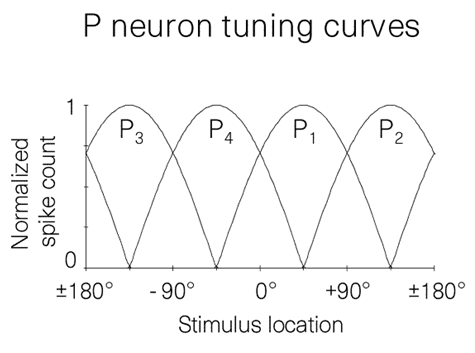
**Graded assignment 1 – 4 credit supplementary**

At this time, you all have implemented leaky integrate and fire neurons and used them to run simple simulations. In this assignment you will use your coding experience and what you have learned about population coding to implement a simple network similar to the leech bend-away-from-touch network that we designed in class.

This assignment is subdivided into several parts and I will try to explain my logic as to how to do this.

1. *Create 4 sensory neurons with sensory receptive fields in response to touch which span 360 degrees so that each stimulus is covered. The figure below is an example but you don’t have to copy this. You don’t need to implement real neurons, just a representation of spikes per unit time as a function of angle. Most probably these will look more triangular than sinusoidal and that is fine. These four neurons represent four “input lines” to your network of motor neurons. Use a 1 second touch stimulus to create the spike trains. Plot spike trains of one neuron in response to at least 6 angles to show how this works.*



- to create the tuning curves create a means to get y-values for each x-value and these depend on the best angle and the angle in question. There are multiple ways to do this, you could use a cosine function for example, or create a triangular function for which the slope dy/dx defines the width of the tuning curve.



* Once you have a way to find the activation values for an given angle you want to translate these into spike trains. The easiest way to think about is that a maximal activation of 1 creates for example a 100Hz train, an activation of 0.5 an 50 Hz train and so on . An easy way to solve this is to create a probability function in which the probability to emit a spike depends on the activation value.



* If you want to test this procedure, you can then retransform these spike trains into rate-numbers to draw a tuning curve . To test that it works you need two loops: one that loops through all the angles and one that loops through your 100 points.
* 

Next you are asked to calculate population vectors from your spike train tuning curves.

1. ***First*** *implement a population vector calculation that allows you to predict the angle of any stimulus from the firing rates of your sensory neurons. Test your population vector calculation on three different angles as well as a situation in which two angles spanned by separate sensory neurons are touched at the same time (similar to what was described in the paper). You can do this what I call “by hand”, just write it all out or use vector algebra.*

We can reuse our function from HW3 to compute a population vector in which max is a vector of maximal angles for each neuron and angles is a vector of spike rates for the particular angle we are looking for.



1. ***First*** *implement a population vector calculation that allows you to predict the angle of any stimulus from the firing rates of your sensory neurons. Test your population vector calculation on three different angles as well as a situation in which two angles spanned by separate sensory neurons are touched at the same time (similar to what was described in the paper). You can do this what I call “by hand”, just write it all out or use vector algebra.*

We can reuse our function from HW3 to compute a population vector in which max is a vector of maximal angles for each neuron and angles is a vector of spike rates for the particular angle we are looking for.

1. **Second** create a network of 12 postsynaptic neurons receiving spiking inputs from these four sensory neurons and which have receptive fields with a better resolution than the sensory neurons. You want to try and keep their receptive fields as narrow as possible. These should be LIF neurons and should all have the same time constant and threshold but receive inputs from these four sensory neurons with specific synaptic weights. Hint: I would figure out first on a piece of paper or with just one neuron how to choose the amplitude of the incoming spikes, the weights and time constant etc to make this work. If you want narrow receptive fields you want the neurons to have a relatively high threshold for. Use the information you gathered in HW3 and in the written portion of the assignment to guide you in parameter choice.

