Project 3 - Action Potentials, Parelius

April 9, 2019

0.1 Project 3 - Action Potential

by Thomas Parelius
TMA4320, Spring 2019
Code below enables numbering of LaTeX equations.

```
In [5]: %%javascript
        MathJax.Hub.Config({
            TeX: { equationNumbers: { autoNumber: "AMS" } }
        });
```

```
<IPython.core.display.Javascript object>
```

0.2 Exercise 2.1

Using the diffusion equation with constant diffusion coefficient D:

$$\frac{\partial \phi(x,t)}{\partial t} = D\frac{\partial^2 \phi(x,t)}{\partial x^2}. \tag{1}$$

to check that

$$\phi(x,t) = \frac{1}{\sqrt{4\pi D t}} e^{-\frac{(x-\mu)^2}{4Dt}} \tag{2}$$

is a solution to the diffusion equation we derivate it with regards to t and x and insert into the equation

$$\begin{aligned}\frac{\partial \phi}{\partial t} &= \frac{e^{-\frac{(x-\mu)^2}{4Dt}} \frac{(x-\mu)^2}{4Dt^2}\sqrt{4\pi D t} - e^{-\frac{(x-\mu)^2}{4Dt}} \frac{4\pi D}{2\sqrt{4\pi D t}}}{4\pi D t} \\ &= \frac{e^{-\frac{(x-\mu)^2}{4Dt}}}{4t\sqrt{4\pi D t}}\left(\frac{(x-\mu)^2}{Dt} - 2\right)\end{aligned} \tag{3}$$

$$\frac{\partial \phi}{\partial x} = \frac{-2(x-\mu)e^{-\frac{(x-\mu)^2}{4Dt}}}{4Dt\sqrt{4\pi D t}} \tag{4}$$

$$
\begin{aligned}
\frac{\partial^2\phi}{\partial x^2} &= \frac{-2e^{-\frac{(x-\mu)^2}{4Dt}} - 2xe^{-\frac{(x-\mu)^2}{4Dt}}\frac{-2(x-\mu)}{4Dt} + 2\mu e^{-\frac{(x-\mu)^2}{4Dt}}\frac{-2(x-\mu)}{4Dt}}{4Dt\sqrt{4\pi Dt}} \\
&= \frac{e^{-\frac{(x-\mu)^2}{4Dt}}}{4Dt\sqrt{4\pi Dt}}\left(-2 + \frac{4x(x-\mu)}{4Dt} - \frac{4\mu(x-\mu)}{4Dt}\right) \\
&= \frac{e^{-\frac{(x-\mu)^2}{4Dt}}}{4Dt\sqrt{4\pi Dt}}\left(-2 + \frac{x^2-2x\mu+\mu^2)}{Dt}\right) \\
&= \frac{e^{-\frac{(x-\mu)^2}{4Dt}}}{4Dt\sqrt{4\pi Dt}}\left(-2 + \frac{(x-\mu)^2}{Dt}\right)
\end{aligned} \tag{5}
$$

0.3 Exercise 2.2.1

Finding the time derivative of $\phi(x,t)$ using the diffusion equation, with inital particle distribution

$$\phi(x,0) = \delta(x-x_o) = \frac{1}{2\pi}\int_{-\infty}^{\infty} dk\, e^{-ik(x-x_0)} \tag{6}$$

where $\delta(x)$ is the Dirac delta function.
Using Taylor expansion to the first degree

$$\phi(x,t) = \phi(x,0) + \frac{t^1}{1}\left.\frac{\partial\phi(x,t')}{\partial t'}\right|_{t'=0} \tag{7}$$

Substituting the first term with the inital condition and the last term with the diffusion equation:

$$
\begin{aligned}
\phi(x,t) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} dk\, e^{-ik(x-x_0)} + t\, D\frac{\partial^2\phi(x,0)}{\partial x^2} \\
&= \frac{1}{2\pi}\int_{-\infty}^{\infty} dk\, e^{-ik(x-x_0)} + \frac{tD}{2\pi}\int_{-\infty}^{\infty} dk\, e^{-ik(x-x_0)}(-ik)^2 \\
&= \frac{1}{2\pi}\int_{-\infty}^{\infty} dk[\, e^{-ik(x-x_0)}(1-Dtk^2)]
\end{aligned} \tag{8}
$$

Where $(1-Dtk^2)$ is the Taylor expansion to the first degree of e^{-Dtk^2}
Thus we get

$$\phi(x,t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} dk\, e^{-Dtk^2}e^{-ik(x-x_0)} \tag{9}$$

This is an integral of type

$$\int_{-\infty}^{\infty} e^{-ax^2}e^{-bx}dx = b\sqrt{\frac{\pi}{a}}e^{\frac{b^2}{4a}} \tag{10}$$

which gives us

$$\phi(x,t) = \frac{1}{\sqrt{4\pi Dt}}e^{-\frac{(x-x_0)^2}{4Dt}} \tag{11}$$

0.4 Exercise 2.2.2

The solution I found is a solution where highly concentrated particles were dropped in one place. The solution follows the form of a gaussian (normal distribution):

$$\frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{12}$$

where μ is the mean and σ is the standard deviation. Comparing this to our equation we see that $\sigma = \sqrt{2Dt}$, so the diffusion coefficient is a measure of how fast the standard deviation is increasing with time. In other words how fast the distribution is widening and the particles are spreading.

0.5 Exercise 2.2.3

Finding the time evolution $\phi(x,t)$ given a new initial condition $\phi(x,0) = g(x)$, where $g(x)$ is an arbitrary function.

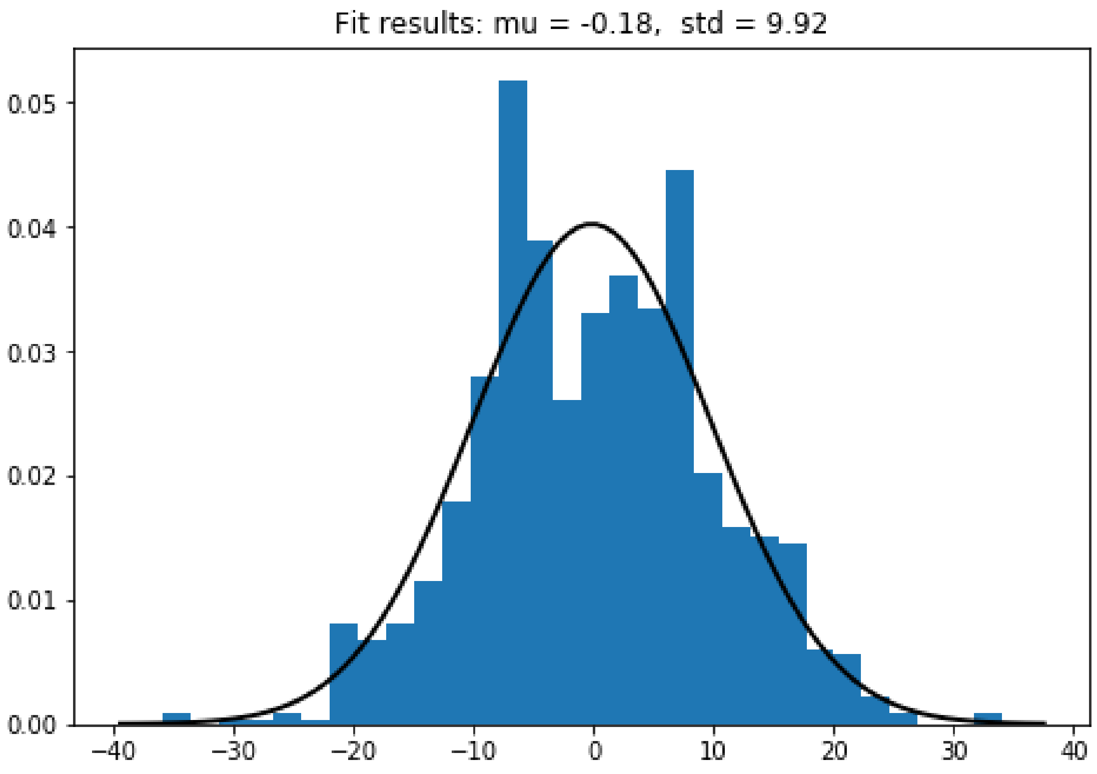
Using the hint $g(x) = \int_{-\infty}^{\infty} dy\, g(y)\delta(x-y)$ we get the solution

$$\phi(x,t) = \int_{-\infty}^{\infty} dy\, g(y) \frac{1}{\sqrt{4\pi Dt}} e^{-\frac{(x-y)^2}{4Dt}} \tag{13}$$

0.6 3 Random walk in 1D

```
In [6]: import numpy as np
        import random
        from scipy.stats import norm
        import matplotlib.pyplot as plt
        %matplotlib inline

        n_part = 1000    # number of particles
        n_steps = 100    # number of steps
        h = 1            # space steps size
        pos = np.zeros([n_part]) # starting all particles at zero

        for i in range(n_part):
            for j in range(n_steps):
                u = random.random()
                if u<0.5:
                    step=-h
                else:
                    step=h
                pos[i] = pos[i]+step

        # fitting the positions to a normal distribution
        mu,std=norm.fit(pos)

        # plotting the histogram
        f=plt.figure()
        ax=f.add_axes([0,0,1,1])
        ax.hist(pos,bins=30,density=True)

        # plotting the probability density function
        xmin, xmax = ax.get_xlim()
        x = np.linspace(xmin, xmax, 100)
        p = norm.pdf(x, mu, std)
        ax.plot(x, p, 'k', linewidth=2)
        title = "Fit results: mu = %.2f,  std = %.2f" % (mu, std)
        ax.set_title(title)
```

```
Out[6]: Text(0.5, 1.0, 'Fit results: mu = -0.18,  std = 9.92')
```



