m78huang_a1

January 26, 2020

1 A1

[1]:
```
# Standard imports
import numpy as np
import math
import matplotlib.pyplot as plt
%matplotlib inline
```

1.0.1 Some solutions (optional)

[2]:
```
# RUN THIS CELL ONLY IF YOU WANT TO USE THE SOLUTIONS THROUGHOUT
from a1_solutions import *
```

[3]:
```
# Run this cell if you want to be able to run functions from the solutions,␣
↪like this...
# lif_a = solutions.LIFNeuron()
import a1_solutions as solutions
```

1.1 Some supplied helper functions

[4]:
```
def PlotSpikeRaster(st, y_range=[0, 1.]):
    '''
    PlotSpikeRaster(spiketimes, y_range=[0, 1.])

    Plots a spike raster plot for a list of arrays of spike times.

    Input:
      spiketimes is a list of arrays of spike times, like that returned
          by the function Stim2Spikes.
      y_range is a 2-tuple that holds the y-values that the raster ticks
          should be drawn between
    '''
    N = len(st)  # number of neurons

    #levels = np.linspace(y_range[0], y_range[1], N+1, endpoint=True)
    levels = np.linspace(y_range[1], y_range[0], N+1, endpoint=True)
    for n in range(N):
```

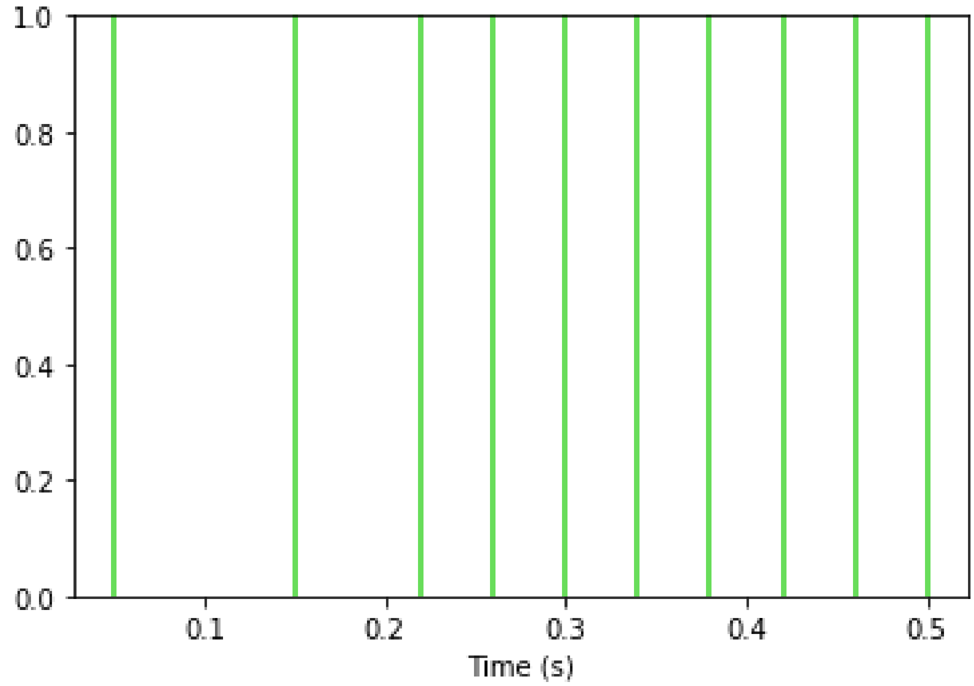
```
        nspikes = len(st[n])
        #y = [ [levels[n]]*nspikes , [levels[n+1]]*nspikes ]
        y = [ [levels[n+1]]*nspikes , [levels[n]]*nspikes ]
        #y = y_range[0] + [levels[n]]*nspikes
        plt.plot(np.vstack((st[n],st[n])), y, color=np.random.rand(3))
    plt.ylim(y_range)
    plt.xlabel('Time (s)')
    return


def GenerateSpikeTrain(rates, T, jitter=0.):
    '''
    spike_times = GenerateSpikeTrain(rates, T)

    Creates a spike train (as an array of time stamps).

    Input:
    rates is an array or list of firing rates (in Hz), one
        firing rate for each interval.
    T is an array or list (the same size as 'rates') that gives
        the time to end each interval
    jitter is a scalar that determines how much the spikes
        are randomly moved

    Output:
    spike_times is an array of times when spikes occurred

    Example: To create a spike train of 10Hz for 0.5s, followed
             by 25Hz that starts at 0.5s and ends at 2s, use

               GenerateSpikeTrain([10, 25], [0.5, 2])
    '''
    s = []
    t = 0.
    for idx in range(0,len(rates)):
        Trange = T[idx] - t
        if rates[idx]!=0:
            delta = 1. / rates[idx]
            N = rates[idx] * Trange
            times = np.arange(t+delta/2., T[idx], delta)
            times += np.random.normal(scale=delta*jitter, size=np.shape(times))
            s.extend(times)
        t = T[idx]
    s.sort()
    return np.array(s)
```

```
class InputNeuron(object):

    def __init__(self, spiketrain):
        '''
        InputNeuron(spiketrain)

        Constructor for InputNeuron class.

        InputNeuron is a class of neuron that can be used to inject spikes into
        the network. When involved in a simulation, an InputNeuron will generate
        spikes at the times specified during its construction.

        Inputs:
         spiketrain is an array or list of spike times
        '''
        self.spikes = np.array(spiketrain)
        self.idx = []  # List index, when added to a network

    def SpikesBetween(self, t_start, t_end):
        '''
        numspikes = InputNeuron.SpikesBetween(t_start, t_end)

        Returns the number of times the neuron spiked between t_start and t_end.
        Specifically, it counts a spike if it occurred at t, where
        t_start <= t < t_end
        '''
        sp_bool = np.logical_and( np.array(self.spikes)>=t_start, np.array(self.
↪spikes)<t_end )
        return np.sum(sp_bool)

    def Set_idx(self, idx):
        self.idx = idx
    def Get_idx(self):
        '''
         idx = InputNeuron.Get_idx()

         Returns the index of the neuron in its network.
        '''
        return self.idx

    def Slope(self):
        return

    def Step(self, dt):
        return

    def Get_spikes(self):
```

```
        return self.spikes
```

