Exercise Sheet 1: Neurons

In [1]:

```
import numpy as np
from scipy import constants
from matplotlib import pyplot as plt
```

Task 1: The logistic function (1 point - programming)

Write a function "logistic(h,a=1)" resembling the logistic function and a second function "signtrafunc" the more discrete tranfer function equivalent in form of a function of the sign of the input. Also write a function "dlogistic(h,a=1)" resembling the derivative of the logistic function.

For defining the sign-based transfer function the inline definition of functions using the syntax trafunc=lambda x: f(x) might be handy, please write the logistic function, however, as a classical function definition. The a=1 in the function definition sets the default to a=1 so the function needs only 1 argument by default.

In [2]:

```
def logistic(h,a=1):
    return 1 / (1+ np.exp(-(a*h)))

sig = lambda x: (np.sign(x)+1)/2

def dlogistic(h, a=1):
    #alternative? return logistic(h,a)*(1-logistic(h,a))
    return (a * np.exp(-a * h)) / (1 + np.exp(-a * h))**2
```

Task 2: Building an artificial neuron (2 points - programming)

Define a class "neuron" resembling a simple perceptron. See https://docs.python.org/3/tutorial/classes.html (https://docs.python.org/3/tutorial/classes.html) for help on classes.

It should have the attributes "neuron.w" for the weights, "neuron.b" for the bias, the methods "neuron.trafunc(self,x)" and "neuron.dtrafunc(self,x)" for the transfer function and the method "neuron.out(self,x)" to calculate the output for input *x*.

Build a constructor method "__init__(self,...)", where self refers to the object itself, which assigns the weight vector *x*, the bias *b* , the transfer function "trafunc" and it's derivative "dtrafunc" to the neuron.

In [40]:

```
class neuron_log:

    def __init__(self, W, B):
        self.w = W
        self.b = B

    def trafunc(self, x):
        return logistic(x)

    def dtrafunc(self, x):
        return dlogistic(x)

    def out(self, x):
        return self.trafunc(np.dot(self.w, x) - b)


##just for task 3
class neuron_sign:

    def __init__(self, W, B):
        self.w = W
        self.b = B

    def trafunc(self, x):
        return sig(x)

    def out(self, x):
        return self.trafunc(np.dot(self.w, x) - b)
```

In [41]:

```
##just a test
class neuron:

    def __init__(self, W, B):
        self.w = W
        self.b = B

    def trafunc(self, x,trans):
        return trans(x)

    def dtrafunc(self, x):
        return dlogistic(x)

    def out(self, x, trans = logistic):
        return self.trafunc(np.dot(self.w, x) - b , trans)

#test
x = np.array([[0,0], [1,0], [0,1], [1,1]])
w = np.array([20,20])
b = 10
n = neuron(w,b)
print("OR sign: ",(n.out(x.T, trans = sig)))
```

```
OR sign:  [ 0.  1.  1.  1.]
```

Task 3: Use an artificial neuron (3 points - thinking and programming)

Use instances of the neuron class above to build the equivalents to logical "or" and "and"-functions and test them for 2-dimensional input vectors *x* resembling all possibilities of combinations ([0,0] [1,0], [0,1], [1,1]).

First, use the sign-based transfer function to solve the task and then apply the same weights w and bias b on a neuron with the logistic transfer function.

What's the difference?

Hint: if you haven't succeeded with Task 2, you can solve Task 3 analytically and write down the needed weights and results.

In [42]:

```
#input array
x = np.array([[0,0], [1,0], [0,1], [1,1]])



#OR Gatter: y_or = np.array([0,1,1,1])
w_1 = np.array([20,20])
b_1 = 10

n1 = neuron_sign(w_1,b_1)
print("OR sign: ",(n1.out(x.T)))

n2 = neuron_log(w_1,b_1)
print("OR logistic: ",(n2.out(x.T)))

#AND Gatter: y_and = np.array([0,0,0,1])
w_2 = np.array([10,10])
b_2 = 20

n3 = neuron_sign(w_2,b_2)
print("\nAND sign: ",(n3.out(x.T)))

n4 = neuron_log(w_2,b_2)
print("AND logistic: ",(n4.out(x.T)))
```

```
OR sign:  [ 0.  1.  1.  1.]
OR logistic:  [  4.53978687e-05   9.99954602e-01   9.99954602e-01
1.00000000e+00]

AND sign:  [ 0.   0.5  0.5  1. ]
AND logistic:  [  4.53978687e-05   5.00000000e-01   5.00000000e-01
9.99954602e-01]
```

Difference: sign is not steady whereas the logistic function is.

