagt
1	-0.019040	-0.000693	0.153815	0.160285	0.314101	21561.550087	1.0
2	-0.038368	-0.001746	0.313650	0.321682	0.635332	14327.834809	2.0
3	-0.057565	-0.002791	0.473101	0.479993	0.953095	10145.681529	3.0
4	-0.077030	-0.003966	0.630292	0.637105	1.267397	7765.946525	4.0
5	-0.096688	-0.005798	0.789397	0.789244	1.578641	6263.875395	5.0
...	...	...	...	...	...	...	...
995	-15.981485	-1.223974	350.328649	122.845797	473.174446	9.029002	995.0
996	-15.997443	-1.139700	348.280344	122.628184	470.908528	9.022624	996.0
997	-15.951158	-1.059571	345.245609	120.026428	465.272037	9.016065	997.0
998	-16.043947	-1.021943	346.324518	119.563738	465.888256	9.009023	998.0
999	-16.178069	-0.963456	351.017382	119.457090	470.474472	9.000000	999.0

999 rows × 7 columns

In [ ]: