第十三题第二小题

邱哲儒
（PB15000034, 中国科学技术大学）

2017 年 11 月 21 日

题　目　研究有取向的布朗粒子（如纳米棒）的随机行走，计算取向的自关联函数 $C(t) = \langle u_x(t)u_x(0)\rangle$。其中 u_x 为取向单位矢量在 x 轴上的投影。

1　理论分析

对于有取向的布朗运动粒子，其运动在二维或三维空间中的特性应当是有类似性的，在一篇法文的比较古老的文献 [4] 中有提及。为了推导与计算上的方便，本题中只考虑粒子被局限在二维平面上运动的情况。这个粒子可以被简化视为为一个有不同长短轴的一个旋转椭球体，在二维投影中是一个椭圆，如实际粒子具有不同的形貌，只要运动特性可以以相似的方式概括，其讨论与此完全相同。

1.1　粒子的运动方程

为了得到在观察者看来的粒子运动状态，参考文献 [1]，[2] 中的讨论，如图1所示，我们可以考虑粒子在原点固连在粒子质心上，轴与粒子的某个特定方向 (在此将 $\tilde{x}$ 选为长轴方向) 重合的参考系中的运动，后做参考系变换来得到感兴趣的观察者坐标系中的运动。

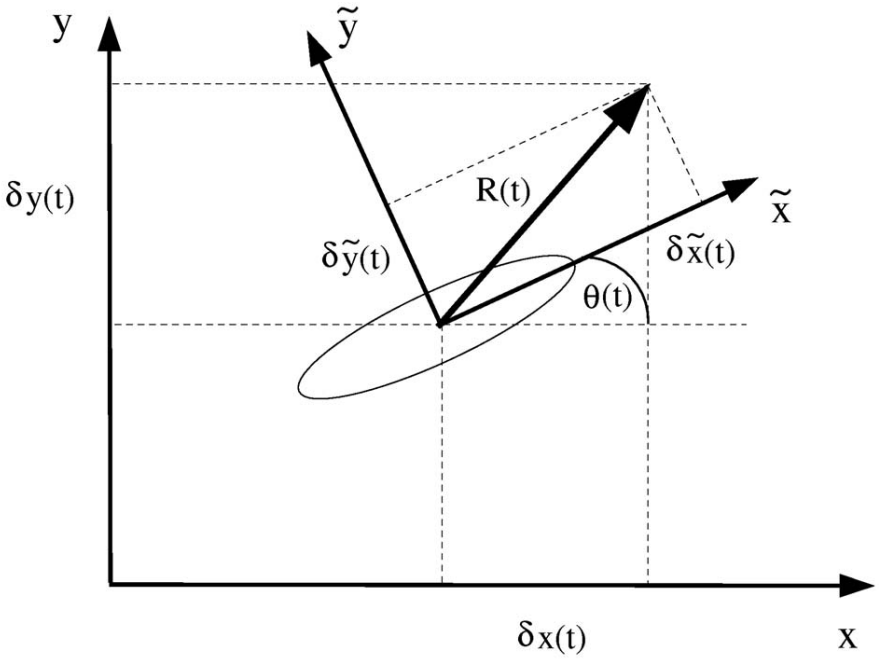

图 1: 粒子随体坐标系示意图，引自 [1]

在 t 时刻粒子的位移可以用其质心在观察者参考系中的坐标 $(\delta x(t), \delta y(t))$ 与 $\tilde{x}$ 轴与观察者系 x 轴的夹角 θ 来描述，在这个随体坐标系中，粒子完全只受随机力 (矩) η_i, $i = 1, 2, 3$ 的作用，可以直接

按 Langevin 方程写出其运动方程

$$\begin{aligned}\Gamma_1^{-1}\frac{\partial \tilde{x(t)}}{\partial t}&=\tilde{\eta_1}(t)\\ \Gamma_2^{-1}\frac{\partial \tilde{y(t)}}{\partial t}&=\tilde{\eta_2}(t)\\ \Gamma_3^{-1}\frac{\partial \theta(t)}{\partial t}&=\tilde{\eta_3}(t)\end{aligned}\tag{1}$$

Γ_1,Γ_2 可以认为是椭球体在长轴与短轴方向受到的阻尼系数的倒数，可以认为长轴方向的阻尼远大于短轴方向 $\Gamma_1\gg\Gamma_2$，椭球本身对转矩的阻力系数的倒数为 Γ_3，这三个参数的具体的取值取决于实际的物理模型。$\tilde{\eta_i}$，$i=1,2,3$ 是均值为 0 的噪声，由于此时的转动与平动完全解耦，每一项均分别由热力学控制，可知噪声 $\tilde{\eta_i}$ 的关联函数为

$$\left\langle\tilde{\eta_i}(t)\tilde{\eta_j}(t')\right\rangle=\frac{2k_bT}{\Gamma_i}\delta_{i,j}\delta(t-t')\tag{2}$$

因此可认为随机噪音的标准差分别为 $D_x=\sqrt{k_bT\Gamma_1},D_y=\sqrt{k_bT\Gamma_2},D_\theta=\sqrt{k_bT\Gamma_3}$，这三个标准差也对应着随体系中三个独立的 Brownian 运动扩散系数的平方根。

接下来可以将方程(1)变换回观察者所在的实验室坐标系中，因为随体坐标系只是简单的对实验室坐标系作了一个角度为 $\theta(t)$ 的旋转变换，有关系

$$\begin{aligned}\delta x&=\cos\theta\delta\tilde{x}-\sin\theta\delta\tilde{y}\\ \delta y&=\sin\theta\delta\tilde{x}+\cos\theta\delta\tilde{y}\end{aligned}\tag{3}$$

1.2 单位矢量在 x 轴投影自相关函数 C(t) 的分析

定义 1: 自相关函数可以如题定义为

$$C(t)=\langle u_x(t)u_x(0)\rangle\tag{4}$$

式中平均值符号理解为对一系列具有不同初值 $\theta(0)=\theta_0$ 的粒子进行平均。这种定义的自相关函数在 $\theta=0$ 时的值 $C(0)=\frac{1}{2}$。

从理论上看，由于 Brownian 运动具有 Markov 性和平稳性 [5]，初态为 θ_0 的粒子在将来的某个时刻 t 的 $\theta(t)$ 取值的分布为正态分布 $\mathsf{N}(\theta_0,D_\theta^2t)$，由积分公式

$$\frac{1}{2\pi}\int_0^{2\pi}\int_{-\infty}^{\infty}\frac{1}{\sqrt{2\pi tD_\theta}}\cos(\theta_0)\cos(\theta+\theta_0)\,e^{-\frac{\theta^2}{2tD_\theta}}\mathrm{d}\theta\mathrm{d}\theta_0=e^{-\frac{t}{2}}\quad t>0\tag{5}$$

可以计算得到自相关函数的理论值为

$$C(t)=\langle u_x(t)u_x(0)\rangle=\frac{1}{2\pi}\int_0^{2\pi}\int_{-\infty}^{\infty}\frac{1}{\sqrt{2\pi tD_\theta}}\cos(\theta_0)\cos(\theta+\theta_0)\,e^{-\frac{\theta^2}{2tD_\theta^2}}\mathrm{d}\theta d\theta_0=\frac{1}{2}e^{-\frac{D_\theta^2t}{2}}\tag{6}$$

为一个随时移量 t 指数衰减的函数。

定义 2: 实际上使用更加广泛的对自相关函数的定义是讲义上给出的，前序题中用于研究随机数发生器性质的

$$C(l)=\frac{\langle x_nx_{n+l}\rangle-\langle x_n\rangle^2}{\langle x_n^2\rangle-\langle x_n\rangle^2}\tag{7}$$

式中平均值符号的意义是对不同的起始计算时刻 n 进行平均。这种定义的自相关函数在 $\theta=0$ 时的值 $C(0)=1$。

取向单位矢量在 x 轴投影 $u_x=\cos\theta$，在以上的分析中可知 θ 符合一个一维 Brownian 运动的模型, 据随机过程理论的相关知识，对于按 Brownian 运动演化的随机变量 θ，$\mathrm{Cov}(X(s),X(t))=D^2\min(s,t)$[5]，按(7)式的定义，其自相关函数与 t 无关，恒等于 1，这可以理解为任一时刻的随机变量值都与过去的取值有着很好的线性相关性。但是对于 u_x，由于 $u_x=\cos\theta$ 对 θ 进行了一个非线性、有周期性的变换，直观来看在 t 较大，θ 波动较大时，余弦函数的作用打乱了这个线性关系，预期在 $t=0$ 时，$C(0)=1$，在 $t\to\infty$ 时，$C(t)\to0$。

1.3 粒子运动的特征

若粒子的初始状态均设置为 $\theta(0)=0$，考虑 $D_\theta \to 0$ 的极限情况，此时粒子的旋转自由度被完全禁止，粒子的运动完全等同于两个方向上独立的一维 Brownian 运动，在观察者坐标系中 y 方向运动将远没有 x 方向运动明显，粒子几乎只能在长轴方向的直线上运动，扩散常数 $D_{yob}^2 = \left\langle y^2 \right\rangle / t = D_y^2 \ll D_{xob}^2 = \left\langle x^2 \right\rangle / t = D_x^2$。

相反的，考虑 $D_\theta \to \infty$ 的极限情况，此时粒子的取向变化太快，粒子的取向会快速趋向随机 (表现为正态分布的方差急剧增大，折叠回 $[0, 2\pi)$ 区间上后接近于均匀分布)，运动会很快的趋向各向同性，粒子的运动会接近二维平面上各向同性粒子的 Brownian 运动。当 $t \to \infty$ 时，x, y 方向的扩散常数均应当很快的趋向于 $\frac{D_x^2+D_y^2}{2}$，关于粒子运动趋向各向同性的过程的精确的理论描述，可以参考文献 [2]。

对于各向异性的粒子，其各向异性的阻力系数使平动与转动之间有了耦合，混合关联函数 $\left\langle x^2 \cos 2\theta \right\rangle / t$,$\left\langle y^2 \cos 2\theta \right\rangle / t$,$2 \left\langle xy \sin 2\theta \right\rangle / t$ 的取值均不平凡，与不相干的平均值 $\left\langle x^2 \right\rangle \left\langle \cos 2\theta \right\rangle / t$, $\left\langle y^2 \right\rangle \left\langle \cos 2\theta \right\rangle / t$,$2 \left\langle xy \right\rangle \left\langle \sin 2\theta \right\rangle / t$ 间会有明显的偏离，其理论推导见文献 [2] 及其补充材料。

1.4 位移的四阶累计量

虽然在随体坐标系中位移的统计分布是高斯分布，但在观察者眼中由于平动与转动之间的耦合而偏离高斯分布 [3]，为了显示其与高斯分布的偏离，可以计算位移的四阶原点矩 [1]，或者四阶累计量 (cumulant)[2]。

在此计算的对于固定的起始方位角 θ_0 的四阶累计量定义为

$$C_{\theta_0}^{(4)}(t) = \left\langle [x(t)]^4 \right\rangle_{\theta_0} - 3 \left\langle [x(t)]^2 \right\rangle_{\theta_0}^2 \tag{8}$$

对于一个服从高斯分布的随机变量，其四阶累计量应当为 0，所以非零值可以作为分布偏离高斯分布的指标。可定义归一化的“非高斯指数”p 为

$$p(t, \theta_0) = \frac{C_{\theta_0}^{(4)}(t)}{3 \left\langle [x(t)]^2 \right\rangle_{\theta_0}^2} \tag{9}$$

2 计算的编程实现

在计算中使用随机游走来近似 Brownian 运动的过程，可以直接对每一时间步计算粒子在前一时刻的随体坐标系中的位移(1)，再用(3)将位移作用在观察者坐标系上。由于运动方程的齐次性，系统的运动尺度是可以与时间一同放缩的，在模拟中各个物理量的单位都是无关紧要的，一秒可以对应一个时间步。噪音 η_i 可以使用每一时间步相互独立的，带有 0 均值和相应标准差的正态分布随机数来模拟，实际编程中使用`RNG` 类中实现的 Box-Muller 方法产生。

除按(7)定义的自相关函数外，各个统计量的计算是通过对同一时刻一群粒子各个相关量的累加进行的，更新各个累加和并计算输出各个统计量的过程被安排在了粒子坐标更新后。为了更快速的完成粒子坐标的更新和统计量的求和，计算过程可以使用 OpenMP 进行并行化，可以在多核机器上大幅提高性能。为了计算自相关函数编制程序如附录C，计算其他统计量的程序如附录D。

按(7)式定义的自相关函数是通过对一个粒子长时间的演化进行分析得到的，使用一个可以随机访问的双端队列来记录先前多个时刻的位置，并且随时计算包括 $\langle x_n x_{n+l} \rangle$ 在内的各个累加和，在达到一定步数时计算输出，编制程序如附录B。

为了能够直观的指示当前参数下单个粒子运动的状态，在模拟一个粒子运动的过程中可实时使用 OpenCV 库绘制粒子的运动轨迹，并可将状态保存至视频文件中以便回看，编写程序如附录A。

3 计算结果

以下计算中初始随机数种子均默认为15000034LL,434343LL。

3.1 自相关函数

在 D_θ=0.05 与 0.03 时，粒子取向 θ 分布为 $[0, 2\pi)$ 上均匀分布，统计粒子总数为 5000000 个时，使用附录C中的程序计算单位矢量在 x 轴投影 u_x 按(4)定义的自相关函数 $C(t)$ 与时移量 t 间的关系。将得到的结果乘二，以便于另一种定义相比较。

在 D_θ=0.05 与 0.03 时，统计步数为 100000000 时，使用附录B中的程序计算按(7)定义的自相关函数 $C(t)$ 与时移量 t 间的关系。得到结果叠绘于图2上。

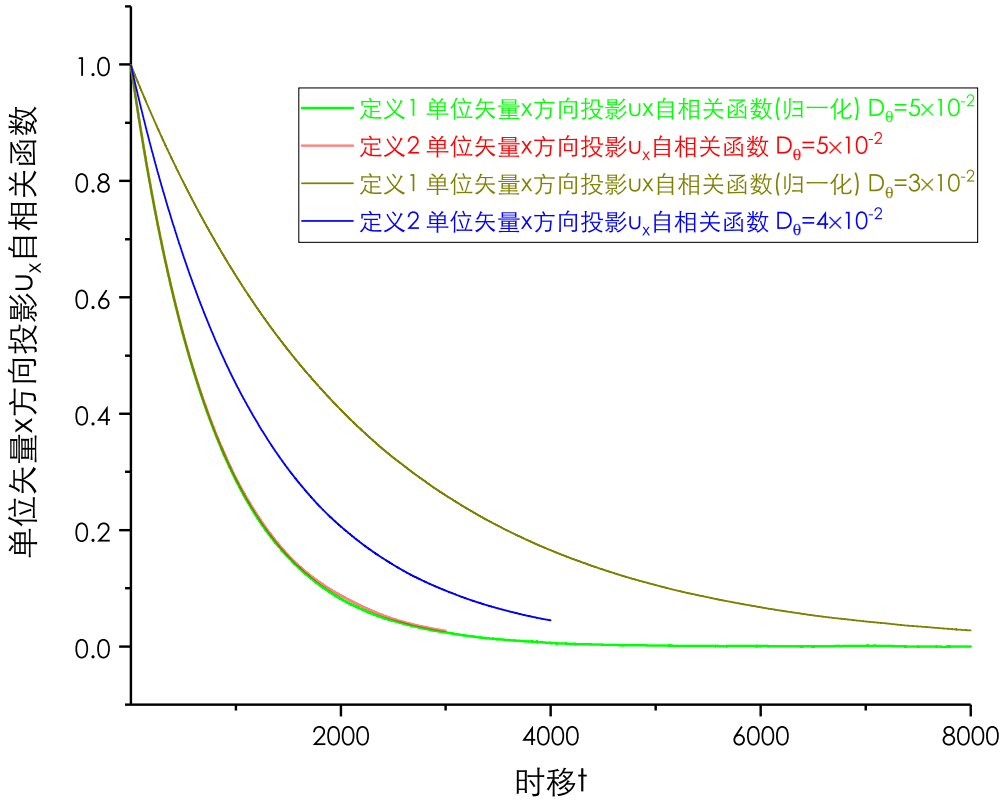

图 2: 不同参数时自相关函数图

从图中可见使用式(4)与(7)的定义计算得到的自相关函数在数值上仅相差一个由 $\langle\cos\theta\rangle = \frac{1}{2}$ 所带来的因子，得到的结果基本一致。可知在时移量 t$\to\infty$ 时，自相关函数值均趋近于 0，在 D_θ 越大时，趋近的速度越快。这个结果与定性分析结果一致。

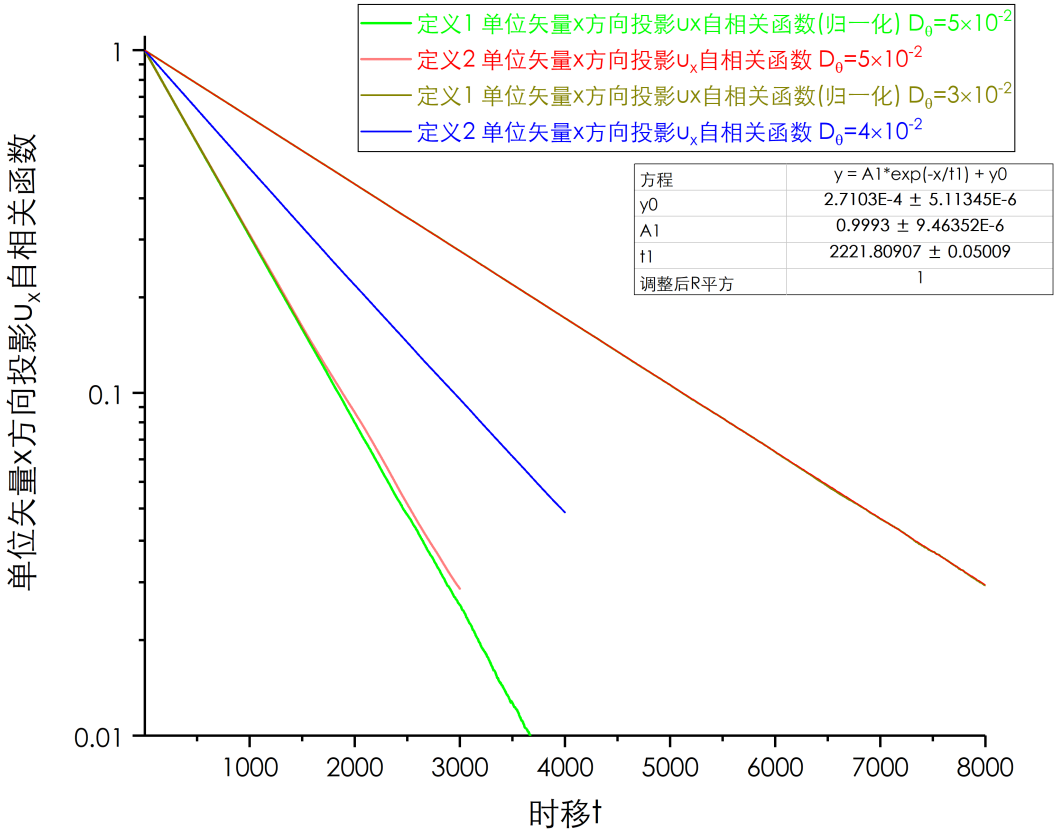

图 3: 不同参数时自相关函数对数图

将自相关函数图改用单对数坐标轴绘制如图3，可见自相关函数值是按指数规律收敛到 0 的，图中对 $D_\theta = 0.03$ 时的自相关函数进行了拟合，得到拟合规律 $C(t) = \mathrm{e}^{-t/2222}$，拟合线与计算值符合程度很好。而理论上对 $D_\theta = 0.03$ 时的自相关函数按式(6)有预期 (已乘归一化因子 2)

$$C(t) = \mathrm{e}^{-\frac{t}{2/D_\theta^2}} = \mathrm{e}^{-\frac{t}{2222.22}} \tag{10}$$

与拟合线方程几乎完全一致。

3.2 粒子运动状态

对于不同的 D_θ 数值，粒子的运动状态各异，粒子轨迹的形态各不相同，在取 $D_x = 0.99, D_y = 0.01$ 时，绘出几个典型的 D_θ 取值所对应的粒子运动轨迹如图4，5，6，7，8所示，粒子运动的视频可在提交文件中找到。

建议

建议查看记录有粒子运动的几个视频以全面了解粒子的运动状态。

在这些视频与图像中，白色的折线代表粒子的运动轨迹，为了区分不同时刻的粒子轨迹，随着时间的演化，轨迹会渐渐“褪色”，绿色的小短线代表在某些特定的抽样时刻粒子位置与长轴的方向。

图中下部的信息代表模拟时的各个参数，DX 对应 D_x,DY 对应 D_y,DT 对应 D_θ,WS 对应绘制图形时一单位长度对应的像素个数,S1 代表随机数种子 1,TS 是时间步数,XP,YP 是观察到的粒子位置,TH 代表 θ,RUN 与STOP 指示模拟程序的状态。

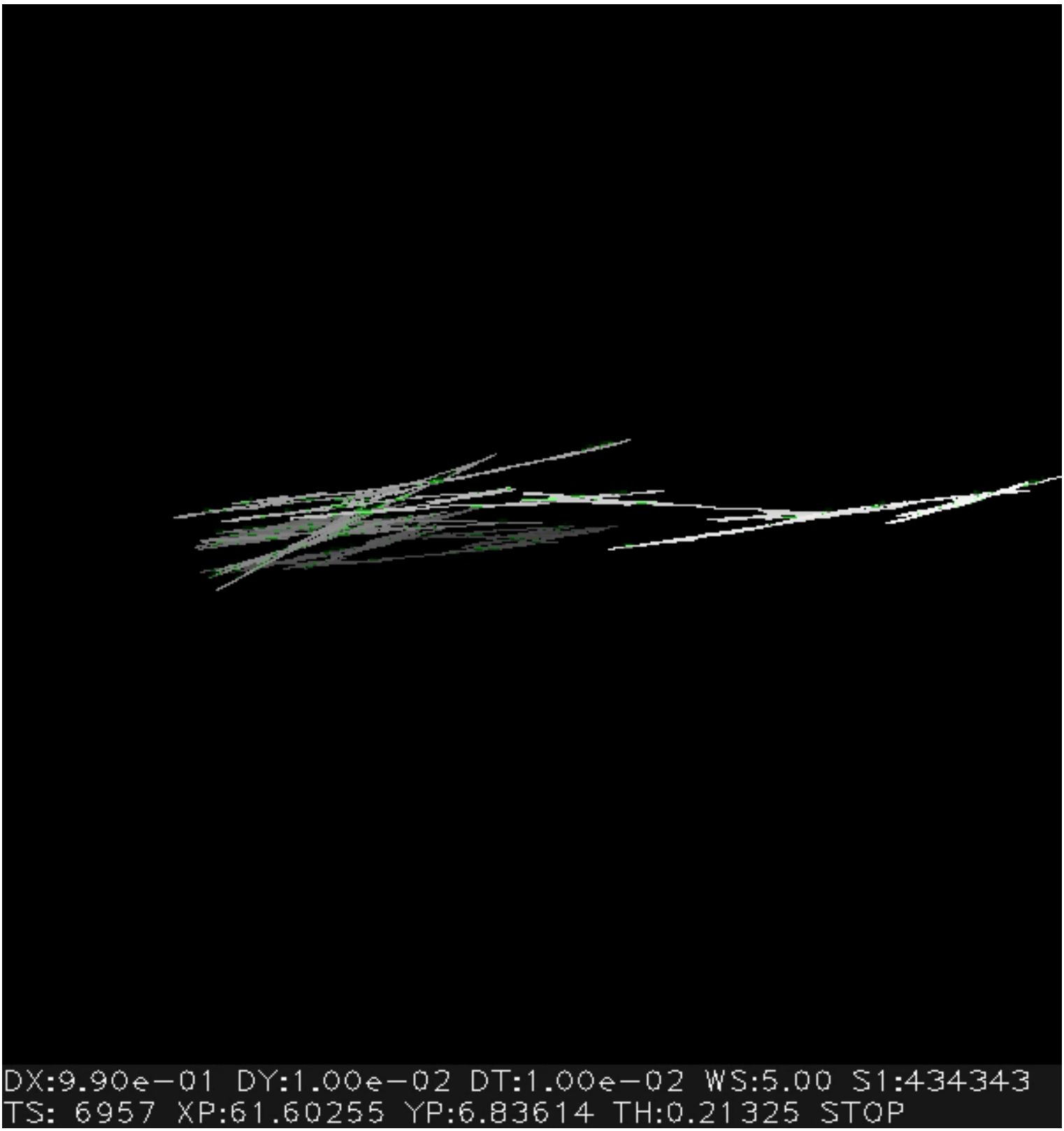

图 4: $D_\theta = 1 \times 10^{-2}$ 时典型运动轨迹图

图 5: $D_\theta = 3 \times 10^{-2}$ 时典型运动轨迹图

图 6: $D_\theta = 5 \times 10^{-2}$ 时典型运动轨迹图

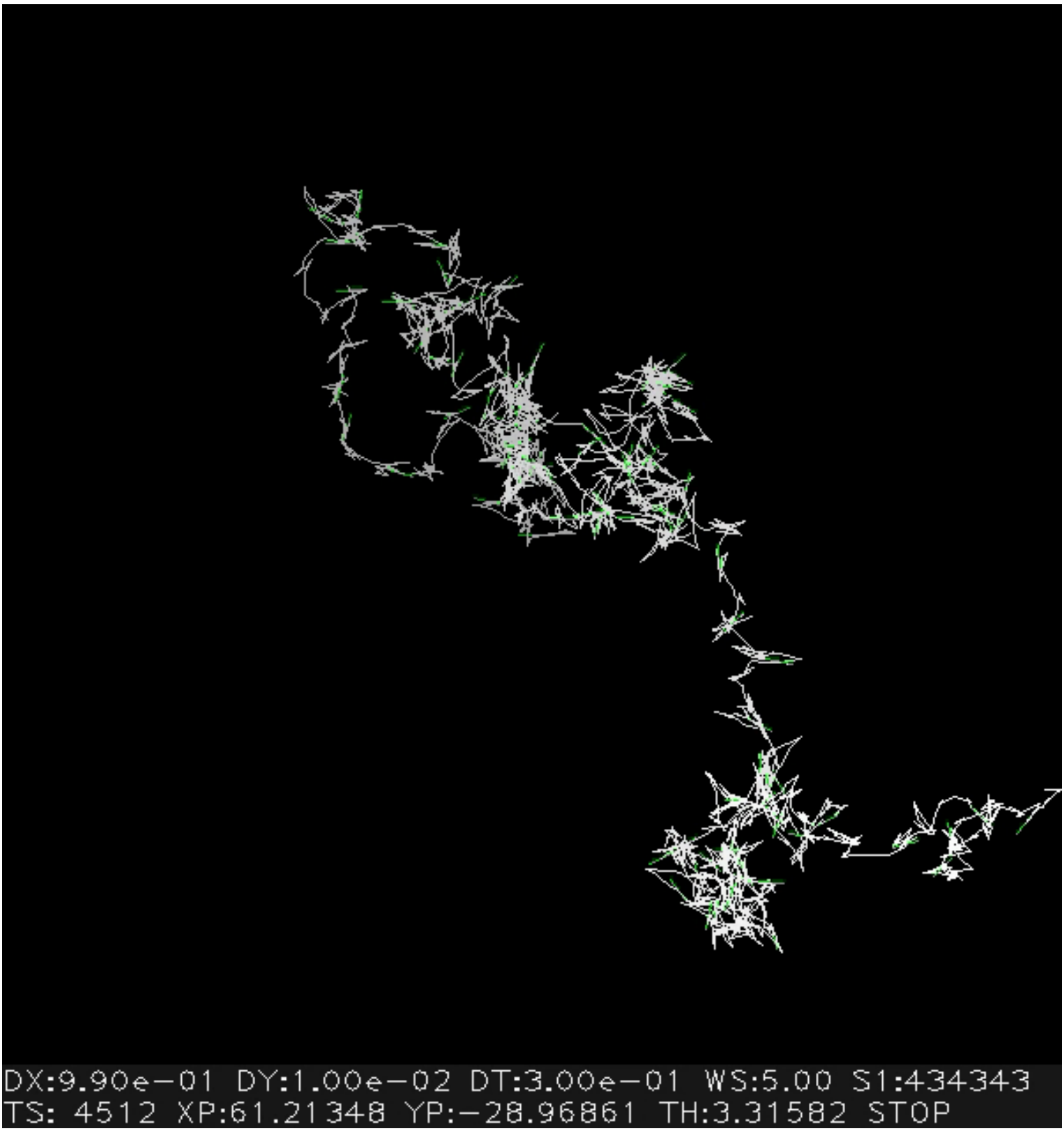

图 7: $D_\theta = 0.3$ 时典型运动轨迹图

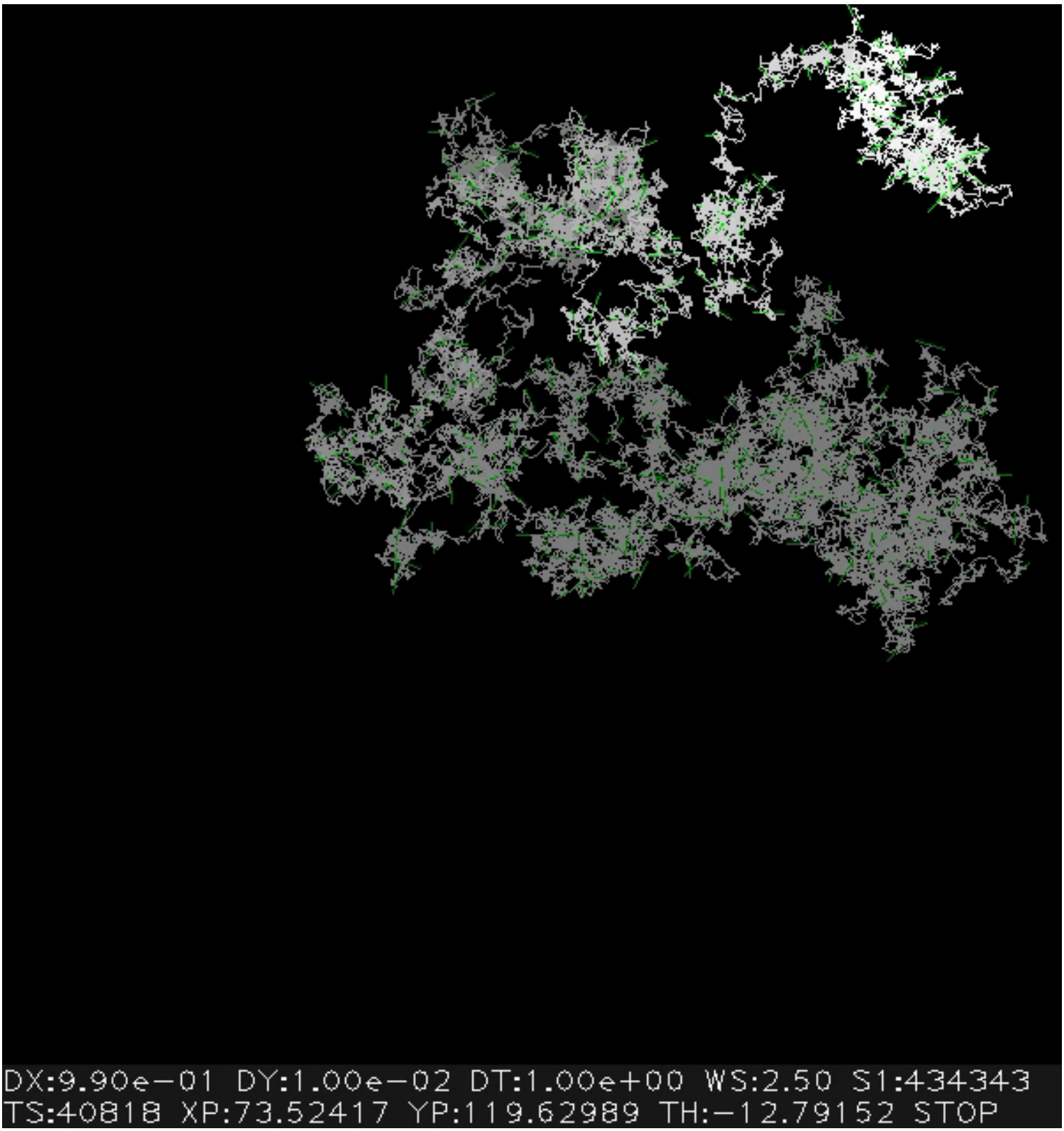

图 8: $D_\theta = 1$ 时典型运动轨迹图

从图与视频中可见，当 D_θ 较小时，粒子确实在一段时间内走 Z 形路线，形态上与各向同性运动有较大差别，可注意到绘制出代表杆方向的绿线常常与白色的轨迹重合，粒子几乎总是按最近的长轴方向运动。

当 D_θ 较大时，粒子的轨迹明显更加“杂乱”，绿线并不常与白色的轨迹重合，可见在两次输出的时间内，θ 已经有了很大变化，并没有一个确定的取向。这一点也可从视频中叠加的状态信息中TH 的值快速变化看出。直观上看接近于二维平面上的各向同性粒子的 Brownian 运动的轨迹。$D_\theta \geq 1$ 时，已经几乎无法看出其与各向同性二维 Brownian 运动轨迹的差别。当 D_θ 逐渐增大时，运动状态在两种极端情况间过渡。这个现象与1.3小节中的定性讨论结论一致。

3.3 粒子 x, y 方向上扩散系数 D_{obx}, D_{oby}

为了能够显示出粒子运动的各向异性与运动时间的关系，可以计算在不同的 D_θ 取值时，不同时刻下观察者坐标系中 x, y 方向位移平方的平均值与时间之比 $\langle x_i^2 \rangle / t$，即不同时间对应的 x, y 方向的扩散系数 D_{obx}, D_{oby}。

在粒子初始取向固定为 0 ($\theta(0) = 0$)，$D_x = 0.99, D_y = 0.01$，统计粒子总数为 10000000 个时，绘出扩散系数 D_{obx}, D_{oby} 与时间关系如图9

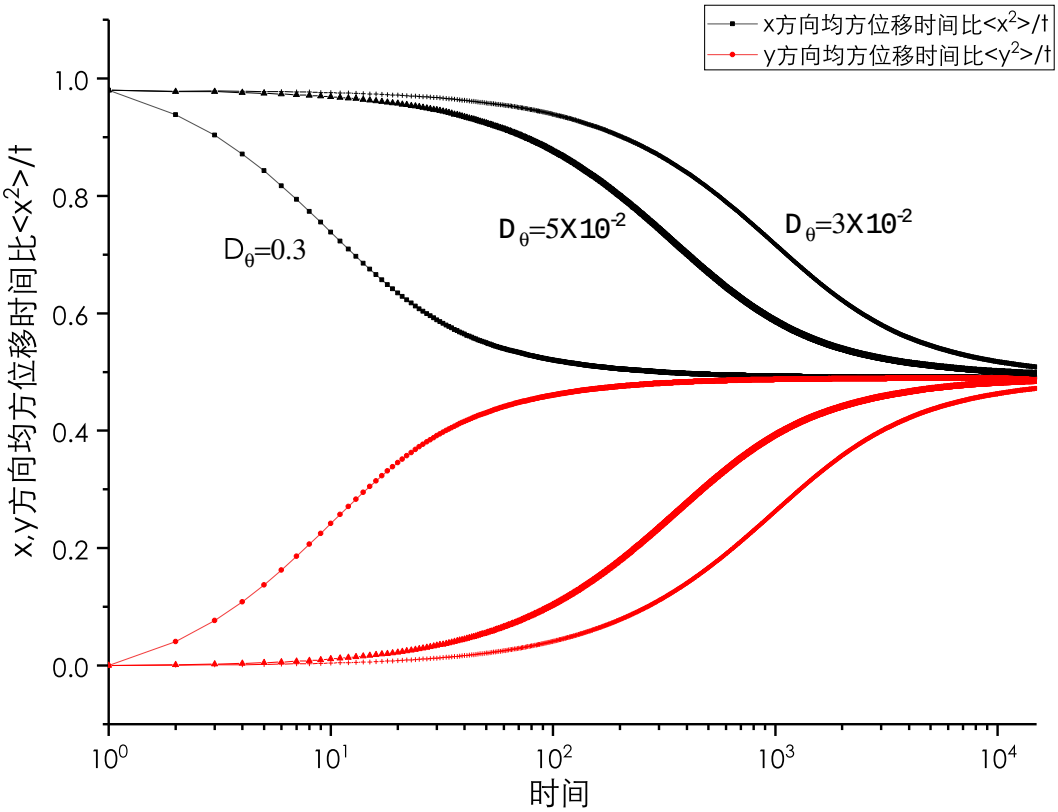

图 9: 扩散系数 D_{obx}, D_{oby} 与时间关系图

$t = 1$ 时，D_{obx}, D_{oby} 分别接近 D_x^2, D_y^2；当 t 增大时，D_{obx}, D_{oby} 趋向于一个共同的稳定值 $\frac{D_x^2+D_y^2}{2} = 0.4901$。当 D_θ 越小时，收敛越慢。这个结果与1.3小节中的定性讨论结论一致，也与观察运动轨迹得到的经验印象一致。当 $D_\theta = 0.3$ 时，仅需要 200 单位的时间即可让粒子在两方向上扩散常数之差小于 0.01, 但对于 $D_\theta = 3 \times 10^{-2}$，下降的过程需要超过 10^4 单位的时间，这可以解释在图7中粒子运动过程很快会变得杂乱，而图5中粒子运动的轨迹直到模拟结束仍然很有规律，水平方向位移较竖直方向更明显。

得到的结果也与文献 [2] 中图 2.(B) 的结果一致。

3.4 四阶累计量

按照式(8)，(9)的定义，可以计算表征粒子分布偏离高斯分布的程度的 p 值与时间的关系，粒子初始取向固定为 0 ($\theta(0)=0$)，$D_x=0.99, D_y=0.01$，使用 x 方向的位移进行统计，结果如图10所示

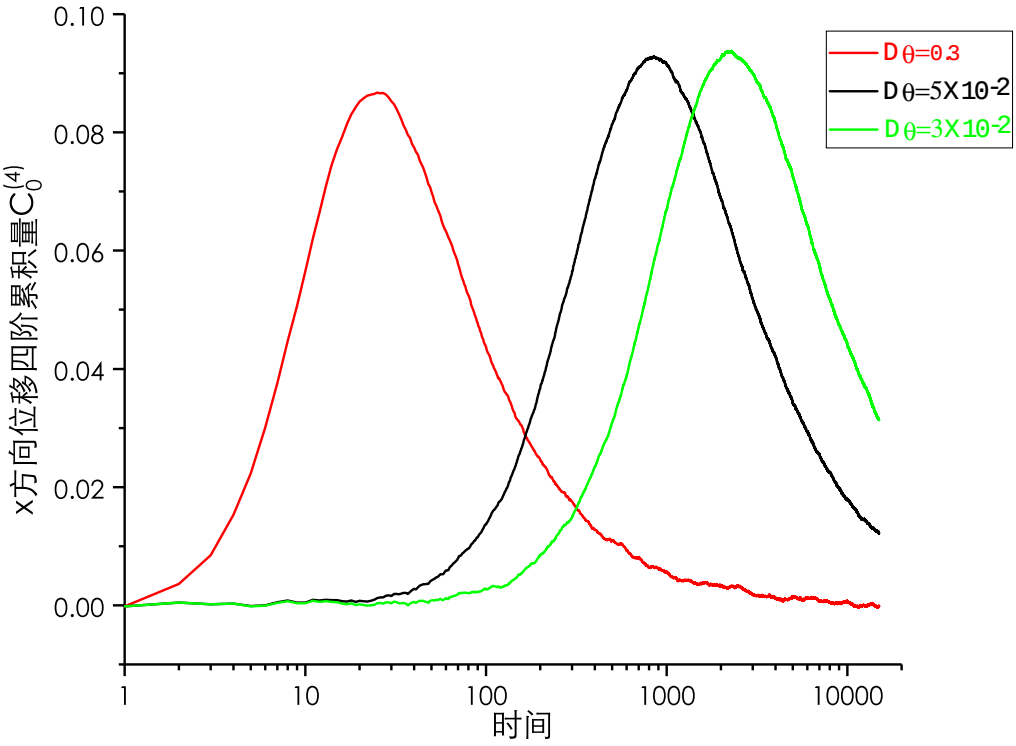

图 10: p 值与时间关系

可见由于平动与转动的耦合，分布偏离高斯分布的程度在 $t\to 0$ 时接近 0，在某个特定的时刻达到最高，在 $t\to\infty$ 时又趋向于 0。

这个结果可以解释为平动转动耦合对位移分布的影响在扩散系数 D_{obx}, D_{oby} 变化较快的运动中段最为明显，而在起始时角度的随机性还不大，运动被较好的局限在 x 方向上 ($D_x \gg D_y$) 运动近似为一维的 Brownian 运动，而 $t\to\infty$ 时，运动趋近于各向同性的二维 Brownian 运动，这两种情况下位移的分布都接近高斯分布。

这个结果也与文献 [2] 中图 3.(A) 中绿色图线的结果一致。

3.5 混合关联函数

粒子初始取向固定为 0 ($\theta(0)=0$)，$D_x=0.99, D_y=0.01, D_\theta=5\times10^{-2}$ 时，可计算出转动、两方向平动的各个混合关联函数与假设平动转动间不存在耦合时的参考结果如图11。

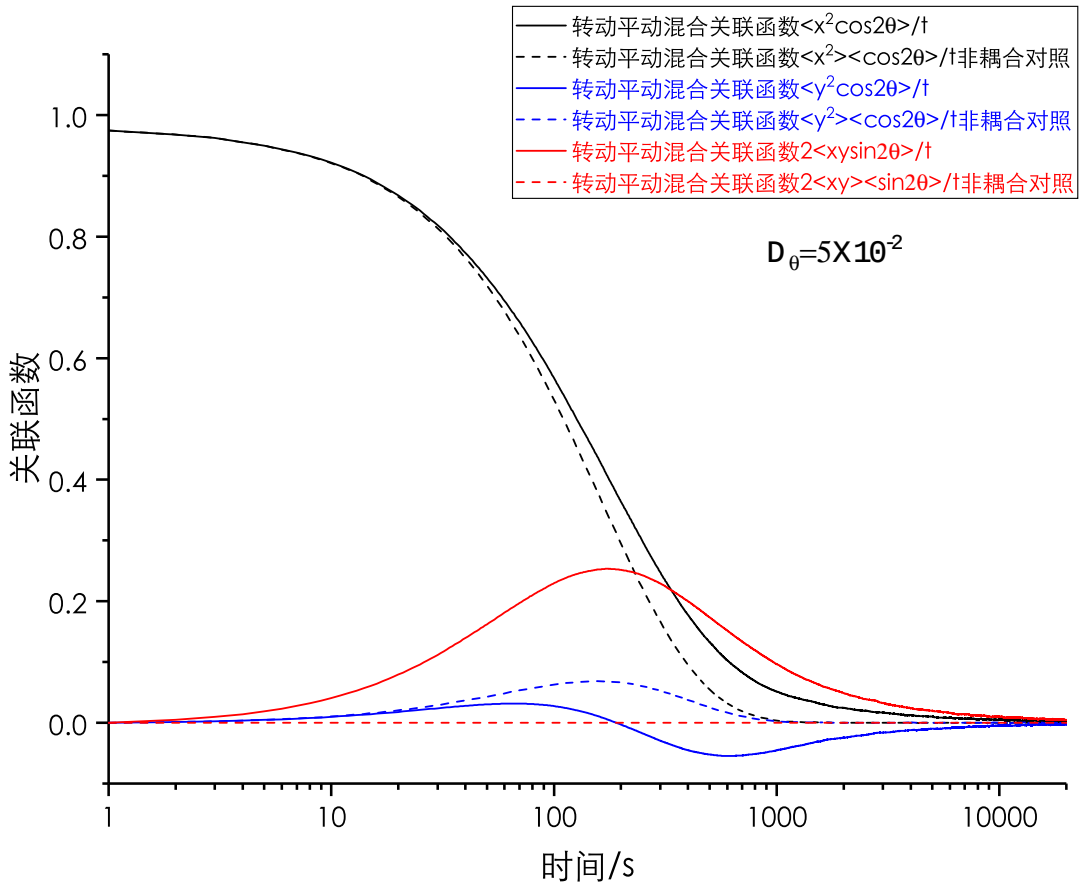

图 11: 各混合关联函数与时间关系

可见图11中各实线与虚线并不重合，可知实际情况下，各混合关联函数与假设平动转动解耦时的结果很不相同。这个结果也与文献 [2] 中图 2.(C) 中的结果一致。

4 总结与讨论

在本题中计算研究了一个各向异性杆状粒子的 Brownian 运动过程的特征，按两种不同的定义(式(4),(7)) 计算了取向的自关联函数，得到的结果基本一致。这说明自相关函数的这两种定义只有归一化方式的不同，并没有本质上的区别。

当 D_θ 不大，即粒子旋转相对不明显时 (物理上对应杆长较长)，粒子运动轨迹的形态明显的呈现出与各向同性运动不同的特征。如图4所示。而当 D_x,D_y 相近或 D_θ 较大时，运动接近一般的各向同性的二维 Brownian 运动。

另外由于转动与平动的相互耦合，系统运动的统计性质较各向同性的 Brownian 运动有一定的区别，从四阶累计量和各个混合关联函数的计算上可以看出其扩散运动与高斯分布在某些时刻有区别。但是随着时间的演化，对于包含大量粒子的系统，这些统计上的差异会被初态的随机性抹平，系统的演化回到二维 Brownian 运动。

参考文献

[1] R. Grima and S. N. Yaliraki. Brownian motion of an asymmetrical particle in a potential field. *The Journal of Chemical Physics*, 127(8):084511, 2007.

[2] Yilong Han, Ahmed M Alsayed, Maurizio Nobili, Jian Zhang, Tom C Lubensky, and Arjun G Yodh. Brownian motion of an ellipsoid. *Science*, 314(5799):626–630, 2006.

[3] S. F. Edwards M. Doi. *The Theory of Polymer Dynamics*. International Series of Monographs on Physics. Oxford University Press, USA, 1988.

[4] Francis Perrin. Mouvement brownien d'un ellipsoide (ii). rotation libre et dépolarisation des fluorescences. translation et diffusion de molécules ellipsoidales. *Journal de Physique et le Radium*, 7(1):1–11, 1936.

[5] 方兆本 and 缪柏其. *随机过程 (第三版)*. 中国科学技术大学出版社, 2011.

附录 A　运动模拟主程序代码

即bm.cpp，为模拟各向异性 Brownian 运动并绘图的主程序，编译时需要 OpenCV 库。

```
#include<stdio.h>
#include<cstdlib>
#include<opencv2\opencv.hpp>
#include "RNG.hpp"

#pragma warning(disable:4996)

const int imgSize = 600;
const int textHeight = 18;//height for ONE LINE of status bar draw on output
const int updateBarPosPeriod = 50;//same of movie output inteval
double barLen = 17.0;//Length of the green indicator of the bar
double Dx = 0.99, Dy = 0.01, Dtheta = 3e-2;//D_i
double windowScale = 5;//how much pixels 1 unit length correspond to
//fading coef (0->complete dispear after 1 single step 1->always here)
const double fadingMult = 0.99;
int MOVIE_WRITING_ENABLED = 0;
const std::string outputMoviePath = "out.avi";

const uint64_t DEFAULT_SEED0 = 15000034LL;
const uint64_t DEFAULT_SEED1 = 434343LL;

//Transform cord to the pixel to draw on
inline int cordToPixelX(double x) {
  return (int)(x*windowScale + imgSize / 2);
}
//Notice OpenCV imshow is UPSIDE-DOWN!
inline int cordToPixelY(double y) {
  return (int)(imgSize / 2 - y*windowScale);
}

void move(double& x, double& y, double& theta, RNG& rng);

int main(void) {
  RNG rng;
  double currTheta = 0;
  double currX = 0;
  double lastX = currX;
  double currY = 0;
  double lastY = currY;
  int timeStep = 0;
  char statusString[200];
  uint64_t seed1 = DEFAULT_SEED1;
//Image Buffer
  cv::Mat oBuffer(imgSize + 2 * textHeight, imgSize, CV_8UC3, cv::Scalar(0, 0, 0));
  cv::VideoWriter* outputVideoWriter = NULL;
//IO
  printf("Use default settings? [Y]/N\n");
  if (getchar() == 'N') {
    getchar();
    printf("Dx?");
    scanf("%lf", &Dx);
    getchar();
    printf("Dy?");
    scanf("%lf", &Dy);
    getchar();
```

```
    printf("Dtheta?");
    scanf("%lf", &Dtheta);
    getchar();
    printf("Window scale?");
    scanf("%lf", &windowScale);
    printf("Bar length?");
    scanf("%lf", &barLen);
    printf("seed1?");
    scanf("%lld", &seed1);
    getchar();
  }
  else {
    getchar();
  }
  barLen /= windowScale;//adjust the bar according to the window scale
  rng.setSeed(DEFAULT_SEED0, seed1);
  printf("Need video output? (Y/[N])\n");
  if (getchar() == 'Y') {
    //Asking for video output (MOVIE_WRITING_ENABLED initialized to 0£©
    MOVIE_WRITING_ENABLED = 1;
    outputVideoWriter = new cv::VideoWriter(outputMoviePath, -1, 30, cv::Size(imgSize,
        imgSize + 2 * textHeight), true);
  }
  if (MOVIE_WRITING_ENABLED) {
    if (!outputVideoWriter->isOpened())
    {
      printf("Output video could not be opened\n");
      return -1;
    }
  }
  cv::namedWindow("Display", cv::WINDOW_AUTOSIZE);
  //Put static label on output image
    //this line won't be changed in futher steps
  oBuffer(cv::Rect(0, imgSize, imgSize, textHeight)) = cv::Scalar(25, 25, 25);
  sprintf(statusString, "DX:%.2e DY:%.2e DT:%.2e WS:%.2f S1:%lld", Dx, Dy, Dtheta,
      windowScale, seed1);
  putText(oBuffer, statusString, cv::Point(0, imgSize + textHeight - 3), cv::
      FONT_HERSHEY_PLAIN, 1.2, cv::Scalar(250, 250, 250), 1, 8, false);
  //stop when the particle reached the edge...
    //not giving it chance to come back :)
    while (cordToPixelX(currX) && cordToPixelX(currX) < imgSize&&cordToPixelY(currY)
        && cordToPixelY(currY) < imgSize) {
        //Draw the path
    cv::line(oBuffer, cv::Point(cordToPixelX(lastX), cordToPixelY(lastY)),
      cv::Point(cordToPixelX(currX), cordToPixelY(currY)), cv::Scalar(255, 255, 255));
    //if we need output now...
        if (!(timeStep%updateBarPosPeriod)) {
      oBuffer(cv::Rect(0, 0, imgSize, imgSize)) *= fadingMult;
      //Updating the DYNAMIC status bar
            //The static bar is not changed
      oBuffer(cv::Rect(0, imgSize+ textHeight, imgSize, textHeight)) = cv::Scalar(25,
          25, 25);
      sprintf(statusString, "TS:%5d XP:%3.5f YP:%3.5f TH:%3.5f RUN", timeStep, currX,
          currY, currTheta);
      putText(oBuffer, statusString, cv::Point(0, imgSize + 2 * textHeight - 3), cv::
          FONT_HERSHEY_PLAIN, 1.2, cv::Scalar(250, 250, 250), 1, 8, false);
      //draw the nano bar
      cv::line(oBuffer, cv::Point(cordToPixelX(currX - barLen / 2 * cos(currTheta)),
```

```
            cordToPixelY(currY - barLen / 2 * sin(currTheta))),
        cv::Point(cordToPixelX(currX + barLen / 2 * cos(currTheta)), cordToPixelY(
            currY + barLen / 2 * sin(currTheta))), cv::Scalar(0, 255, 0));
    }
    cv::imshow("Display", oBuffer);
    cv::waitKey(1);
    if (MOVIE_WRITING_ENABLED&& !(timeStep%updateBarPosPeriod)) {
      outputVideoWriter->write(oBuffer);
    }
    lastX = currX; lastY = currY;
        //Simulate!
    move(currX, currY, currTheta, rng);
    timeStep++;
  }
  printf("Done\n");
  //Drawing status bar
  oBuffer(cv::Rect(0, imgSize + textHeight, imgSize, textHeight)) = cv::Scalar(25, 25,
      25);
  sprintf(statusString, "TS:%5d XP:%3.5f YP:%3.5f TH:%3.5f STOP", timeStep, currX,
      currY, currTheta);
  putText(oBuffer, statusString, cv::Point(0, imgSize + 2 * textHeight - 3), cv::
      FONT_HERSHEY_PLAIN, 1.2, cv::Scalar(250, 250, 250), 1, 8, false);
  cv::imshow("Display", oBuffer);
  if (MOVIE_WRITING_ENABLED) {
    outputVideoWriter->write(oBuffer);
    outputVideoWriter->release();
  }
  cv::waitKey(0);
  cv::destroyWindow("Display");
  return 0;
}

void move(double & x, double & y, double & theta, RNG & rng)
{
    //movement in body frame
  double bodyFrameMovX = Dx*rng.normalGen();
  double bodyFrameMovY = Dy*rng.normalGen();
    //doing cord transform
  x += bodyFrameMovX*cos(theta) - bodyFrameMovY*sin(theta);
  y += bodyFrameMovY*cos(theta) + bodyFrameMovX*sin(theta);
    //random walk of theta
  theta += Dtheta*rng.normalGen();
}
```

附录 B　式(4)定义的自相关函数计算程序代码

即main1.cpp，为计算式(4)定义的自相关函数的主程序。

```
#include<stdio.h>
#include<cstdlib>
#include<cmath>
#include<vector>
#include<deque>
#include "RNG.hpp"


#pragma warning(disable:4996)
//No matter how many particles used in simulation
```

```
//Only the 1st one is used for autocorrelation function calculation
const int PARTICLE_NUM = 1;
const int TIMESTEP_MAX = 100000000;
const int OUTPUT_INTEVAL = 500000;
const int MAX_AUTOCORR_CALC_LEN = 3000;
//output separation (eg.getting C(2),C(4),C(6)...C(3000) if set to 2)
const int AUTOCORR_CALC_INTEVAL = 1;

const char xPosOutFile[] = "xpos.csv";
const char yPosOutFile[] = "ypos.csv";
const char angOutFile[] = "thetapos.csv";

const char xCorrFuncOutFile[] = "xcorr.csv";
const char yCorrFuncOutFile[] = "ycorr.csv";
const char angCorrFuncOutFile[] = "angcorr.csv";

double Dx = 9.9;
double Dy = 1.0;
double Dtheta = 0.05;

const uint64_t DEFAULT_SEED0 = 15000034LL;
const uint64_t DEFAULT_SEED1 = 434343LL;

void move(double xPos[], double yPos[], double theta[], int numParticle, RNG& rng);

int main(void) {
  FILE* xPosOutFilePointer = fopen(xPosOutFile, "w");
  FILE* yPosOutFilePointer = fopen(yPosOutFile, "w");
  FILE* angOutFilePointer = fopen(angOutFile, "w");
  FILE* xCorrFuncOutFilePointer = fopen(xCorrFuncOutFile, "w");
  FILE* yCorrFuncOutFilePointer = fopen(yCorrFuncOutFile, "w");
  FILE* angCorrFuncOutFilePointer = fopen(angCorrFuncOutFile, "w");
  RNG rng;
    //Recording the recent steps of the particles using deque
  std::deque<double> lastStepsOf1stParticleX;
  std::deque<double> lastStepsOf1stParticleY;
  std::deque<double> lastStepsOf1stParticleCosTheta;
  double xPos[PARTICLE_NUM] = { 0 };
  double yPos[PARTICLE_NUM] = { 0 };
  double theta[PARTICLE_NUM] = { 0 };
  double xCorrSum[MAX_AUTOCORR_CALC_LEN] = { 0 };
  double xSqSum = 0;
  double xSum = 0;
  double yCorrSum[MAX_AUTOCORR_CALC_LEN] = { 0 };
  double ySqSum = 0;
  double ySum = 0;
  double cosThetaCorrSum[MAX_AUTOCORR_CALC_LEN] = { 0 };
  double cosThetaSqSum = 0;
  double cosThetaSum = 0;
  int timeStep = 0;
  int nCorrSum = 0;
  for (int nCorrPushed = 0; nCorrPushed < MAX_AUTOCORR_CALC_LEN; timeStep++) {
    double cosTheta0 = cos(theta[0]);
        //if we need output here?
    if (!(timeStep%AUTOCORR_CALC_INTEVAL)) {
      lastStepsOf1stParticleX.push_back(xPos[0]);
      lastStepsOf1stParticleY.push_back(yPos[0]);
      lastStepsOf1stParticleCosTheta.push_back(cosTheta0);
```

```
      nCorrPushed++;
    }
        //updating the sum
    xSum += xPos[0];
    xSqSum += xPos[0] * xPos[0];
    ySum += yPos[0];
    ySqSum += yPos[0] * yPos[0];
    cosThetaSum += cosTheta0;
    cosThetaSqSum += cosTheta0 * cosTheta0;
    //Run serveral steps without giving the output of corr func (not yet calced)
        //and of course no need to poping elements from the deque
    if (!(timeStep%OUTPUT_INTEVAL)) {
      printf("TS%d\n", timeStep);
      fprintf(xPosOutFilePointer, "%d,", timeStep);
      fprintf(yPosOutFilePointer, "%d,", timeStep);
      fprintf(angOutFilePointer, "%d,", timeStep);
      for (int n = 0; n < PARTICLE_NUM; n++) {
        fprintf(xPosOutFilePointer, "%.16f%c", xPos[n], (n < PARTICLE_NUM - 1) ? ',' : '\n');
        fprintf(yPosOutFilePointer, "%.16f%c", yPos[n], (n < PARTICLE_NUM - 1) ? ',' : '\n');
        fprintf(angOutFilePointer, "%.16f%c", theta[n], (n < PARTICLE_NUM - 1) ? ',' : '\n');
      }

    }
    move(xPos, yPos, theta, PARTICLE_NUM, rng);
  }
    //Run,Record and Pop front from here
  for (; timeStep <= TIMESTEP_MAX; timeStep++) {
    double cosTheta0 = cos(theta[0]);
    if (!(timeStep%AUTOCORR_CALC_INTEVAL)) {
      for (int n = 0; n < MAX_AUTOCORR_CALC_LEN; n++) {
        xCorrSum[n] += xPos[0] * lastStepsOf1stParticleX[MAX_AUTOCORR_CALC_LEN - n - 1];
        yCorrSum[n] += yPos[0] * lastStepsOf1stParticleY[MAX_AUTOCORR_CALC_LEN - n - 1];
        cosThetaCorrSum[n] += cosTheta0 * lastStepsOf1stParticleCosTheta[MAX_AUTOCORR_CALC_LEN - n - 1];
      }
      nCorrSum++;
            //Remove the earliest record and put the most recent one in
      lastStepsOf1stParticleX.pop_front();
      lastStepsOf1stParticleY.pop_front();
      lastStepsOf1stParticleCosTheta.pop_front();
      lastStepsOf1stParticleX.push_back(xPos[0]);
      lastStepsOf1stParticleY.push_back(yPos[0]);
      lastStepsOf1stParticleCosTheta.push_back(cosTheta0);
    }
    xSum += xPos[0];
    xSqSum += xPos[0] * xPos[0];
    ySum += yPos[0];
    ySqSum += yPos[0] * yPos[0];
    cosThetaSum += cosTheta0;
    cosThetaSqSum += cosTheta0 * cosTheta0;
    if (!(timeStep%OUTPUT_INTEVAL)) {
      printf("TS%d\n", timeStep);
      fprintf(xPosOutFilePointer,"%d,", timeStep);
```

```
      fprintf(yPosOutFilePointer, "%d,", timeStep);
      fprintf(angOutFilePointer, "%d,", timeStep);
      for (int n = 0; n < PARTICLE_NUM; n++) {
        fprintf(xPosOutFilePointer, "%.16f%c", xPos[n], (n < PARTICLE_NUM - 1) ? ',' : '\n');
        fprintf(yPosOutFilePointer, "%.16f%c", yPos[n], (n < PARTICLE_NUM - 1) ? ',' : '\n');
        fprintf(angOutFilePointer, "%.16f%c", theta[n], (n < PARTICLE_NUM - 1) ? ',' : '\n');
      }
            //flush the file streams for people eager to get early result
      fflush(xPosOutFilePointer); fflush(yPosOutFilePointer); fflush(angOutFilePointer);
      fprintf(xCorrFuncOutFilePointer, "%d,", timeStep);
      fprintf(yCorrFuncOutFilePointer, "%d,", timeStep);
      fprintf(angCorrFuncOutFilePointer, "%d,", timeStep);
      for (int n = 0; n < MAX_AUTOCORR_CALC_LEN; n++) {
        double xAvg = xSum / timeStep;
        double yAvg = ySum / timeStep;
        double cosThetaAvg = cosThetaSum / timeStep;
                //Calculating the corr func
        double xCorr = (xCorrSum[n] / nCorrSum - xAvg*xAvg) / (xSqSum / timeStep - xAvg*xAvg);
        double yCorr = (yCorrSum[n] / nCorrSum - yAvg*yAvg) / (ySqSum / timeStep - yAvg*yAvg);
        double cosThetaCorr = (cosThetaCorrSum[n] / nCorrSum - cosThetaAvg*cosThetaAvg) / (cosThetaSqSum / timeStep - cosThetaAvg*cosThetaAvg);
        fprintf(xCorrFuncOutFilePointer, "%.16f%c", xCorr, (n < MAX_AUTOCORR_CALC_LEN - 1) ? ',' : '\n');
        fprintf(yCorrFuncOutFilePointer, "%.16f%c", yCorr, (n < MAX_AUTOCORR_CALC_LEN - 1) ? ',' : '\n');
        fprintf(angCorrFuncOutFilePointer, "%.16f%c", cosThetaCorr, (n < MAX_AUTOCORR_CALC_LEN - 1) ? ',' : '\n');
      }
      fflush(xCorrFuncOutFilePointer); fflush(yCorrFuncOutFilePointer); fflush(angCorrFuncOutFilePointer);
    }
    move(xPos, yPos, theta, PARTICLE_NUM, rng);
  }
  fclose(xPosOutFilePointer);
  fclose(yPosOutFilePointer);
  fclose(angOutFilePointer);
  return 0;
}

void move(double xPos[], double yPos[], double theta[], int numParticle, RNG& rng)
{
    //movement in body frame
  double bodyFrameMovX;
  double bodyFrameMovY;
  for (int i = 0; i < numParticle; i++) {
    double cosThetaI = cos(theta[i]);
    double sinThetaI = sin(theta[i]);
    bodyFrameMovX = Dx*rng.normalGen();
    bodyFrameMovY = Dy*rng.normalGen();
        //doing cord transform
    xPos[i] += bodyFrameMovX *cosThetaI - bodyFrameMovY*sinThetaI;
    yPos[i] += bodyFrameMovY *cosThetaI + bodyFrameMovX*sinThetaI;
```

```
        //random walk of theta
    theta[i] += Dtheta*rng.normalGen();
  }
}
```

