Biasing Brownian motion with a flashing ratchet potential to seperate particles by size

Fredrik Knapskog[a]

[a]*Institute for physics, Norwegian University of Science and Technology, N-7491 Trondheim, Norway.*

Abstract

DNA-molecules were separated with a flashing ratchet potential to bias the Brownian motion as in [1] with values taken from the real experiment [2]. The Langevin approach was used with a forward Euler scheme. The density distributions evolved in time as described by the Fokker-Planck equation. The optimal flashing period was found as 0,4086 s.

1. Intoduction

In chemical and biological analysis, separation of a solution's constituents can be of interest if studying a specific constituent's property or estimating the quantities of the constituents is desired. Separation processes exploits differences in chemical or physical properties such as mass, density, binding affinity, size and shape. Some separation processes are chromatography, centrifugation, flotation, filtration, distillation and electrophoresis. The last method involves particles dispersed in a fluid and put under a spatially uniform electric field in order to separate the particles by size. The electric field makes charged particles drift, and their sizes affects the drift velocities. Electrophoresis is mostly used for separating macromolecules such as the DNA-molecule. The potential used for migrating the particles does not have to be electric, but can be gravitational too as in [1]. Here a flashing ratchet potential as viewed in Figure 1 is used to bias the Brownian motion and thus separate the particles from different drift velocities. This article deals with a computer experiment of the concept shown in Figure 2 in [1] with relevant parameters from the real experiment [2]. That is, the Langevin approach was used with the Euler method to separate a mixture of two types of DNA molecules with different sizes.

2. Theory

The ratchet potential, $U_r(x)$, is a periodic asymmetric saw-tooth potential of period, L,

$$U_r(x) = \begin{cases} \frac{x}{\alpha L}\Delta U & \text{if } 0 \leq x < \alpha L \\ \frac{L-x}{L(1-\alpha)}\Delta U & \text{if } \alpha L \leq x < L \end{cases}. \tag{1}$$

The asymmetry factor, α, is a number between 0 and 1. ΔU is the amplitude of the potential and x is the spatial position. In order to get a flashing ratchet potential, U, the ratchet potential, U_r, must be multiplied by a flashing function, f,

$$U(x,t) = U_r(x)f(t). \tag{2}$$

$f(t)$ is an asymmetric square signal in time, t, of period, τ,

$$f(t) = \begin{cases} 0 & \text{if } 0 \leq t < \beta\tau \\ 1 & \text{if } \beta\tau \leq t < \tau \end{cases}. \tag{3}$$

Like α is β also an asymmetry factor between 0 and 1.

The Langevin approach here deals with spherical molecules undergoing Brownian motion in a solvent and reduced to one dimension. These molecules, also referred to as particles, are subjected to collisions with the solvent molecules, friction and the potential from (2). Assuming each particle can be treated independently, the motion of a particle with radius, r, and mass, m, can be described by Newton's second law

$$m\frac{d^2x}{dt^2} = -\frac{\partial U}{\partial x}(x,t) - \gamma\frac{dx}{dt} + \xi(t). \tag{4}$$

Here the obesrvation time is much larger than the characteristic damping time. Hence the inertial term can be neglected. The Langevin equation (4) is thus reduced to

$$\gamma\frac{dx}{dt} = -\frac{\partial U}{\partial x}(x,t) + \xi(t), \tag{5}$$

with the friction constant, $\gamma = 6\pi\eta r$, where η is the viscosity of the solvent. ξ is a random stochastic variable modelling collisions between the particles and the fluid molecules. ξ is assumed to be an uncorrelated white noise, so from the fluctuation-dissipation theorem

$$\langle\xi(t)\rangle = 0, \quad \langle\xi(t)\xi(t')\rangle = 2\gamma k_B T\delta(t-t'). \tag{6}$$

T is the temperature of the solvent and k_B is Boltzmann constant.

The probability distribution, p, of a free particle in a solvent, exhibits diffusion

$$p(x,t) = \frac{1}{\sqrt{4\pi Dt}}\exp\left(-\frac{x^2}{4Dt}\right), \tag{7}$$

where $D = k_BT/\gamma$ is the diffusion constant. If the particle is subjected to a potential however, the probability distribution becomes a Boltzmann distribution

$$p(U) = \frac{e^{-\frac{U}{k_B T}}}{k_B T(1 - e^{-\frac{\Delta U}{k_B T}})}. \tag{8}$$

The density distribution n is achieved by multiplying the probability density, p, with the number of particles, N.

3. Method

The forward Euler scheme in reduced units was used for (5)

$$\hat{x}_{n+1} = \hat{x}_n - \frac{\partial \hat{U}}{\partial \hat{x}}(\hat{x}_n, \hat{t}_n)\delta\hat{t} + \sqrt{2\hat{D}\delta\hat{t}}\hat{\xi}_n, \tag{9}$$

where

$$\hat{x} = \frac{x}{L}, \quad \hat{t} = \omega t, \quad \omega = \frac{\Delta U}{\gamma L^2}, \tag{10}$$

$$\hat{U}(\hat{x}, \hat{t}) = \frac{U(x,t)}{\Delta U}, \quad \hat{D} = \frac{k_B T}{\Delta U}. \tag{11}$$

The time step, δt, has to be sufficiently small in order to prevent an iteration from jumping through several variations in the potential. That is

$$|x_{n+1} - x_n| << \alpha L. \tag{12}$$

As $\left|\hat{\xi}_n\right|$ is Gaussian distributed it is more than a 99,99% probability for $\left|\hat{\xi}_n\right| < 4$. Hence the time step must satisfy

$$\frac{1}{\gamma}\max\left|\frac{\partial U}{\partial x}\right|\delta t + 4\sqrt{\frac{2k_B T\delta t}{\gamma}} << \alpha L. \tag{13}$$

The values from [2] are $L = 20\,\mu\text{m}$, $\alpha = 0,2$, $\beta = 0,75$, $\eta = 1\,\text{mPa s}$, $k_B T = 26\,\text{meV}$, $\Delta U = 80\,\text{eV}$, $r_1 = 12\,\text{nm}$ and $r_2 = 3r_1 = 36\,\text{nm}$. δt is then chosen as $20\,\mu\text{s}$ or smaller. As the derivative of the potential can be found analytically and remains constant over such large areas when the time step is small, a first order finite difference method is sufficient here. Finally, instead of changing the radius for the larger DNA molecule, the time scale, ω, is simply divided by the size ratio which here is 3.

The language to compute the simulation here is python, run from a jupyter console. For $\hat{\xi}_n$, numpy's built in function, normal, from the random library was used. numpy.random.normal has a mean, μ, of 0, a standard deviation, σ, of 1 and is Gaussian distributed. Furthermore numba's mode njit was used to decrease running time by translating written code to machine code.

4. Results

The density distributions from (7) and (8) are shown in Figure 3 and 4 respectively. The distribution in Figure 3 is the distribution when the potential is about to be activated for $\tau = 0{,}4\,\text{s}$. The distribution of radius r_2 has narrower peak than the one of radius r_1 as they diffuse slower. The fraction to the right of the right dashed line will be pushed towards right, and for a particle to be pushed towards left it would have to cross the left dashed line. Based on 1 million particles, for the particles of radius r_1, 11,44% cross the right dashed line, while for the particles of radius r_2, only 1,86% make it over before the potential turns on. The analytical distributions from (7) matches almost perfectly.

The distribution in Figure 4 however depends more on the time step, δt. The smaller δt is, the closer the numerical distribution is to the analytical distribution in (8). Still the entire distribution is located within a very small fraction of the of the potential well. As the distribution converges sufficiently fast the particles will be drawn to the bottom of the energy well long before the potential

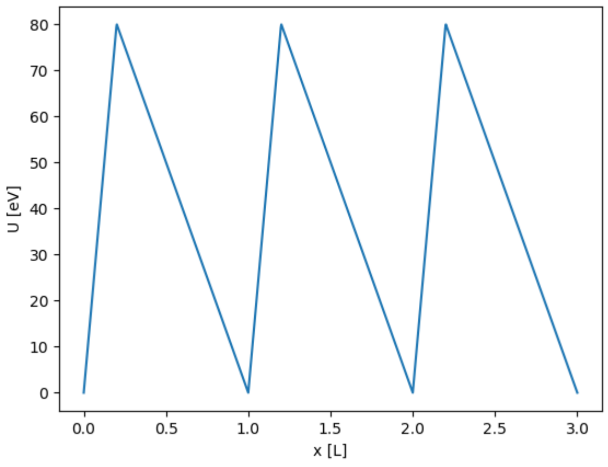

Figure 1: The ratchet potential from (1) with $\Delta U = 80\,\text{eV}$ and $\alpha = 0,2$.

