notebook_DC_deutsch_jakob_AColi

December 16, 2017

1 Mathematische Beschreibung der Ionenkonzentrationen

1.1 Struktur des Modells

The following set of variables and equations is described for one ion species. It would be similar for multiple species, with the so far unknown part being how they influence each others rates across the membrane.

1.1.1 Variablen

Zeit t

Concentrations $c(t)$

- Inside the cell $c_{int}(t)$
- Outside the cell $c_{ext}(t)$

$\implies$ Which make the concentration gradient

$$\Delta c = c_{int} - c_{ext}$$

(Diffusion-) Rates $v(\Delta c, ...)$ Where v is the sum the rates across all membrane components, namely

- Some base level rate for the rest of the membrane v_{mem}
- Respiratory chain v_{res}
- ATP-Synthetase v_{ATP}
- Channel of interest v_{ch}

$$\implies v = \sum_{i \in C} v_i = v_{res} + v_{ATP} + v_{ch} + v_{mem}$$

where we call the set of $(res, ATP, ch, mem) = C$ and with the v_i each directed inward.

1.1.2 Parameters

Volumes V

- Inside the cell V_{int}
- Outside the cell V_{ext}

with V_{ext} being some fixed volume of effective distribution around the membrane.

1.1.3 Equations

The main idea is that the rate of movement across the membrane determines the concentration (gradient), which in turn changes the rates.

Explicitly, the changes in concentration on the in- and outside $\frac{\partial c_{int}}{\partial t}$ and $\frac{\partial c_{ext}}{\partial t}$ correspond to

$$\frac{\partial c_{int}}{\partial t} = \frac{v}{V_{int}} \qquad \text{and} \qquad \frac{\partial c_{ext}}{\partial t} = -\frac{v}{V_{ext}}$$

For implementation, we introduce the spacial "coordinate" x which can take the values 0 or 1 meaning inside or outside, respectively. Therefore, c, V and other variables can be used in code as c[0], V[1], ...

2 Implementierung

```
In [15]: import numpy as np
         from scipy.integrate import odeint
```

2.1 Parameter

```
In [16]: # Innen- und Außenvolumen
         V = [4, 6]

         # Proportionalität der Diffusionsrate v
         a = 0.8
```

2.2 Funktionen

```
In [7]: def v(c):
            gradc = c[1] - c[0]
            return a * gradc

        def cdot(c, t):
            dcdt = [v(c)/V[0], -v(c)/V[1]]
            return dcdt
```

2.3 Integration

2.3.1 Anfangsbedingungen und Auflösung

```
In [8]: # Anfangskonzentrations
        c_0 = [1, 10]

        # Zeitraum und Integrationsschritte
        steps = 300
        time = 10
```

2.3.2 Lösung mit scipy.odeint

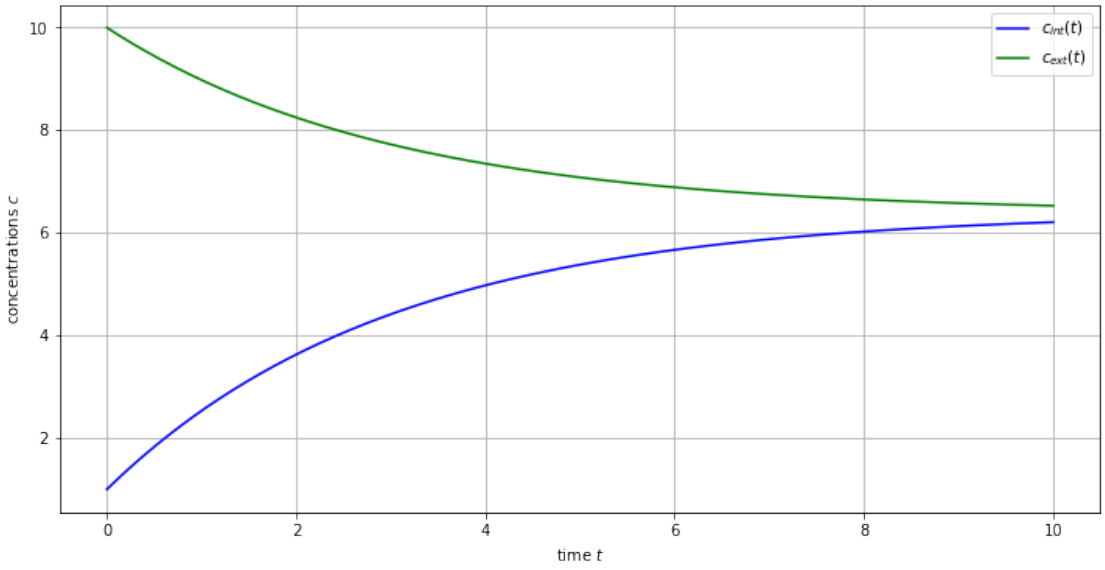
```
In [9]: t = np.linspace(0, time, steps+1)
        solution = odeint(cdot, c_0, t, args=())
```