附录 C　式(7)定义的自相关函数计算程序代码

即main2.cpp，为计算式(7)定义的自相关函数的主程序。

```
#include<stdio.h>
#include<cstdlib>
#include<cmath>
#include<vector>
#include<deque>
#include<omp.h>
#include "RNG.hpp"


#pragma warning(disable:4996)

const int PARTICLE_NUM = 5000000;//num of particles used in simulation
const int TIMESTEP_MAX = 20000;//limit of thimesteps
const int OUTPUT_INTEVAL = 1;

const char OutFile[] = "out3e-2m.csv";

//D_i
double Dx = 0.99;
double Dy = 0.01;
double Dtheta = 3e-2;

const uint64_t DEFAULT_SEED0 = 15000034LL;
const uint64_t DEFAULT_SEED1 = 434343LL;

void move(double xPos[], double yPos[], double theta[], RNG rng[], int numParticle);

double cosThetaCorr(double theta[], double initCosTheta[], int numParticle);

int main(void) {
  FILE* OutFilePointer = fopen(OutFile, "w");
  double xSqAvg, ySqAvg, thetaSqAvg, xSqAvgcos2t, xSqAvgcos2tUnCor, ySqAvgcos2t,\
    ySqAvgcos2tUnCor, xySin2tm2, xySin2tm2UnCor, nonGaussianPara;
  RNG rngInit;
  rngInit.setSeed(DEFAULT_SEED0, DEFAULT_SEED1);
  double *xPos = new double[PARTICLE_NUM];
  for (int n = 0; n < PARTICLE_NUM; n++) {
    xPos[n] = 0;
  }
  double *yPos = new double[PARTICLE_NUM];
  for (int n = 0; n < PARTICLE_NUM; n++) {
    yPos[n] = 0;
  }
  double *theta = new double[PARTICLE_NUM];
  double *initCosTheta = new double[PARTICLE_NUM];//for theta(0)s
  for (int n = 0; n < PARTICLE_NUM; n++) {
    theta[n] = rngInit.gen()*2*M_PI;//uniform distribution of initial angle from 0 to
        2pi
    initCosTheta[n] = cos(theta[n]);//prepare the cos(theta(0))s
```

```
  }
  RNG *rngs = new RNG[PARTICLE_NUM];
  //Generating seeds
  //use the generated random number from rngs as seeds
  for (int n = 0; n < PARTICLE_NUM; n++) {
    long long int seed1, seed2;
    //make sure the generated seeds are non-zero
    //otherwise always try next one
    while (!(seed1 = rngInit.rawGen()));
    while (!(seed2 = rngInit.rawGen()));
    rngs[n].setSeed(seed1, seed2);
  }
  int timeStep = 1;
  for (; timeStep <= TIMESTEP_MAX; timeStep++) {
    //Simulate!
    move(xPos, yPos, theta,rngs, PARTICLE_NUM);
    //if we need output now...
    if (!(timeStep%OUTPUT_INTEVAL)) {
      printf("TS%d\n", timeStep);
      fprintf(OutFilePointer, "%d,%.16f\n",timeStep, cosThetaCorr(theta, initCosTheta,
          PARTICLE_NUM));
      fflush(OutFilePointer);
    }
  }
  fclose(OutFilePointer);
  delete[] xPos, yPos, theta, initCosTheta;
  return 0;
}

//the general simulation code (using OpenMP)
void move(double xPos[], double yPos[], double theta[],RNG rng[],int numParticle)
{
#pragma omp parallel for
  for (int i = 0; i < numParticle; i++) {
        //movement in body frame
    double cosThetaI = cos(theta[i]);
    double sinThetaI = sin(theta[i]);
    double bodyFrameMovX = Dx*rng[i].normalGen();
    double bodyFrameMovY = Dy*rng[i].normalGen();
        //doing cord transform
    xPos[i] += bodyFrameMovX *cosThetaI - bodyFrameMovY*sinThetaI;
    yPos[i] += bodyFrameMovY *cosThetaI + bodyFrameMovX*sinThetaI;
        //random walk of theta
    theta[i] += Dtheta*rng[i].normalGen();
  }
}

//subrotine used for calc of <u_x(0)u_x(t)>
double cosThetaCorr(double theta[], double initCosTheta[], int numParticle)
{
  double cosThetaCorrSum = 0.0;
#pragma omp parallel for reduction(+:cosThetaCorrSum)
  for (int n = 0; n < numParticle; n++) {
    cosThetaCorrSum += cos(theta[n])*initCosTheta[n];
  }
  return(cosThetaCorrSum / numParticle);
}
```