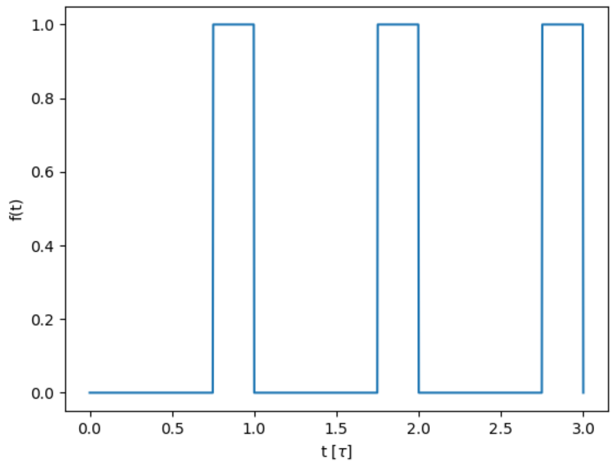

Figure 2: The flashing function from (3) with $\beta = 0,75$.

turns off again. If that wasn't the case the particles would not be able to drift towards right at the same pace.

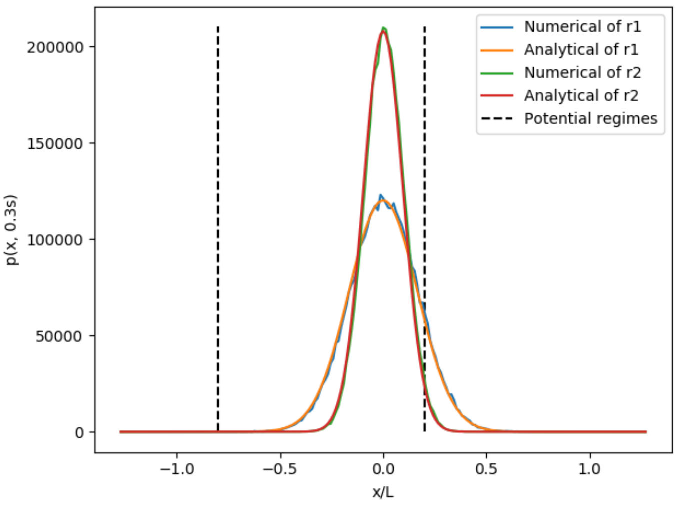

Figure 3: The diffusion probability distribution from (7) with $D = 3{,}25 \cdot 10^{-4}$ at time $t = 0{,}3\,\mathrm{s}$ for particles of two sizes.

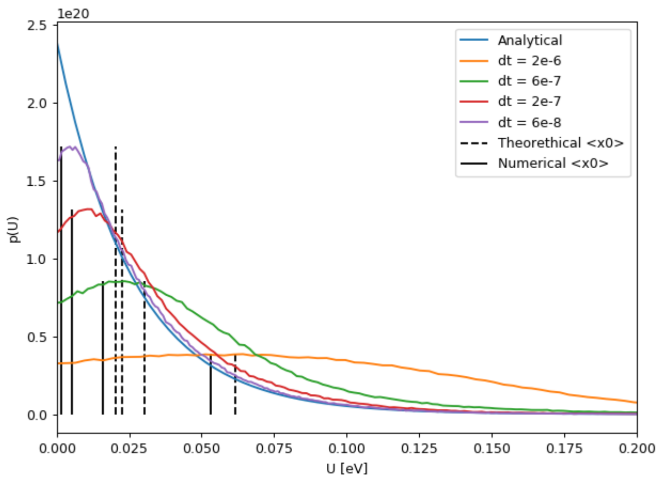

Figure 4: The Boltzmann distribution from (8) with k_BT 26 meV and $\Delta U = 80\,\mathrm{eV}$. The distributions are for several δt, and their corresponding $\langle x_0 \rangle$ are shown as vertical lines.

In Figure 5 the average drift velocity, $\langle v \rangle$, was computed for 20 values for τ ranging from 0,1 to 2,0. The plot takes the same form as Figure 2B in [1]. The optimal flashing period was found to be $\tau_{op} = 0{,}4\,\mathrm{s}$, and the maximum average velocity, $\langle v_{max} \rangle = 5{,}65\,\mathrm{\mu m\,s^{-1}}$. The average drift velocity in Figure 3 in [2] is approximately $5Lf/20$, with frequency, $f = 0{,}7\,\mathrm{Hz}$, resulting in $3{,}5\,\mathrm{\mu m\,s^{-1}}$. The average velocity corresponding to $f = 0{,}7\,\mathrm{Hz}$ in Figure 5 is $3{,}61\,\mathrm{\mu m\,s^{-1}}$, which suits well.