Task 4: The XOR function (100 extra points - thinking and maybe programming)

Build an XOR function with a neuron of the class you have created.

doesn't make sense, because how can I make the sum of the weight * input to be less than the bias/threshold, while mainting the other conditions, for example the [0,1] or [1,0]. you need at least 2 neurons to build an XOR function

Task 5: The current density (4 points - analytic derivation)

Analytically derive the current density $\vec{J} = -\sigma\nabla\phi$ of a dipole field at origin step-by-step from the potential equation $\phi(\vec{r}) = \frac{1}{4\pi\epsilon_0}\frac{\vec{p}\cdot\vec{r}}{|\vec{r}|^3}$ in Cartesian coordinates.

To this extent you will need to build the gradient $\nabla = \begin{pmatrix} \frac{\delta}{\delta x} \\ \frac{\delta}{\delta y} \\ \frac{\delta}{\delta z} \end{pmatrix}$!

You can take the dipolar moment $\vec{p}$ to be a constant vector $\vec{p} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}$.

The vector $\vec{r} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ is the coordinates in cartesian coordinates and the norm is the euclidean norm $|\vec{r}| = \sqrt{x^2 + y^2 + z^2}$.

σ is the specific conductivity of the material (a constant).

Hint: You can hand in the solution in your format of choice: Write direclty below (Markdown takes latex within $$, jsut double-click on the cell below), Pdf from Latex or Word, scan or photo of a hand-written derivation, etc....

Solution:

$\vec{J} = -\sigma\nabla\phi =$ siehe unten fotos

Task 6: Electrical Field strength & potential of a dipole (5 points - programming)

Write a python function resembling the electrical field potential $\phi(\vec{r}, \vec{r}_0)$ and another one for the field strength $\vec{E}$ of a dipole in infinite charge free space at point $\vec{r}_0$ in Cartesian coordinates. The function should take the position of the observed point $\vec{r}$, the position of the dipole $\vec{r}_0$ and the dipole moment $\vec{p}$ as inputs and return the scalar potential. $\vec{r}_0$ should be set to the origin by default.

$$\phi(\vec{r}) = \frac{1}{4\pi\epsilon_0} \frac{\vec{p}\cdot(\vec{r}-\vec{r}_0)}{|\vec{r}-\vec{r}_0|^3}$$

$$\vec{E} = \frac{1}{4\pi\epsilon_0} \left(3\frac{(\vec{r}-\vec{r}_0)\cdot\vec{p}}{|\vec{r}-\vec{r}_0|^5} \cdot (\vec{r}-\vec{r}_0) - \frac{\vec{p}}{|\vec{r}-\vec{r}_0|^3} \right)$$

Make a plot of the two variables in two dimensions from -10 to 10 for both axes in the x/y-plane for a dipole at origin with dipole moment $\vec{p} = [1, 0]$. Plot the potential as decibel values using the matlpotlib function plt.contour/contourf and the field strength using the function plt.steamplot. Draw both plots into the same figure (plt.hold('on')). The arguments to the plot function call for $\vec{E}$- and ϕ- values need to have the shape corresponding to the XX and YY variables from np.meshgrid (see below).

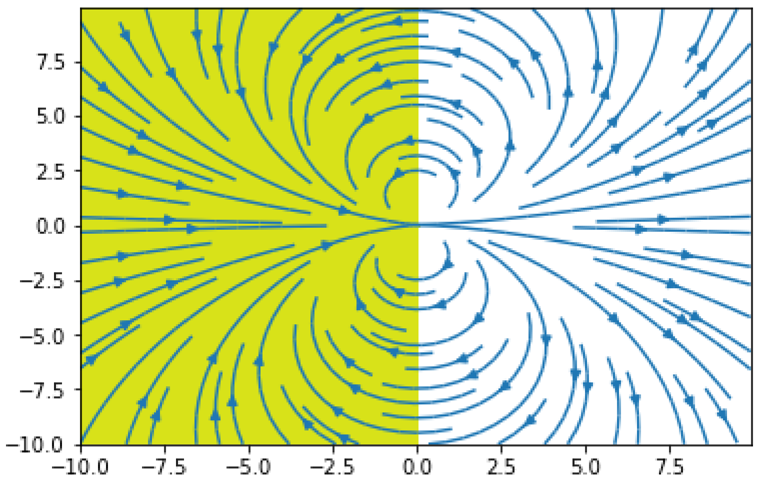
Hint: You can find ϵ_0 in the scipy constants. Numpy's tensordot might simplify the treatment of multiple positions at once.