2.4 Visualisierung

```
In [10]: import matplotlib.pyplot as plt
         %matplotlib inline
```

2.4.1 Zeit-Konzentrations-Diagramm

```
In [11]: tc, tc_ax = plt.subplots(figsize=(12, 6))
         plt.plot(t, solution[:, 0], 'b', label='$c_{int}(t)$')
         plt.plot(t, solution[:, 1], 'g', label='$c_{ext}(t)$')
         plt.legend(loc='best')
         plt.xlabel('time $t$')
         plt.ylabel('concentrations $c$')
         plt.grid()
         plt.show()
         tc.savefig('./figures/time-conc-diagram.png')
```



2.4.2 Räumliche Animation

```
In [12]: # Zum Animieren in diesem notebook
         %matplotlib notebook
         from matplotlib import animation
         from matplotlib.animation import FuncAnimation
```

```
        # Array für Bild definieren
        height = 400
        width = 400

        # half-height
        hh = int(height/2)
        # maximum concentration
        maxconc = max([max([solution[x][0] for x in range(0,len(solution))]),
                       max([solution[x][1] for x in range(0,len(solution))])])

        # Array mit Wert 0 initialisieren
        image = np.zeros((height, width))
        frame = image
```

In [14]:
```
        %matplotlib notebook

        # Figure und Animation
        memanim = plt.figure()

        # Membran einzeichnen
        quer, mem1, mem2 = [0, width], [-5, -5], [+5, +5]
        plt.plot(quer, mem1, quer, mem2, color='green', lw=3, marker = None)

        im=plt.imshow(image, interpolation='none', cmap='Blues', vmin=0, vmax=2,
                      aspect='auto', extent=[0,width,-hh,hh])
        plt.xticks([])


        # Initialisierungsfunktion: Plottet für jeden Frame den Hintergrund
        def init():
            return [image]

        # Animationsfunktion: Wird sequenziell vom Animator aufgerufen
        def animate(i):
            frame[0 : hh-5, 0 : width] = solution[i][1]/maxconc*1.5+0.5
            frame[hh+5 : height, 0 : width] = solution[i][0]/maxconc*1.5+0.5

            im.set_array(frame)
            return [im]

        # Animator aufrufen
        # blit=True damit nur veränderte Pixel neu gesetzt werden
        anima = animation.FuncAnimation(memanim, animate, init_func=init, frames=steps,
                                        interval=int(time/steps*1000), blit=True)

        # animation als *.mp4 speichern
        #anima.save('./figures/animation_mem_conc.mp4', fps=30,
        #           extra_args=['-vcodec', 'libx264'])
```

```
        # animation als *.gif speichern
        # Unter Windows muss dafür evtl. ImageMagick installiert
        # und der convert_path definiert werden, um *.gif s zu speichern
        #plt.rcParams['animation.convert_path'] = '<path-to>/magick.exe'

        writer = animation.ImageMagickFileWriter()
        writer.fps = 30
        #anima.save('./figures/animation_mem_conc.gif', writer=writer)
```

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

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

Animationen funktionieren nur beim Ausführen in jupyter notebook, nicht auf Github und nicht im PDF. Die Animation wird aber auch als *.mp4 und als* .gif gespeichert.