The trajectory of two particles is simulated in Figure 6. They're simulated at τ_{op} to time $t = 80\,\mathrm{s}$, which is 200 cycles. Here the smallest particle travels further than the

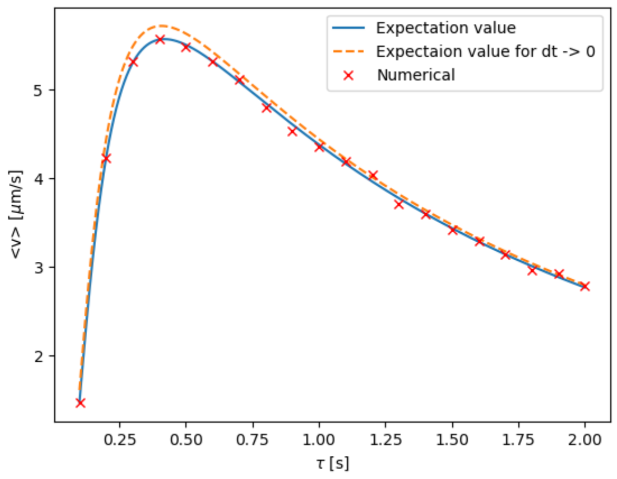

Figure 5: The red crosses are numerically computed average velocities for different values of τ, each value computed for $t = 10\,\mathrm{ks}$. The blue line is the expectation values from (17) for the average velocities, and the dashed line is the expectation value when δt approaches zero.

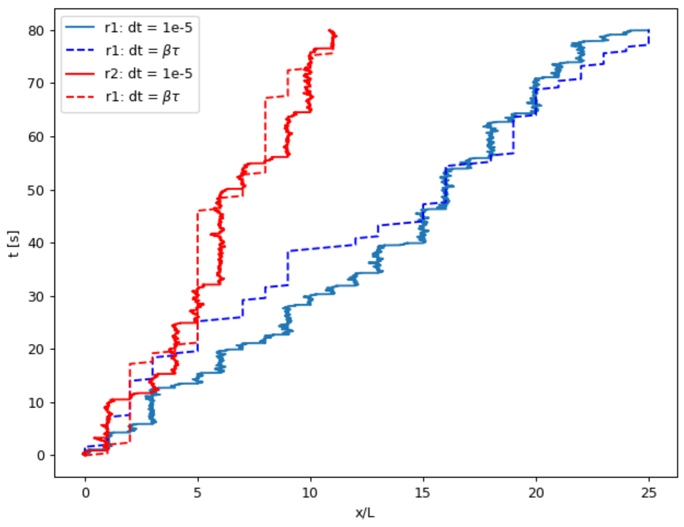

Figure 6: Two particles of radius r_1 and r_2 simulated for 20 cycles at τ_{op}. The dashed lines are only simulated through diffusion stages.

larger one. An ensamble of 20.000 of each simulation as in Figure 6 results in the density distribution in 7. The spikes are caused by the particles being trapped in local minimas at time t. The particles of radius r_1 has diffused more, and are advected further to the right. The differential equation describing the time evolution of the density distributions is thus the convection-diffusion equation, or more specific within this field, the Fokker-Planck equation. The two waves overlap a little bit in the middle, but are mostly separated. For even more cycles the waves will be about completely distinctive.

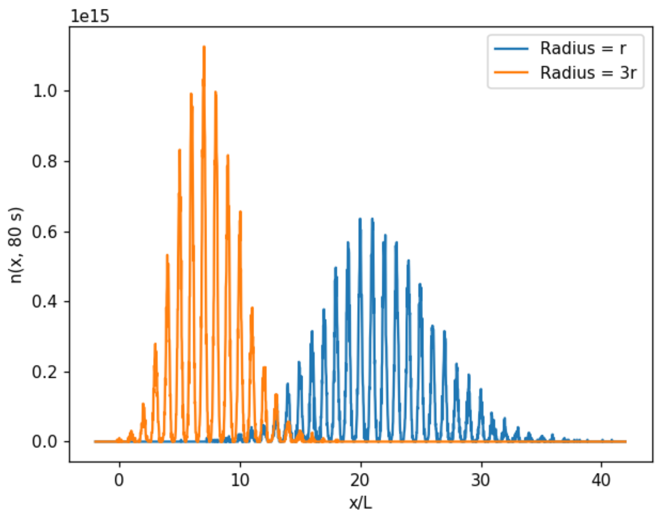

Figure 7: Density distribution of 20.000 particles of each size simulated 200 cycles at τ_{op}.