In this random walk we start with all the particles at position zero, which corresponds with the Dirac delta from the previous excercises. And as expected we see that we get a gaussian after several steps.

Using the standard deviation we get from the fitting we can calculate the diffusion coefficient $D = \frac{\sigma^2}{2t} = \frac{10^2}{2\cdot 100} = \frac{1}{2}$, which coincides with the probablity of a particle moving to the left or right in this example. By running the program several times I observed that σ was approximately 10, and that μ was approximately 0.

0.7 5 Random walk in a potential

Potential for exercise 5.1:

$$V(x) = kx \tag{14}$$

Potential for exercise 5.2:

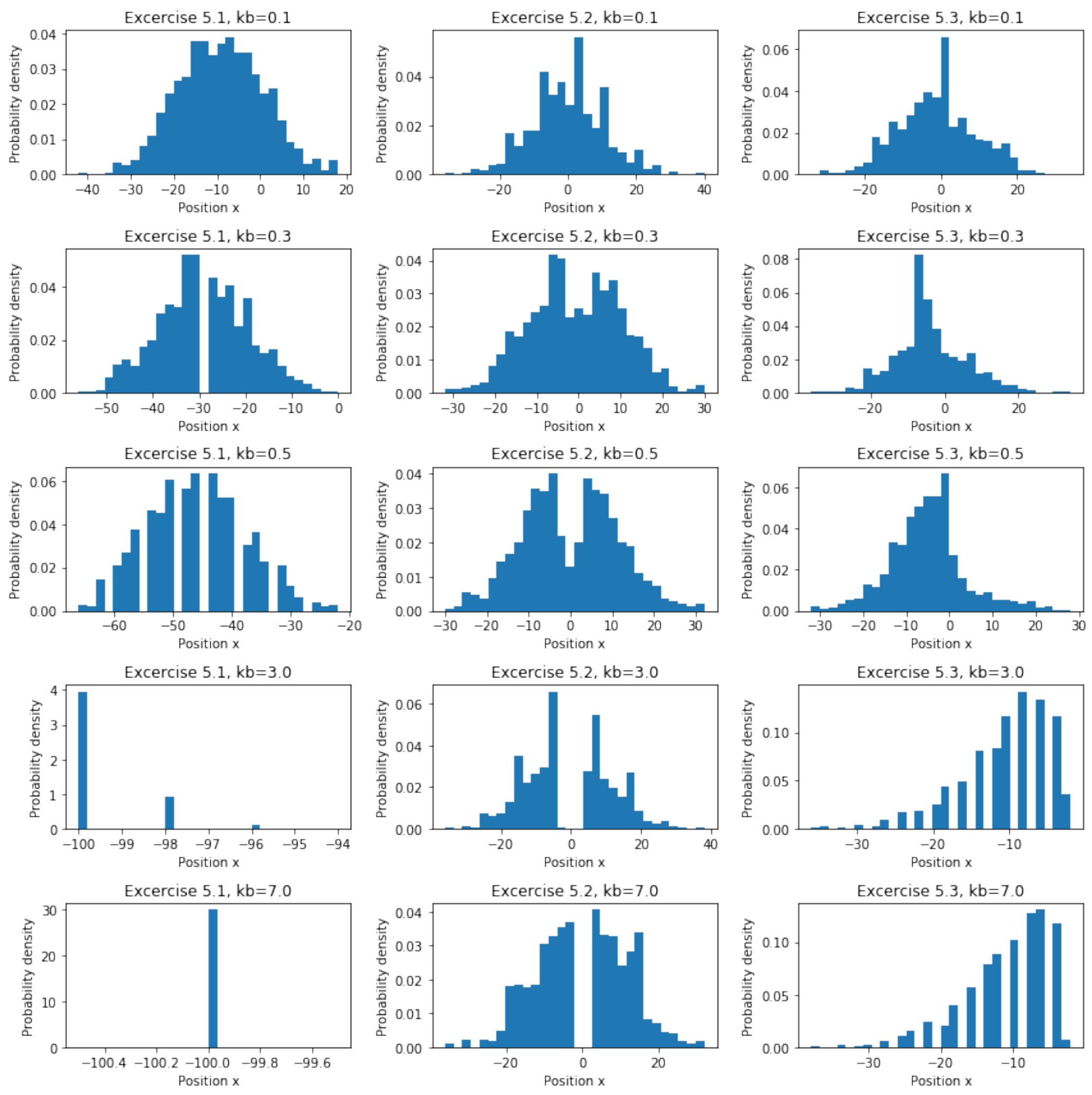
$$\begin{aligned} V(x) &= k, \quad \text{for } -3h < x < 3h \\ V(x) &= 0, \quad \text{otherwise} \end{aligned} \tag{15}$$

Potential for exercise 5.3:

$$\begin{aligned} V(x) &= -k, \quad \text{for } x < -3h \\ V(x) &= k(-1 + 2\frac{x+3h}{6h}), \quad \text{for } -3h < x < 3h \\ V(x) &= k, \quad \text{for } x > 3h \end{aligned} \tag{16}$$

```
In [7]: n_part = 1000    # number of particles
        n_steps = 100    # number of steps
        h = 1            # space steps size
        pos0 = np.zeros([n_part])

        # I define the potentials with beta*k being the scaling factor
        # so the functions return beta*V, which is dimentionless.
        def potential1(x,h,bk):
            bV=bk*x
            return bV

        def potential2(x,h,bk):
            if x>-3*h and x<3*h:
                bV=bk
            else:
                bV = 0
            return bV

        def potential3(x,h,bk):
            if x<=-3*h:
                bV=-bk
            elif x>-3*h and x<=3*h:
                bV = bk*(-1+2*(x+3*h)/6/h)
            else:
                bV = bk
            return bV

        # this function calculates the probability of a particle stepping to the right
        def pplus(x,step,potential,bk):
            pm_over_pp=np.exp(-potential(x-step,step,bk)+potential(x+step,step,bk))
            pp=1/(1+pm_over_pp)
            return pp
```

```python
def randWalk(pos_start, n_steps,h, potentialf,bk):
    n_part=len(pos_start)
    pos=pos_start.copy()
    for i in range(n_part):
        for j in range(n_steps):
            u = random.random()
            pp=pplus(pos[i],h,potentialf,bk)
            pm=1-pp
            if u<pm:
                step=-h
            else:
                step=+h
            pos[i] = pos[i]+step
    return pos

pos=np.empty([n_part,5,3])
kb=[0.1, 0.3, 0.5, 3, 7]
for i in range(len(kb)):
    pos[:,i,0]=randWalk(pos0,n_steps,h,potential1,kb[i])
    pos[:,i,1]=randWalk(pos0,n_steps,h,potential2,kb[i])
    pos[:,i,2]=randWalk(pos0,n_steps,h,potential3,kb[i])

(fig, ax)=plt.subplots(nrows = 5,ncols=3,figsize=[12,12])
for i in range(len(kb)):
    for ex in range(3):
        ax[i,ex].hist(pos[:,i,ex],bins=30,density=True)
        ax[i,ex].set_xlabel("Position x")
        ax[i,ex].set_ylabel("Probability density")
        ax[i,ex].set_title("Excercise 5.%i, kb=%.1f" % (ex+1,kb[i]))
plt.tight_layout()
```



In the first column there is a linear potential of increasing slope kb. This means that the particles are more likely to step to the left than to the right. As the slope gets steeper the particles are pushed further away from their starting position, until at kb=7.0 where the slope is so steep all the particles have taken 100 steps to the left.

In the second column there is a constant potential in the membrane, and zero elsewhere. The height of the potential increases with kb and the particles are less likely to be in the area of high potential.

In the third column there is a potential linear with x in the membrane, while constant negative on the left and constant positive on the right. As kb increases so does the height difference between the two constant potentials. This increase further increases the diffusion to the left, until no particles cross the barrier to the right.

The ratio between the number of particles left of the membrane vs right of the membrane is decided by the voltage difference between them and settles at a level where the number of particles crossing from left to right and that from right to left during a time step is the same. The likelihood of any given particle crossing to the right is small, but the number of particles on the left side of the boundary is high and conversely the probability of a particle crossing from right to left is high, but the number of particles there is low. The ratio between particle amounts will settle at a value where the same number of particles cross the membrane from both sides during each time step. The particles are also diffusing further into the space on both sides.

0.8 7 Interplay between diffusion and voltage

```
In [8]: kb=8.617e-5 #eV/K
        T=273+37 # K
        beta=1/(kb*T) # eV-1
        print("beta =",beta,"1/eV")
```

```
beta = 37.43537718014278 1/eV
```

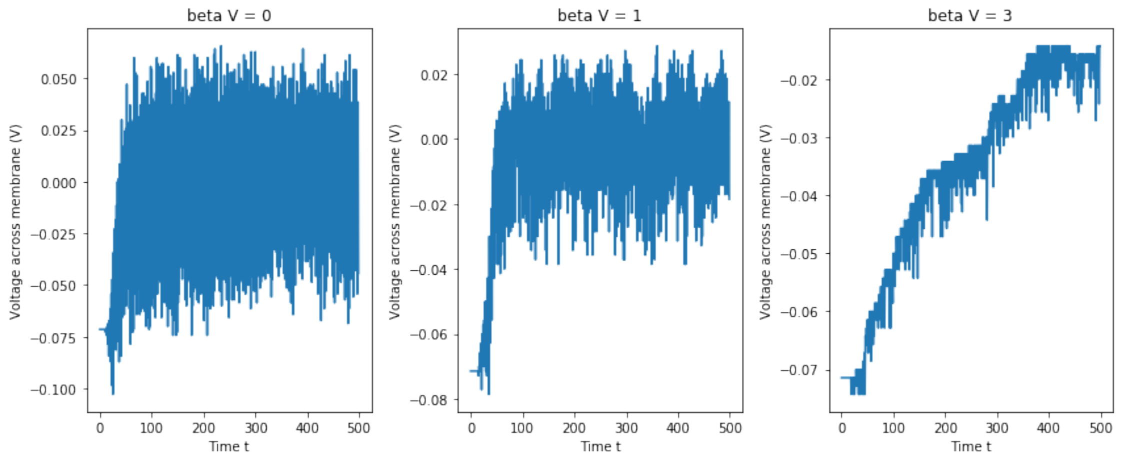
```
In [9]: # membrane potentials for sodium and potasium ions given in eV
        def VNa(x,h,bVNa0):
            beta=37.435
            V=x.copy()
            for i in range(len(x)):
                if x[i]>=-h and x[i]<=h:
                    V[i]=bVNa0/beta
                else:
                    V[i] = 0
            return V

        def VK(x,h,bVK0):
            beta=37.435
            V=x.copy()
            for i in range(len(x)):
                if x[i]>=-h and x[i]<=h:
                    V[i]=bVK0/beta
                else:
                    V[i] = 0
            return V

        # Vel electric potential difference between inside and outside
        def Vel(x,h,posNa,posK,Cc):
            # number of particles inside and outside
            count_in=np.sum(posNa<-h)+np.sum(posK<-h)
            count_out=1450+50 # constant outside
            # voltage difference between inside and outside
            Vdiff=(count_in-count_out)*0.1/Cc
            # vector V holds the potential values for each x,
            # such that potential outside is always set to be 0
            V=x.copy()
            k=-Vdiff/2/h # slope of the potential in the membrane
```

```
    c=-k*h # intercept of the potential in the membrane
    V[(x>=-h) & (x<=h)]=k*x[(x>=-h) & (x<=h)]+c
    V[x<-h]=Vdiff
    V[x>h]=0

    return V

def updateIonPos(x,pos,h,pot_arr):
    beta=37.435
    n_part=len(pos)
    for i in range(n_part):
        if (pos[i]<=x[0]+h): # if particle is at the left border
            step=+h
        elif (pos[i]>=x[len(x)-1]-h): # if particle is at the right border
            step=-h
        else:
            u = random.random()
            pm_over_pp=np.exp(-beta*(pot_arr[x==pos[i]-h]-pot_arr[x==pos[i]+h]))
            pp=1/(1+pm_over_pp)
            pm=1-pp
            if u<pm:
                step=-h
            else:
                step=+h
        pos[i] = pos[i]+step
    return pos

def runRandWalk(x,h,t,posNa0,posK0,cc,bVNa0,bVK0):
    posNa=posNa0.copy()
    posK=posK0.copy()
    n_t=len(t)
    Vout=np.empty(len(t))
    for i in range(n_t):
        V=Vel(x,h,posNa,posK,Cc) # calculate electric potential
        Vout[i]=V[1] # voltage across the cell wall
        # adding the membrane potentials for the ions
        potNa=V+VNa(x,h,bVNa0)
        potK=V+VK(x,h,bVK0)
        # updating ion positions
        posNa=updateIonPos(x,posNa,h,potNa)
        posK=updateIonPos(x,posK,h,potK)
    return Vout,posNa,posK

L=50 # domain length
h=1 # size step
dt=1 # time step
n_t=500 # number of time steps 500
Cc=70 # e mM/V concentration capacitance of the membrane
```

```
# Initial particle positions
# each particle is 0.1 mM inside the cell
n_Na_in=50
n_Na_out=1450
n_K_in=1400
n_K_out=50
posNa0=np.concatenate((-int(L/4)*np.ones(n_Na_in),int(L/4)*np.ones(n_Na_out)))
posK0=np.concatenate((-int(L/4)*np.ones(n_K_in),int(L/4)*np.ones(n_K_out)))

x=np.arange(-L/2,L/2+h,h)
t=np.arange(n_t)
```

