notebook_NP_jakob_AColi

December 12, 2017

0.1 Die Nernst-Planck Gleichung

Die Nernst-Planck-Gleichung beschreibt über Teilchenzahl- bzw. Massenerhaltung die Bewegung von Ionen in Flüssigkeiten unter Einfluss eines elektrischen Feldes.

Allgemein ergibt sich für ein externes elektrisches Feld der Form

$$\mathbf{E} = \nabla\phi\frac{\partial \mathbf{A}}{\partial t} \tag{1.1}$$

aus der Kontinuitätsgleichung

$$\frac{\partial c}{\partial t} = -\nabla \cdot J \tag{1.2}$$

mit

$$J = -\left[D\nabla c - uc + \frac{Dze}{k_\mathrm{B}T}c\left(\nabla\phi + \frac{\partial \mathbf{A}}{\partial t}\right)\right] \tag{1.3}$$

die Nernst-Planck-Gleichung

$$\frac{\partial c}{\partial t} = \nabla \cdot \left[D\nabla c - uc + \frac{Dze}{k_\mathrm{B}T}c\left(\nabla\phi + \frac{\partial \mathbf{A}}{\partial t}\right)\right] \tag{1.4}$$

Dabei sind $\mathbf{r}$ der Ort und t die Zeit, $c = c(\mathbf{r}, t)$ die Ionenkonzentration mit dem entsprechenden Konzentrationsgradienten ∇c, u die Geschwindigkeit des Fluids, D der Diffusionskoeffizient, z die Wertigkeit der Ionenspezies, e die Elementarladung, k_B die Boltzmannkonstante, T die Temperatur, sowie ϕ das skalare und $\mathbf{A}$ das elektromagnetische Vektorpotential.

1 Implementierung

```
In [68]: import numpy as np
         from scipy.integrate import odeint
```

1.1 Parameter und Naturkonstanten

In [69]:
```
# Diffusionskonstante, Wertigkeit der Ionenspezies, Elementarladung, Boltzmann-Konstant
D = 1
z = 1
e = 1
k_B = 1
T = 1
```

2 Funktionen

In [82]:
```
# Berechnung der Divergenz eines Vektorfeldes
def div(f):
    """
    Computes the divergence of the vector field f, corresponding to dFx/dx + dFy/dy + .
    :param f: List of ndarrays, where every item of the list is one dimension of the ve
    :return: Single ndarray of the same shape as each of the items in f, which correspo
    """
    num_dims = len(f)
    return np.ufunc.reduce(np.add, [np.gradient(f[i], axis=i) for i in range(1, num_dim


def cdot(c, t):
    dcdt = div(D * np.gradient(c))
    return dcdt
```

2.1 Integration

2.1.1 Anfangsbedingungen und Auflösung

In [83]:
```
# Dimension und Größe des Raums
length = [100]
dim = len(length)

# Initialisierung des Konzentrations-Arrays, Anfangskonzentration
#c_0 = np.empty([length[i] for i in range(dim)])
#c_0 = np.tile(np.arange(length[0]), 1)
c_0 = np.zeros(length)
c_0[42]=10

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

In [84]:
```
c_0
```

Out[84]:
```
array([  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
         0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
         0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
```

```
       0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,  10.,   0.,
       0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
       0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
       0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
       0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
       0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
       0.])
```

2.1.2 Lösung mit scipy.odeint

```
In [86]: t = np.linspace(0, time, steps+1)
         solution = odeint(cdot, c_0, t, args=())


        ---------------------------------------------------------------------------

        AxisError                                 Traceback (most recent call last)

        <ipython-input-86-00d1d384dc6d> in <module>()
          1 t = np.linspace(0, time, steps+1)
        ----> 2 solution = odeint(cdot, c_0, t, args=())


        ~/.local/share/virtualenvs/lsl/lib/python3.5/site-packages/scipy/integrate/odepack.py in
        213     output = _odepack.odeint(func, y0, t, args, Dfun, col_deriv, ml, mu,
        214                              full_output, rtol, atol, tcrit, h0, hmax, hmin,
        --> 215                              ixpr, mxstep, mxhnil, mxordn, mxords)
        216     if output[-1] < 0:
        217         warning_msg = _msgs[output[-1]] + " Run with full_output = 1 to get quantita


        <ipython-input-82-cab065f602c1> in cdot(c, t)
         11
         12 def cdot(c, t):
        ---> 13     dcdt = div(D * np.gradient(c))
         14     return dcdt


        <ipython-input-82-cab065f602c1> in div(f)
          7     """
          8     num_dims = len(f)
        ----> 9     return np.ufunc.reduce(np.add, [np.gradient(f[i], axis=i) for i in range(1, num_
         10
         11


        <ipython-input-82-cab065f602c1> in <listcomp>(.0)
          7     """
```

```
      8     num_dims = len(f)
----> 9     return np.ufunc.reduce(np.add, [np.gradient(f[i], axis=i) for i in range(1, num_
     10
     11


~/.local/share/virtualenvs/lsl/lib/python3.5/site-packages/numpy/lib/function_base.py in
   1681         axes = tuple(range(N))
   1682     else:
-> 1683         axes = _nx.normalize_axis_tuple(axes, N)
   1684
   1685     len_axes = len(axes)


~/.local/share/virtualenvs/lsl/lib/python3.5/site-packages/numpy/core/numeric.py in norm
   1534     except TypeError:
   1535         axis = tuple(axis)
-> 1536     axis = tuple(normalize_axis_index(ax, ndim, argname) for ax in axis)
   1537     if not allow_duplicate and len(set(axis)) != len(axis):
   1538         if argname:


~/.local/share/virtualenvs/lsl/lib/python3.5/site-packages/numpy/core/numeric.py in <gen
   1534     except TypeError:
   1535         axis = tuple(axis)
-> 1536     axis = tuple(normalize_axis_index(ax, ndim, argname) for ax in axis)
   1537     if not allow_duplicate and len(set(axis)) != len(axis):
   1538         if argname:


AxisError: axis 1 is out of bounds for array of dimension 0
```

2.2 Referenzen

[1] https://en.wikipedia.org/wiki/Nernst%E2%80%93Planck_equation (27. November 2017) [2] http://www.columbia.edu/cu/biology/courses/w3004/Nernstequationderiv.pdf (30. November 2017) [3] http://hpfem.org/wp-content/uploads/doc-web/doc-examples/src/hermes2d/examples/nernst-planck.html (30. November 2017) [4] http://www.sci.osaka-cu.ac.jp/~ohnita/2010/TCLin.pdf (12. Dezember 2017)