5. Discussion

The fraction of particles having crossed the right dashed line in Figure 5 when the potential is turned on determines the average velocity of a substance. Larger flashing periods will also allow for particles to travel far enough backwards to enter a previous energy well. The average velocity could then be a the difference between those fractions multiplied by one length, L, divided by one period, τ. However, according to Figure 4, the numerical particles will not stay at rest at the bottom of the energy well when the potential is turned on. The particles will rather oscillate a distance d, depending on δt

$$d = \frac{1}{\gamma}\left[\left|\frac{\partial U}{\partial x}\right|_{x=0^-}\right| + \left|\frac{\partial U}{\partial x}\right|_{x=0^+}\left|\right]\delta t. \tag{14}$$

As the potential is asymmetric, the mean position, $\langle x_0\rangle$, at which the particles start from when the potential is turned off again, is not 0. $\langle x_0\rangle$ is in fact skewed a distance

$$\langle x_0\rangle = \frac{1}{2\gamma L}\left[\frac{\partial U}{\partial x}\Big|_{x=0^-} + \frac{\partial U}{\partial x}\Big|_{x=0^+}\right]\delta t. \tag{15}$$

The average diffusion density distribution is then shifted the distance, $\langle x_0\rangle$ away from origin. Hence $\langle v\rangle$ becomes

$$\langle v(\tau,\alpha,\beta,D,\delta t)\rangle = \frac{L}{\tau}\left[p(\hat{x} > \alpha, t = \beta\tau) - p(\hat{x} < 1-\alpha, t = \beta\tau)\right] \tag{16}$$

$$= \frac{L}{\tau\sqrt{4\pi\hat{D}\beta\hat{\tau}}}\left[\int_{\alpha}^{\infty}\exp\left(-\frac{(\hat{x}-\langle\hat{x}_0\rangle)^2}{4\hat{D}\beta\hat{\tau}}\right)d\hat{x} - \int_{-\infty}^{\alpha-1}\exp\left(-\frac{(\hat{x}-\langle\hat{x}_0\rangle)^2}{4\hat{D}\beta\hat{\tau}}\right)d\hat{x}\right], \tag{17}$$

given that no particles passes by more than one energy well during a cycle. The function used to compute the integrals was scipy.stats.norm. The expectation values for $\langle v\rangle$ are plotted in Figure 5 for $\delta t = 1\cdot 10^{-5}\,\mathrm{s}$ and thus $\langle x_0\rangle = -2{,}656\cdot 10^{-3}\cdot L$ and $\delta t = 0$ and then $\langle x_0\rangle = 0$ along with the numerically computed values. Here they match about perfectly. For smaller δt such as in Figure 4 on the other hand, the simplification in (15) breaks down. The dashed lines are placed further away from the solid lines, the lower δt is. Figure 8 shows how the relative error of $\langle x_0\rangle$ varies with δt. As the time step becomes very small the force term from the collisions becomes larger than the term from the central force as shown by the orange line. Thus the mean position is more disturbed relatively by collisions and is pushed further away from origin. Nevertheless the

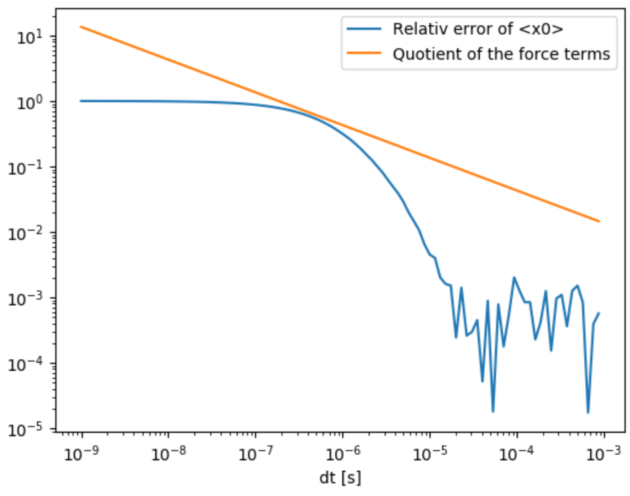

Figure 8: The blue line is the relative error for the theoretical $\langle x_0\rangle$ from (15). The orange line is the collision force term divided by the central force term, $\sqrt{\frac{2k_BT\delta t}{\gamma}}\sigma/(\frac{1}{\gamma}\max\left|\frac{\partial U}{\partial x}\right|\delta t)$