附录 D　其他统计量计算程序代码

即main3.cpp，为计算式(7)定义的自相关函数的主程序。

```
#include<stdio.h>
#include<cstdlib>
#include<cmath>
#include<vector>
#include<deque>
#include<omp.h>
#include "RNG.hpp"

//kill stupid CRT security check of Visual C++
#pragma warning(disable:4996)

const int PARTICLE_NUM = 2500000;//num of particles used in simulation
const int TIMESTEP_MAX = 20000;//limit of thimesteps
const int OUTPUT_INTEVAL = 1;
const char OutFile[] = "out5e-2m.csv";

//D_i
double Dx = 0.99;
double Dy = 0.01;
double Dtheta = 5e-2;

const uint64_t DEFAULT_SEED0 = 15000034LL;
const uint64_t DEFAULT_SEED1 = 434343LL;

void move(double xPos[], double yPos[], double theta[], RNG rng[], int numParticle);

void calc(int time, double x[], double y[], double theta[], int numParticle, \
  double & xSqAvg, double & ySqAvg, double&thetaSqAvg, double & xSqAvgcos2t, double&
      xSqAvgcos2tUnCor, \
  double & ySqAvgcos2t, double&ySqAvgcos2tUnCor, double & xySin2tm2, double &
      xySin2tm2UnCor, double &nonGaussianPara);

int main(void) {
  FILE* OutFilePointer = fopen(OutFile, "w");
  double xSqAvg, ySqAvg, thetaSqAvg, xSqAvgcos2t, xSqAvgcos2tUnCor, ySqAvgcos2t,\
    ySqAvgcos2tUnCor, xySin2tm2, xySin2tm2UnCor, nonGaussianPara;
  RNG rngInit;
  rngInit.setSeed(DEFAULT_SEED0, DEFAULT_SEED1);
  double *xPos = new double[PARTICLE_NUM];
  for (int n = 0; n < PARTICLE_NUM; n++) {
    xPos[n] = 0;
  }
  double *yPos = new double[PARTICLE_NUM];
  for (int n = 0; n < PARTICLE_NUM; n++) {
    yPos[n] = 0;
  }
  double *theta = new double[PARTICLE_NUM];
  for (int n = 0; n < PARTICLE_NUM; n++) {
    theta[n] = 0;
  }
  RNG *rngs = new RNG[PARTICLE_NUM];
  for (int n = 0; n < PARTICLE_NUM; n++) {
    long long int seed1, seed2;
    while (!(seed1 = rngInit.rawGen()));
    while (!(seed2 = rngInit.rawGen()));//make sure the seeds are non-zero
```

```
    rngs[n].setSeed(seed1, seed2);//use the generated random number from rngs as seeds
  }
  int timeStep = 1;
  for (; timeStep <= TIMESTEP_MAX; timeStep++) {
    move(xPos, yPos, theta,rngs, PARTICLE_NUM);
    calc(timeStep, xPos, yPos, theta, PARTICLE_NUM, xSqAvg, ySqAvg, thetaSqAvg,\
      xSqAvgcos2t,xSqAvgcos2tUnCor, ySqAvgcos2t,ySqAvgcos2tUnCor, xySin2tm2,
          xySin2tm2UnCor, nonGaussianPara);
    if (!(timeStep%OUTPUT_INTEVAL)) {
      printf("TS%d\n", timeStep);
      fprintf(OutFilePointer, "%d,%.16f,%.16f,%.16f,%.16f,%.16f,%.16f,%.16f,%.16
          f,%.16f\n",\
        timeStep, xSqAvg, ySqAvg, thetaSqAvg, xSqAvgcos2t, xSqAvgcos2tUnCor,\
        ySqAvgcos2t, ySqAvgcos2tUnCor, xySin2tm2, xySin2tm2UnCor, nonGaussianPara);
      fflush(OutFilePointer);
    }
  }
  fclose(OutFilePointer);
  delete[] xPos, yPos, theta;
  return 0;
}

void move(double xPos[], double yPos[], double theta[],RNG rng[],int numParticle)
{
#pragma omp parallel for
  for (int i = 0; i < numParticle; i++) {
    double cosThetaI = cos(theta[i]);
    double sinThetaI = sin(theta[i]);
    //movement in body frame
    double bodyFrameMovX = Dx*rng[i].normalGen();
    double bodyFrameMovY = Dy*rng[i].normalGen();
    bodyFrameMovY = Dy*rng.normalGen();
        //doing cord transform
    xPos[i] += bodyFrameMovX *cosThetaI - bodyFrameMovY*sinThetaI;
    yPos[i] += bodyFrameMovY *cosThetaI + bodyFrameMovX*sinThetaI;
        //random walk of theta
    theta[i] += Dtheta*rng[i].normalGen();
  }
}

//subrotine used for calc of all the statistics
//all we need is calculated here for best efficiency
//more modular -> more times of evaluating same thing :(
void calc(int time, double x[], double y[], double theta[], int numParticle,\
  double & xSqAvg, double & ySqAvg, double & thetaSqAvg, double & xSqAvgcos2t,\
  double & xSqAvgcos2tUnCor, double & ySqAvgcos2t, double & ySqAvgcos2tUnCor,\
  double & xySin2tm2, double & xySin2tm2UnCor, double & nonGaussianPara)
{
  double xSqSum = 0.0, ySqSum = 0.0, thetaSqSum = 0.0, xSqC2tSum = 0.0,xySum=0.0;
  double ySqC2tSum = 0.0, xySin2tSum = 0.0, xQuadSum = 0.0, cos2tSum = 0.0, sin2tSum =
      0.0;
#pragma omp parallel for reduction(+:xSqSum,ySqSum,xySum,thetaSqSum,xSqC2tSum,
    ySqC2tSum,xySin2tSum,xQuadSum,cos2tSum,sin2tSum)
  for (int n = 0; n < numParticle; n++) {
    double currXSqSum = x[n] * x[n];
    double currXQuadSum = currXSqSum*currXSqSum;
    double currYSqSum = y[n] * y[n];
    double c2t = cos(theta[n] * 2.0);
```

```
    double s2t = sin(theta[n] * 2.0);
    double currXY = x[n] * y[n];
    xSqSum += currXSqSum;
    xQuadSum += currXQuadSum;
    ySqSum += currYSqSum;
    thetaSqSum += theta[n] * theta[n];
    xSqC2tSum += currXSqSum*c2t;
    ySqC2tSum += currYSqSum*c2t;
    xySin2tSum += currXY * s2t;
    xySum += currXY;
    cos2tSum += c2t;
    sin2tSum += s2t;
  }
  xSqAvg = xSqSum / time / numParticle;
  ySqAvg = ySqSum / time / numParticle;
  thetaSqAvg = thetaSqSum / time / numParticle;
  xSqAvgcos2t = xSqC2tSum / time / numParticle;
  xSqAvgcos2tUnCor = xSqAvg*cos2tSum / numParticle;
  ySqAvgcos2t = ySqC2tSum / time / numParticle;
  ySqAvgcos2tUnCor = ySqAvg*cos2tSum / numParticle;
  xySin2tm2 = 2 * xySin2tSum / time / numParticle;
  xySin2tm2UnCor = 2 * xySum*sin2tSum / numParticle / numParticle / time;
  nonGaussianPara = (xQuadSum * numParticle - 3 * xSqSum * xSqSum)\
    / (3 * xSqSum * xSqSum);
}
```

附录 E　RNG 随机数发生模块代码

即RNG.hpp，为 64 位xorshift+128 伪随机数发生模块，封装成RNG 类，各个程序中用其产生服从正态分布的随机数，编译其他程序均需要此模块。

```
/*
General purpose 64-bits Random Number Generator Class
Implementation of xorshift*128
*/

#include <cstdint>
#define _USE_MATH_DEFINES
#include <math.h>
#include <limits.h>

//----------------Class Def-----------------

class RNG {
public:
  RNG(uint64_t seed0 = 15000034, uint64_t seed1 = 43) {
    seed[0] = seed0;
    seed[1] = seed1;
  }
  inline void setSeed(uint64_t seed0 = 15000034, uint64_t seed1 = 43) {
    seed[0] = seed0;
    seed[1] = seed1;
  }
  inline double gen() {
    return (double)rawGen() / _UI64_MAX;
  }

```

```
  inline uint64_t rawGen() {
    uint64_t x = seed[0];
    uint64_t const y = seed[1];
    seed[0] = y;
    x ^= x << 23; // a
    seed[1] = x ^ y ^ (x >> 17) ^ (y >> 26); // b, c
    return seed[1] + y;
  }

  double normalGen();
  double lorentzGen();
  double logisticGen();
  double betaa1Gen(double a);
  double sqGen();
  double expGen();
  int bernGen(double p);
  int binoGenSmallN(int n, double p);
  double binoGenSmallNWrappedDoub(double p, double n);
  int binoGenBigN(int n, double p);
  double binoGenBigNWrappedDoub(double p, double n);
  int geomGen(double p);
/* The state must be seeded so that it is not everywhere zero. */
  uint64_t seed[2] = { 15000034,43 };
};


typedef double(RNG::*distrGenPointerVoid)(void);
typedef double(RNG::*distrGenPointer1Double)(double);
typedef double(RNG::*distrGenPointer2Double)(double, double);

//----------Standard Normal Distrib Generator----------------
//TWO random number is consumed by each call
//Using Box-Muller Formula x=\sqrt{-2\ln u}\cos 2\pi v
inline double RNG::normalGen() {
  double u = gen();
  double v = gen();
  return sqrt(-2.0 * log(u)) * cos(2.0 * M_PI * v);
}

//----------Lorentz (Cauchy) Distrib Generator--------------
//ONE random number is consumed by each call and called tan once
//Using Formula x=\tan(\pi u-\pi/2)
inline double RNG::lorentzGen() {
  double u = gen();
  return tan(M_PI*u - M_PI / 2);
}

//----------Logistic Distrib Generator--------------
//ONE random number is consumed by each call and called log once
//Using Formula x=\ln(\frac{u}{1-u})
inline double RNG::logisticGen() {
  double u = gen();
  return log(u / (1 - u));
}

//----------Beta(a,1) Distrib Generator--------------
//ONE random number is consumed by each call and called pow once
//PDF is x^{a-1}/a (0<=x<=1)
```

```
inline double RNG::betaa1Gen(double a) {
  double u = gen();
  return pow(u,1/a);
}
//Special case with a=2
inline double RNG::sqGen() {
  return betaa1Gen(2);
}

//----------Exp Distrib Generator--------------
//ONE random number is consumed by each call and called log once
//LAMBDA IS FIXED TO 1, DO SCALLING ( /lambda) TO MODIFY
//PDF is \lambda*\exp(-\lambda*x)  (x>=0)

inline double RNG::expGen() {
  return -log(gen());
}

//----------Bernoulli Distrib Generator--------------
//ONE random number is consumed by each call
//PDF is p^x * (1 - p)^x   x=0 or x=1

inline int RNG::bernGen(double p) {
  return (gen() < p) ? 1 : 0;
}

//----------Binomial Distrib Generator (for small n)--------------
//n random number is consumed by each call
//PDF is p^x * (1 - p)^(n - x)\binom{n}{x}   x=0 or x=1

int RNG::binoGenSmallN(int n,double p) {
  int cnt = 0;
  for (int m = 0; m < n; m++) {
    cnt += bernGen(p);
  }
  return cnt;
}

inline double RNG::binoGenSmallNWrappedDoub(double n,double p) {
  return binoGenSmallN(n, p);
}

//----------Binomial Distrib Generator (for big n or small p)--------------
//PDF is p^x * (1 - p)^(n - x)\binom{n}{x}   x=0 or x=1

int RNG::binoGenBigN(int n,double p ) {
  int cnt = 0;
  double c = log(1 - p);
  int s;
  double u;
  u = gen();
  s = ceil(log(u) / c);
  while (s < n + 1) {
    u = gen();
    s += ceil(log(u) / c);
    cnt++;
  }
  return cnt;
```

```
}

inline double RNG::binoGenBigNWrappedDoub(double n,double p) {
  return binoGenBigN(n, p);
}


//----------Geometric Distrib Generator--------------
//PDF is (1-p)^(x-1)*p x=1,2,3...

inline int RNG::geomGen(double p) {
  return ceil(expGen() / (-log(1 - p)));
}
```