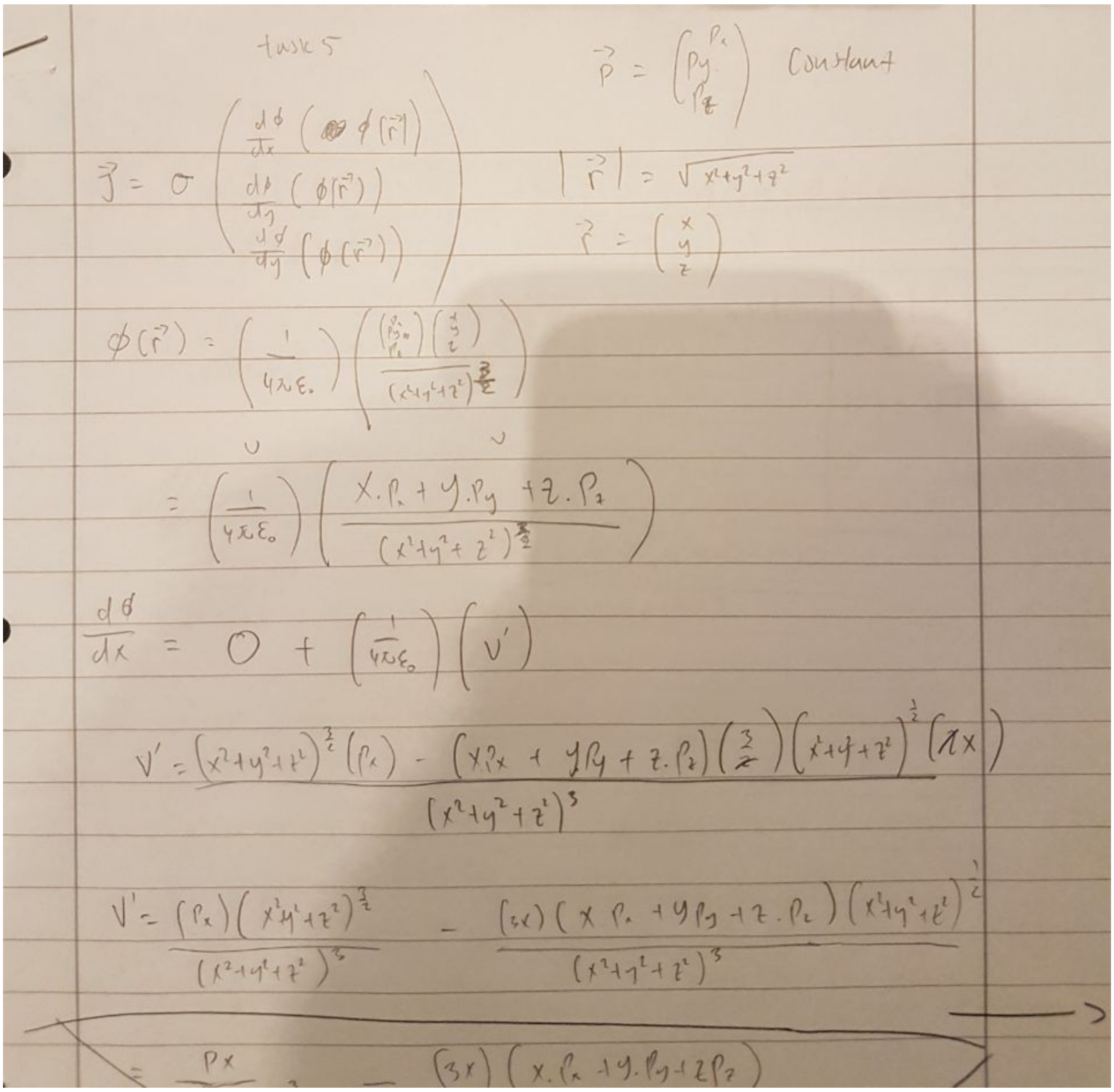
In [7]:

```
x=np.arange(-10,10,0.1)
y=np.arange(-10,10,0.1)

XX,YY=np.meshgrid(x,y)

p = np.array([1,0]) #dipole moment
r = np.array([XX,YY])

def phi_dip(r, p, r_0 = np.array([0,0])):
    k = 1 / 4 * constants.pi * constants.epsilon_0
    return k * np.tensordot(p,r-r_0[:,np.newaxis, np.newaxis], axes = 1) / (np.l
inalg.norm(r-r_0[:,np.newaxis, np.newaxis],axis = 0))**3

def E_dip (r,p,r_0 = np.array([0,0])):
    k = 1 / 4 * constants.pi * constants.epsilon_0
    div = (np.linalg.norm(r-r_0[:,np.newaxis, np.newaxis],axis = 0))**3
    h = np.zeros((2,div.shape[0],div.shape[1]))
    h[0,:,:] = p[0] / div
    h[1,:,:] = p[1] / div
    return k* (3 * np.tensordot(p,r-r_0[:,np.newaxis, np.newaxis], axes = 1) / (
np.linalg.norm(r-r_0[:,np.newaxis, np.newaxis],axis = 0))**5 * (r-r_0[:,np.newax
is, np.newaxis]) - h)

#Calculate the field and potential on the whole meshgrid
phi=phi_dip(r,p)

E=E_dip(r,p)

%matplotlib inline
plt.contourf(XX, YY, phi)
plt.streamplot(x, y, E[0], E[1])
```