You can first test if this works by using a single spike train and having it activate one LIF neuron and try reproduce its tuning curve:

def LIF (i, oldv, tau, thresh):

sum=(1-exp(-1/tau))\*i+exp(-1/tau)\*oldv calculate new voltage

if sum >= thresh: compare to threshold

return ([0,5])

else:

return ([sum, 0]) return voltage and output

This function takes an input, a previous voltage, time constant and threshold and returns the voltage value and output value

To run this neuron over a given number of time steps we can use

def RUN\_LIF (y, tau, thresh):

n=len(y)

x=zeros(n)

v=zeros(n)

i=1

while i<n:

[v[i],x[i]]=LIF(y[i],v[i-1],tau, thresh)

i=i+1

return([v,x])

in which y is a vector of n input values, tau and thresh the LIF parameters.

Now, to test this, we first will compute the spike trains for one of our sensory neurons for a given angle, drive the LIF neuron with it, get its response spike train, compute the sum of this, and d o this again for the range of angles to obtain a tuning curve.

def TESTLIF (anglestep, tau, thresh, weight):

n=360/anglestep #calculates the number of angles

n1=zeros(n+1) #creates a vector for the tuning curve

angles=zeros(n+1) #creates a vector for the angles

a=0

nn=0

while a<=360:

ny1=TUNE(45, a, 0.01) #compute the activity of the sensory neuron at angle a

n1spikes = zeros(100) #create a vector to put spikes in

i=0

while i<100:

if random()<ny1:

n1spikes[i]=1.0 #create spike train from activation value

i=i+1

outy1=RUN\_LIF(weight\*n1spikes, tau, thresh) #send spike train to LIF and get output

n1[nn]=sum(outy1)

angles[nn]=a #write angle into angle vector

a=a+anglestep increment angle

nn=nn+1

return (angles, n1) return angle vector and tuning curve of LIF

We can call this function :

[a,n] = TESTLIF(15, 2, 1, 1) using a step of 15 degrees, time constant of 2, threshold pf 0.8 and weight of 1 we get:



1. **Third**, as output, plot the responses (spikes per unit of stimulus application) as a function of angle for each of the motor neurons.

now we just want to repeat this exercise but have postsynaptic neurons which receive input from more than one sensory cell with different weights! I am demonstrating here with just 4 neurons covering 1/3 of the range (0-120 degrees)

def POSTSYNAPTIC (anglestep):

n=360/anglestep

n1=zeros(n+1)

n2=zeros(n+1)

n3=zeros(n+1)

n4=zeros(n+1)

angles=zeros(n+1)

a=0

nn=0

while a<=360:

ny1=TUNE(45, a, 0.01)

ny2=TUNE(135, a, 0.01)

ny3=TUNE(225, a, 0.01)

ny4=TUNE(315, a, 0.01)

n1spikes=zeros(100)

n2spikes=zeros(100)

n3spikes=zeros(100)

n4spikes=zeros(100)

i=0

while i<100:

if random()<ny1:

n1spikes[i]=1.0

if random()<ny2:

n2spikes[i]=1.0

if random()<ny3:

n3spikes[i]=1.0

if random()<ny4:

n4spikes[i]=1.0

i=i+1

[v1, x1]=RUN\_LIF(1.0\*n1spikes, 2, 0.9)

[v2,x2]=RUN\_LIF(8.0\*n1spikes+2.0\*n2spikes, 2, 8.0)

[v3,x3]=RUN\_LIF(2\*n1spikes+8\*n2spikes, 2, 8)

[v4,x4]=RUN\_LIF(1.0\*n2spikes, 2, 0.9)

n1[nn]=sum(x1)

n2[nn]=2\*sum(x2)

n3[nn]=2\*sum(x3)

n4[nn]=sum(x4)

angles[nn]=a

a+=anglestep

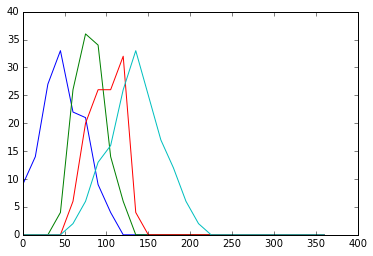
nn+=1

plot(angles,n1)

plot(angles,n2)

plot(angles,n3)

plot(angles,n4)



1. Please submit your code as well as a detailed explanation of what you did !