0.9 Exercise 7.1

```
In [19]: #memb_pot=[5]
         memb_pot=[0,1,3] # bV membrane potential
         V=np.empty([n_t,len(memb_pot)])
         for i in range (len(memb_pot)):
             V[:,i],posNa,posK=runRandWalk(x,h,t,posNa0,posK0,Cc,memb_pot[i],memb_pot[i])

         (fig, ax)=plt.subplots(nrows = 1,ncols=len(memb_pot),figsize=[12,5])
         for i in range(len(memb_pot)):
             ax[i].plot(t,V[:,i])
             ax[i].set_xlabel("Time t")
             ax[i].set_ylabel("Voltage across membrane (V)")
             ax[i].set_title("beta V = %i" % (memb_pot[i]))
         fig.tight_layout()
```



For all three of the membrane potential, the voltage across the membrane moves towards zero. Increasing the hight of the barrier makes this process longer and also decreases the size of the voltage fluctuations.

With a membrane potential equal to zero the ions have a very low threshold to cross. This is also apparent from the first plot where we see the voltage is equalized quickly followed by wide rapid fluctuations. With such a low threshold a great number of ions will cross back and forth easily until the concentrations are eventually equalized.

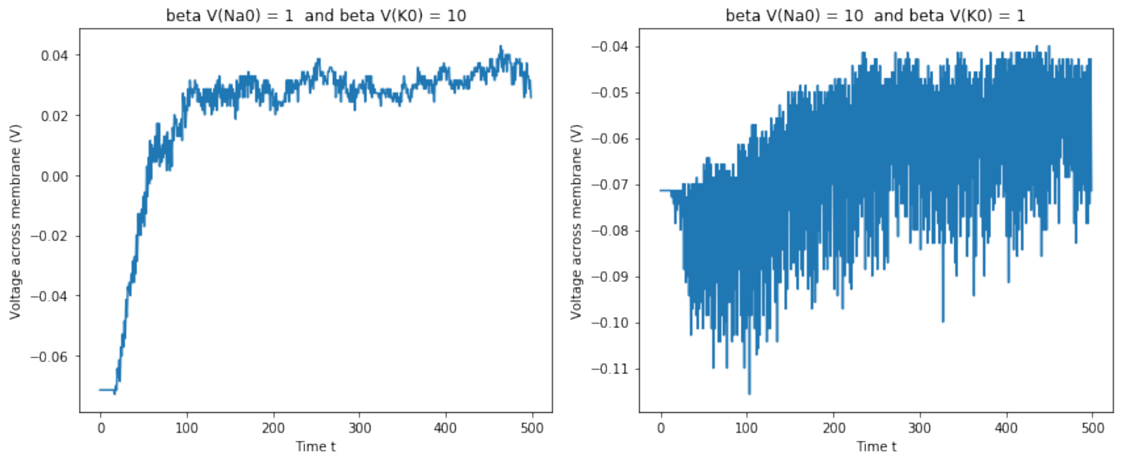
In the second plot the membrane potential has been increased by 1, and the effects are easily spotted in the plot. While the voltage is still quickly equalized, the fluctuations that follow are not as wide as in the previous plot. This is because this time the ions have to cross a small barrier of 1 beta V to cross the membrane.

In the last plot the barrier is so high the ion transfer is noticably slower compared to previous plots. Here we do not get an immediate voltage equalization, rather a slow fluctuating climb towards 0 volt. The slow climb is a result of the high barrier, it causes fewer ions to be able to cross the membrane per step than before. Seeing as diffusion demands a high amount of ’steps’ to equalize a concentration, the high barrier might leave us with less equal concentrations of sodium and potassium at t=500 compared to before.

In all three cases it is important to remember that the difference in number of particles outside and inside the cell is only 50 ions. Through diffusion this will not take long to equalize. But this does not mean that the sodium and potassium concentrations themselves are equalized, as that will take far longer, and many more random steps.