Out[7]:

```
<matplotlib.streamplot.StreamplotSet at 0xac0eaaac>
```





task 5

$$\vec{P} = \begin{pmatrix} P_x \\ P_y \\ P_z \end{pmatrix} \quad \text{Constant}$$

$$\vec{J} = \sigma \begin{pmatrix} \frac{d\phi}{dx}\left(\phi(\vec{r})\right) \\ \frac{d\phi}{dy}\left(\phi(\vec{r})\right) \\ \frac{d\phi}{dy}\left(\phi(\vec{r})\right) \end{pmatrix}$$

$$|\vec{r}| = \sqrt{x^2+y^2+z^2}$$

$$\vec{r} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$$\phi(\vec{r}) = \left(\frac{1}{4\pi\varepsilon_0}\right)\left(\frac{\begin{pmatrix} P_x \\ P_y \\ P_z \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix}}{(x^2+y^2+z^2)^{\frac{3}{2}}}\right)$$

$$= \underset{u}{\left(\frac{1}{4\pi\varepsilon_0}\right)} \underset{v}{\left(\frac{x.P_x + y.P_y + z.P_z}{(x^2+y^2+z^2)^{\frac{3}{2}}}\right)}$$

$$\frac{d\phi}{dx} = 0 + \left(\frac{1}{4\pi\varepsilon_0}\right)\left(v'\right)$$

$$v' = \frac{(x^2+y^2+z^2)^{\frac{3}{2}}(P_x) - (x.P_x + y.P_y + z.P_z)\left(\frac{3}{\cancel{2}}\right)(x^2+y^2+z^2)^{\frac{1}{2}}(\cancel{2}x)}{(x^2+y^2+z^2)^3}$$

$$v' = \frac{(P_x)(x^2+y^2+z^2)^{\frac{3}{2}}}{(x^2+y^2+z^2)^3} - \frac{(3x)(x.P_x + y.P_y + z.P_z)(x^2+y^2+z^2)^{\frac{1}{2}}}{(x^2+y^2+z^2)^3}$$

$$= \frac{Px}{\quad} - (3x)(x.P_x + y.P_y + zP_z)$$

$$\frac{d\phi}{dx} = \frac{(p_x)\left(x^2+y^2+z^2\right)^{\frac{3}{2}}}{(x^2+y^2+z^2)^3} - \frac{(3x)\left(x.p_x + y.p_y + z.p_z\right)\left(x^2+y^2+z^2\right)^{\frac{1}{2}}}{(x^2+y^2+z^2)^3}$$

$$= \frac{p_x}{\left(\sqrt{x^2+y^2+z^2}\right)^3} - \frac{3x \begin{pmatrix} x \\ y \\ z \end{pmatrix}\begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}\left(x^2+y^2+z^2\right)^{\frac{1}{2}}}{(x^2+y^2+z^2)^3}$$

$$\boxed{\begin{aligned} \vec{p} &= \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \\ \vec{r} &= \begin{pmatrix} x \\ y \\ z \end{pmatrix} \\ |\vec{r}| &= \sqrt{x^2+y^2+z^2} \end{aligned}}$$

= [illegible]

$$= \frac{p_x}{|\vec{r}-\vec{r}_0|^3} - \frac{3x\,(\vec{r}-\vec{r}_0)(\vec{p})\left(x^2+y^2+z^2\right)^{\frac{1}{2}}}{(x^2+y^2+z^2)^3}$$

$$= \frac{p_x}{|\vec{r}-\vec{r}_0|^3} - \frac{3x\,(\vec{r}-\vec{r}_0)(\vec{p})}{(x^2+y^2+z^2)^{\frac{5}{2}}}$$

$$\frac{d\phi}{dx} = \frac{p_x}{|\vec{r}-\vec{r}_0|^3} - 3 \cdot \frac{(\vec{r}-\vec{r}_0)(\vec{p})}{|\vec{r}-\vec{r}_0|^5} \cdot x$$

$$\frac{d\phi}{d\begin{pmatrix} x \\ y \\ z \end{pmatrix}} = \frac{\begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}}{|\vec{r}-\vec{r}_0|^3} - 3 \cdot \frac{(\vec{r}-\vec{r}_0)(\vec{p})}{|\vec{r}-\vec{r}_0|^5} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$$\vec{E} = \nabla\phi = \frac{\vec{p}}{|\vec{r}-\vec{r}_0|^3} - 3 \cdot \frac{(\vec{r}-\vec{r}_0)(\vec{p})}{|\vec{r}-\vec{r}_0|^5} \left(\vec{r}-\vec{r}_0\right)$$