numerical $\langle x_0\rangle$ becomes very close to 0 for small δt, and the boltzmann distributions in Figure 4 matches the analytical one better for smaller δt. Thus the analytical $\langle x_0\rangle$ for a real experiment may be zero. The time step could then be set to $\beta\tau$ during the diffusion process, and when the potential is on the particles could simply be placed in their respective potential minimas assuming that the potential strength is sufficient. The dashed lines in Figure 6 are computed by this method and gives just as good result as the ones simulated every step. The dashed lines can be considered more accurate as well as their $\langle x_0\rangle$ are 0. They also shorten the computation power needed by at least 10.000 times.

The optimal flashing period, τ_{op}, can be found by taking the derivative of (17) equal to 0 and solving for τ. Neglecting the second integral yields

$$\int_{\alpha}^{\infty} \left[\frac{(\hat{x} - \langle \hat{x}_0 \rangle)^2}{4\hat{D}\beta\hat{\tau}} - \frac{3}{2} \right] \exp{(-\frac{(\hat{x} - \langle \hat{x}_0 \rangle)^2}{4\hat{D}\beta\hat{\tau}})} d\hat{x} = 0. \quad (18)$$

To solve this integral the function scipy.integrate.quad was used. For $\langle \hat{x}_0 \rangle = 0$, τ_{op} was determined as 0,4086 s which corresponds well with the peak in Figure 5. In [1] the optimal *tau* for largest amount of particles crossing α during one cycle without going backwards is

$$\tau_{op,flux} = \frac{1}{\beta} \frac{\alpha^2}{2D}, \quad (19)$$

which here equals 0,58 s. This is within the same magnitude as in (18), but here it is not taken into consideration that the shorter time period, the more cycles are completed per time. Also their α may have been 0,18. As (19) is higher than (18), the neglection of the second integral is justified.

Considering all this the numerical scheme works well according to theory. The theory however is not as precise when it comes to the distance, d from (14), and thus $\langle x_0 \rangle$ from (15). It is difficult to determine how large these oscillations in the well bottom should be experimentally, but the case where $\delta t \to 0$ can be used as a maximum value for the average drift velocities as in Figure 5.

Other simplifications from the Langevin approach may however cause differences from the experimental values. The friction coefficients may be somewhat inaccurate, especially when assuming the DNA-molecules take shapes as spheres. The random collision term may be stronger or weaker or just not behave as uniformly as the random function used here. Neglecting the inertial term could also affect the results a bit. When it comes to generating random numbers numpy has amongst the best algorithms for such. Here the variance of 100.000 samples from the normal function had a variance of 1.003 which is rather low.

6. Conclusion

The numerically computed values for average drift velocities matched well with both the experimental ones from [2] and the analytical ones from (17). There is thus much trust to the computed results. The error in the computation seems to be smaller to the analytical results than to the experimental ones. The simplifications in the theory may be of cause for this. The time evolution of the density distribution for large substances is well described by the Fokker-Planck equation.

References

[1] R. Dean Astumian. Thermodynamics and kinetics of a brownian motor. *Science*, 276(5314):917–922, 1997.