0.10 Exercise 7.2

```
In [17]: V1,posNa1,posK1=runRandWalk(x,h,t,posNa0,posK0,Cc,1,10)
         V2,posNa2,posK2=runRandWalk(x,h,t,posNa0,posK0,Cc,10,1)
         (fig, ax)=plt.subplots(nrows = 1,ncols=2,figsize=[12,5])
         ax[0].plot(t,V1)
         ax[1].plot(t,V2)
         ax[0].set_title("beta V(Na0) = 1  and beta V(K0) = 10" )
         ax[1].set_title("beta V(Na0) = 10  and beta V(K0) = 1" )
         for i in range(2):
             ax[i].set_xlabel("Time t")
             ax[i].set_ylabel("Voltage across membrane (V)")
         fig.tight_layout()
```



In the first plot the membrane is open for sodium and closed for potassium. This means the sodium ions will have little resistance crossing the membrane, while the potassium ions can be considered static, as in they will not cross the membrane. This will cause the voltage to settle at a different level than before.

Our initial voltage across the membrane is negative, and ends up settling at a level above zero. Seeing as potassium ions will not cross the membrane we can assume that sodium ions will be the cause of the change in voltage. Because the voltage is increasing, and sodium ions are positively charged, it must mean that sodium ions are crossing the membrane into the cell.

Sodium ions are highly concentrated outside the cell and will therefore have a large number of ions move across the membrane into the cell quickly, moving along the concentration gradient, as well as along the electric potential gradient. This accounts for the rapid increase in voltage as we see in the plot. The voltage at which the potential settles is positive, because of the large concentration gradient driving the sodium ions in. At equilibrium, the number of sodium ions crossing the membrane per step in each direction is the same. The ions going out are going towards lower electric potential, but against a concentration gradient, while the ions crossing from outside in are moving along a concentration gradient and against an electric potential. Since there are many sodium ions outside, a considerable number will cross inwards each step even though the probability for a single ion crossing is low due to the electric potential being larger inside. At equilibrium, this number is equal to the number of ions coming out (high probability of crossing for each ion, but a lower concentration inside).

In the second plot the membrane is closed for sodium and open for potassium. This means that sodium ions will not be able to cross the barrier. Potassium on the other hand with its low barrier and high concentration inside will have little issue crossing the membrane.

The initial voltage is negative and lower than where it ends up settling, which is still negative. Because the sodium in this case will not cross the membrane, it is the potassium that causes the increase in voltage. And because the voltage outside is set to zero, and the potassium ions are positively charged, it means that the change in voltage we see in the second plot is due to a net increase of potassium concentration inside the cell, despite the high initial concentration there.

The high concentration of potassium ions inside the cell (compared to outside) will cause a considerable number of them moving out against the electric potential, but this increases the potential difference which again makes it more likely for the ions from the outside to cross inwards. This movement of a large number of potassium ions alternatively out then in can be the reason for the large fluctuations in the observed membrane potential.