[2] JS Bader, RW Hammond, SA Henck, MW Deem, GA McDermott, JM Bustillo, JW Simpson, GT Mulhern, and JM Rothberg. Dna transport by a micromachined brownian ratchet device. *Proceedings of the National Academy of Sciences of the United States of America*, 96(23):13165—13169, November 1999.

```python
#!/usr/bin/env python
# coding: utf-8

# In[2]:


#Importing libraries
import numpy as np
from matplotlib import pyplot as plt
from IPython.display import clear_output
import scipy
from scipy.integrate import quad
from scipy.integrate import solve_ivp
from numba import jit
import copy
from scipy.stats import norm


# In[3]:


#Values
r = 12e-9
eta = 1e-3
gamma = 6*np.pi*eta*r
gamma2 = 3*gamma
L = 20e-6
alpha = 0.2
kbT = 26e-3*1.602e-19
DeltaU = 80*1.602e-19
omega = DeltaU/(gamma*L**2)
omega2 = omega/3
D = kbT/DeltaU

#Flashing function
@jit(nopython=True)
def f(t, tau):
    #global tau
    return int(t%tau >= 0.75*tau) #np.heaviside(t%tau - 0.75*tau, 1)

#Ratchet potential
@jit(nopython=True)
def Ur(x):
    global alpha, L, DeltaU
    return int(x%L < alpha*L)*x%L/(alpha*L)*DeltaU + int(x%L >= alpha*L)*(L-x%L)/(L*(1-alpha))*DeltaU

#Flashin ratchet potential
def U(x, t, tau):
    return Ur(x)*f(t, tau)

#-dU/dx analytically
@jit(nopython=True)
def Fx(x):
    global alpha, L, DeltaU
    return int(x%L < alpha*L)*-1/(alpha*L) + int(x%L >= alpha*L)/(L*(1-alpha))
```

```
#Force, or -dU/dx
@jit(nopython=True)
def F(x, t, tau):
    return DeltaU*Fx(x)*f(t, tau)

#max dU/dx
def maxF():
    global DeltaU
    x = np.linspace(0, 2*L, 1000)
    test = 0
    for i in range(1000):
        if np.abs(Fx(x[i])) > test:
            test = np.abs(Fx(x[i]))
    return DeltaU*test

#Criterion for dt being small enough
def criterion(dt):
    global gamma, kbT, alpha, omega
    return 1/gamma*maxF()*dt + 4*np.sqrt((2*kbT*dt)/gamma) < 0.1*L*alpha

#Euler scheme reduced units one iteration
@jit(nopython=True)
def ForwardEuler(x, t, dt, tau):
    global D, L, omega
    xn = x + F(x*L, t/omega, tau)*L*dt/DeltaU + np.sqrt(2*D*dt)*np.random.normal()
    tn = t + dt
    return xn, tn

#Euler scheme normal units one iteration
@jit(nopython=True)
def ForwardEuler2(x, t, dt, tau):
    global gamma, kbT
    xn = x + 1/gamma*F(x, t, tau)*dt + np.sqrt(2*kbT*dt/gamma)*np.random.normal()
    tn = t + dt
    return xn, tn

#Multi iteration Euler scheme reduced units
@jit(nopython=True)
def ParticleMotion(x0, t0, dt, tau, tEnd, Res):
    Length = int((tEnd - t0)/dt/Res)+1
#     if not criterion(dt):
#         print("dt not sufficiently small, choose a lower dt.")
#         return 0, 0
    c = 1
    t = t0
    x = x0
    T = np.zeros(Length)
    X = np.zeros(Length)
    while t <= tEnd:
        x, t = ForwardEuler(x, t, dt, tau)
        c += 1
        if c%Res == 0:
            X[c//Res], T[c//Res] = x, t
    return X, T

#Multi iteration Euler scheme normal units
```

```python
@jit(nopython=True)
def ParticleMotion2(x0, t0, dt, tau, tEnd, Res):
    Length = int((tEnd - t0)/dt/Res)+1
#     if not criterion(dt):
#         print("dt not sufficiently small, choose a lower dt.")
#         return 0, 0
    c = 1
    t = t0
    x = x0
    T = np.zeros(Length)
    X = np.zeros(Length)
    while t <= tEnd:
        x, t = ForwardEuler2(x, t, dt, tau)
        c += 1
        if c%Res == 0:
            X[c//Res], T[c//Res] = x, t
    return X, T

#Maps positions as energy, requires potential on at all times. Needs simulated trajectory, x, as input. V i