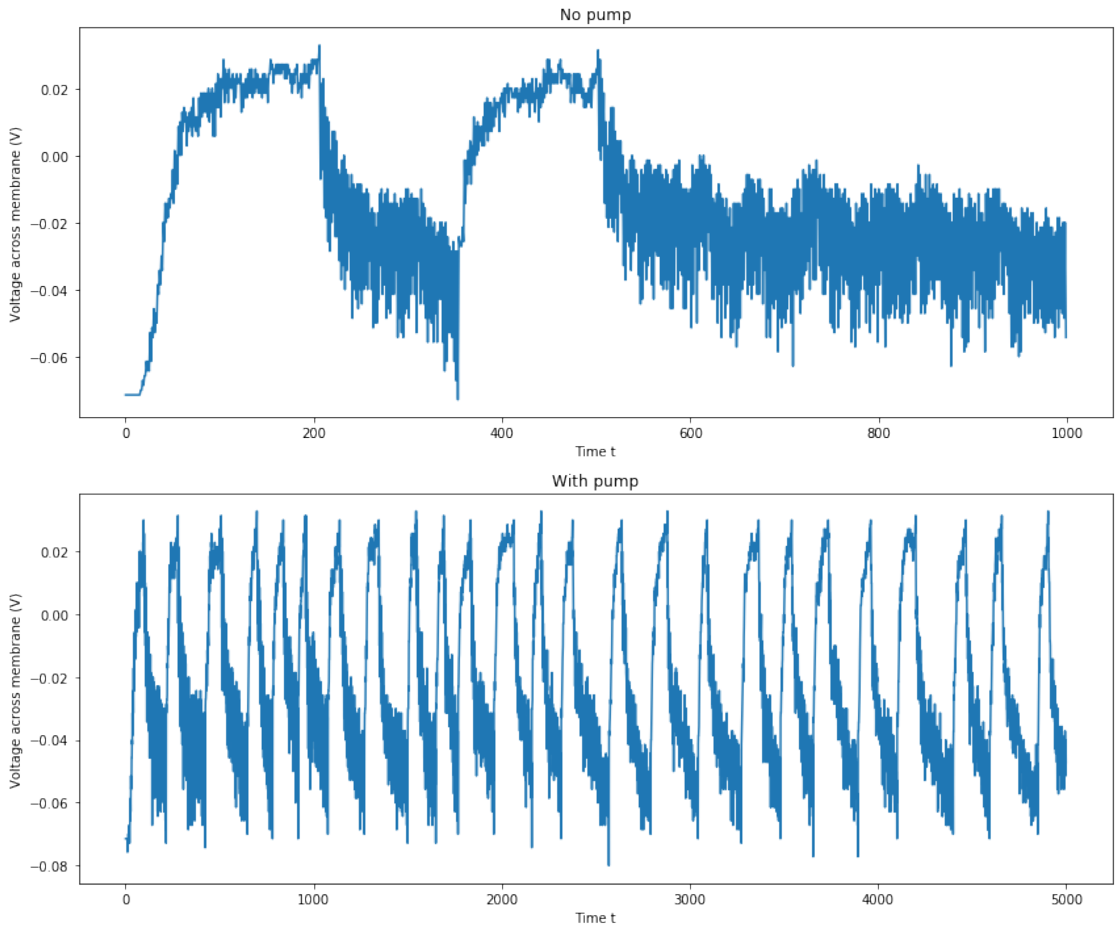
0.11 8 Simulating the action potential

```
In [24]: def ActionPotential(x,h,t,posNa0,posK0,cc,pump_on):
             posNa=posNa0.copy()
             posK=posK0.copy()
             n_t=len(t)
             Vout=np.empty(len(t))
             bVNa0=50
             bVK0=50
             SPP_counter=0
             SPP_not_enough=False
             for i in range(n_t):
                 V=Vel(x,h,posNa,posK,Cc) # calculate electric potential
                 Vout[i]=V[1]
```

```
            if V[1]<=-0.07:
                bVNa0=1
                bVK0=50
            elif V[1]>=0.03:
                bVNa0=50
                bVK0=1
            # update potential for the two ions
            potNa=V+VNa(x,h,bVNa0)
            potK=V+VK(x,h,bVK0)
            # move the ions
            posNa=updateIonPos(x,posNa,h,potNa)
            posK=updateIonPos(x,posK,h,potK)

            if pump_on==True:
                # run the sodium potassium pump
                SPP_counter +=1
                if SPP_counter%10 == 0 or SPP_not_enough==True:
                    if len(posNa[posNa<-h])>=3 and len(posK[posK>h])>=2:
                        posNa[np.where(posNa<-h)[0][0:3]]=h
                        posK[np.where(posK>h)[0][0:2]]=-h
                        SPP_counter = 0
                        SPP_not_enough=False
                    else:
                        SPP_not_enough=True
        return Vout,posNa,posK

    L=50 # domain length
    h=1 # size step
    dt=1 # time step
    n_t1=1000# number of time steps
    n_t2 = 5000# number of time steps
    Cc=70 # e mM/V concentration capacitance of the membrane

    # each particle is 0.1 mM inside the cell
    n_Na_in=50
    n_Na_out=1450
    n_K_in=1400
    n_K_out=50
    posNa0=np.concatenate((-int(L/4)*np.ones(n_Na_in),int(L/4)*np.ones(n_Na_out)))
    posK0=np.concatenate((-int(L/4)*np.ones(n_K_in),int(L/4)*np.ones(n_K_out)))

    x=np.arange(-L/2,L/2+h,h)
    t1=np.arange(n_t1)
    t2 = np.arange(n_t2)
    V1,posNa,posK=ActionPotential(x,h,t1,posNa0,posK0,Cc,False)
    V2,posNa,posK=ActionPotential(x,h,t2,posNa0,posK0,Cc,True)
```

```
In [25]: f,ax=plt.subplots(nrows = 2, ncols = 1, figsize=[12,10])
```

```
ax[0].plot(t1,V1)
ax[1].plot(t2,V2)

ax[0].set_title("No pump" )
ax[1].set_title("With pump" )
for i in range(2):
    ax[i].set_xlabel("Time t")
    ax[i].set_ylabel("Voltage across membrane (V)")
f.tight_layout()
```



The plots shows us that it runs twice before we get no more action potentials. It stops shooting voltage signals because we run out of potassium ions on the inside, because the potassium ion concentration on the inside is decreasing. The sodium ion concentration is also decreasing, though this is just for our simulation as it is not constant concentration outside.

The pump is actively creating concentration gradients, which is a form of stored energy that is partially discharged with each action potential. So with the pump running it will continue shooting voltage signals. During the action potential potassium spills out, and sodium in, but the pump returns them back. So we avoid the problem of running out of ions and concentration gradients.