@jit(nopython=True)
def Vlist(V, x, dV, R):
    for i in range(len(x)):
        V[int(np.rint(Ur(x[i])//(R*dV)))] += 1
    return V

#Finds average velocity from tau list, reduced units
def VofTau(x0, t0, dt, tau, tEnd, Res):
    global L, omega
    v = np.zeros(len(tau))
    for i in range(len(tau)):
        print(i)
        X, T = ParticleMotion(x0, t0*omega, dt*omega, tau[i], tEnd*omega, Res)
        v[i] = (X[-1]-X[0])/(T[-1]-T[0])
    return v*L*omega

#Multi iteration Euler scheme reduced units for N particles of each size, reduced units
def MultiParticleMotion(x0, t0, dt, tau, tEnd, Res, N):
    global omega, omega2
    L1 = int((tEnd*omega - t0*omega)/(dt*omega*Res))+1
    L2 = int((tEnd*omega2 - t0*omega2)/(dt*omega2*Res))+1
    X1 = np.zeros([N, L1])
    T1 = np.zeros([N, L1])
    X2 = np.zeros([N, L2])
    T2 = np.zeros([N, L2])
    for i in range(N):
        if i%100000 == 0:
            print(i)
        X1[i], T1[i] = ParticleMotion(x0, t0*omega, dt*omega, tau, tEnd*omega, Res)
        X2[i], T2[i] = ParticleMotion(x0, t0*omega2, dt*omega2, tau, tEnd*omega2, Res)
    return X1, T1, X2, T2


#Returns distribution of Browninan motion after time tEnd, requires potential off at all times
def DensityOfBrownian(N, x0, t0, dt, tau, tEnd, Res, Res2):
    global kbT, gamma, gamma2
    X1, T1, X2, T2 = MultiParticleMotion(x0, t0, dt, tau, tEnd, Res, N)
    Xmin = np.min([np.amin(X1), np.amin(X2)])
    Xmax = np.max([np.amax(X1), np.amax(X2)])
    Lim = 1.5*np.max(np.abs([Xmin, Xmax]))
    xlist = np.linspace(-1*Lim*L, Lim*L, Res2)
    dX = xlist[1] - xlist[0]
    X1list = np.zeros(Res2)
    X2list = np.zeros(Res2)
    for i in range(N):
        X1list[int(rint((X1[i][-1]+Lim)*L/dX))] += 1
        X2list[int(rint((X2[i][-1]+Lim)*L/dX))] += 1
    Diff1 = kbT/gamma
    Diff2 = kbT/gamma2
    Analyt1 = N/np.sqrt(4*np.pi*Diff1*tEnd)*np.exp(-xlist**2/(4*Diff1*tEnd))
    Analyt2 = N/np.sqrt(4*np.pi*Diff2*tEnd)*np.exp(-xlist**2/(4*Diff2*tEnd))
    return xlist, Analyt1, Analyt2, X1list/dX, X2list/dX

#Creates density distributions from positions after time tEnd
def DensityOfParticles(X1f, T1f, X2f, T2f, tEnd, tau, Nx, N, lim):
    global L
    xlist = np.linspace(-2*L, tEnd/tau*L*lim, Nx)
    print(len(xlist))
    dX = xlist[1] - xlist[0]
    X1list = np.zeros(Nx)
    X2list = np.zeros(Nx)
    for i in range(N):
        X1list[int(rint((X1f[i][-1]+2)*L/dX))] += 1
        X2list[int(rint((X2f[i][-1]+2)*L/dX))] += 1
    return xlist, X1list/dX, X2list/dX

#Determines theoretical average velocity based on tau and shift. Shift is determined by dt
def theoreticalV(alpha, kbT, gamma, beta, tau, L, shift):
    diff = kbT/gamma
    return (norm.sf(alpha, shift, np.sqrt(2*diff*beta*tau)/L)-norm.cdf(alpha-1, shift, np.sqrt(2*diff*beta*

#Simulates trajectory with only one iteration per cycle.
#chi is ratio of particle radiuses
def test(beta, tau, N, chi):
    global L, gamma, kbT, alpha
    x = np.zeros(N+1)
    t = np.zeros(N+1)
    for i in range(N):
        t[i+1] = (i+1)*tau
        x1 = np.sqrt(2*kbT*beta*tau/(gamma*chi))*np.random.normal()/L
        x[i+1] = x[i] + ((x1-alpha)//1 +1)*L
    return x, t


# In[4]:


def integrand(x, x0, tau, D, beta):
    return ((x-x0)**2/(4*D*tau*beta) - 3/2)*np.exp(-(x-x0)**2/(4*D*beta*tau))

N = 10000
tau = np.linspace(0.39, 0.42, N)
I = np.zeros(N)
for i in range(N):
    I[i] = quad(integrand, alphax, np.inf, args=(0, tau[i]*omega, Datt, beta))[0]
```