1.1.1 Example use

[5]:
```
spikes = GenerateSpikeTrain([10, 25], [0.2, 0.5])  # Generates a specified spike train
PlotSpikeRaster([spikes])
```



[6]:
```
neur_in = InputNeuron(spikes)
print(neur_in.Get_spikes())  # Spike times
print(neur_in.SpikesBetween(0.3,0.4)) # Number of spikes between 0.3 and 0.4
```

```
[0.05 0.15 0.22 0.26 0.3  0.34 0.38 0.42 0.46 0.5 ]
3
```

2 Q1: Implementing LIFNeuron Class

2.1 LIFNeuron class

[7]:
```
class LIFNeuron(object):

    def __init__(self, Tau_m=0.02, Tau_ref=0.002, Tau_s=0.05):
        '''
        LIFNeuron(Tau_m=0.02, Tau_ref=0.002, Tau_s=0.05)
```

```
        Constructor for LIFNeuron class

        Inputs:
         Tau_m   membrane time constant, in seconds (s)
         Tau_ref refractory period (s)
         Tau_s   synaptic time constant (s)
        '''
        self.tau_m = Tau_m      # membrane time constant
        self.tau_ref = Tau_ref  # refractory period
        self.tau_s = Tau_s      # synaptic time constant
        self.v = 0.             # sub-threshold membrane potential (voltage)
        self.s = 0.             # post-synaptic current (PSC)
        self.t = 0.             # current time
        self.dvdt = 0.          # slope of v (w.r.t. time)
        self.dsdt = 0.          # slope of s (w.r.t. time)

        self.idx = []           # List index, when added to a network

        self.weighted_incoming_spikes = 0.  # weighted sum of incoming spikes␣
↪(for one time step)
        self.ref_remaining = 0. # amount of time remaining in the refractory␣
↪period

        # For plotting
        self.v_history = []     # records v over time
        self.s_history = []
        self.spikes = []        # list of times when this neuron spiked


    # A bunch of set and get functions.
    def Set_v(self, v):
        self.v = v
    def Get_v(self):
        return self.v
    def Set_s(self, s):
        self.s = s
    def Get_s(self):
        return self.s
    def Set_ref_remaining(self, ref_remaining):
        self.ref_remaining = ref_remaining
    def Get_ref_remaining(self):
        return self.ref_remaining
    def Set_idx(self, idx):
        self.idx = idx
    def Get_idx(self):
        return self.idx
```

```python
    def Get_spikes(self):
        return self.spikes
    def Set_t(self, t):
        self.t = t
    def Get_v_history(self):
        return self.v_history

    def SpikesBetween(self, t_start, t_end):
        '''
        numspikes = LIFNeuron.SpikesBetween(t_start, t_end)

        Returns the number of times the neuron spiked between t_start and t_end.
        Specifically, it counts a spike if it occurred at t, where
        t_start <= t < t_end
        '''
        sp_bool = np.logical_and( np.array(self.spikes)>=t_start, np.array(self.
↪spikes)<t_end )
        return np.sum(sp_bool)

    def Slope(self):
        '''
        LIFNeuron.Slope()

        Evaluates the right-hand side of the differential equations that
        govern v and s. The slopes get stored in the internal variables
          self.dvdt, and
          self.dsdt
        '''
        #===== REPLACE THE CODE BELOW =====
        self.dvdt = (self.s - self.v)/self.tau_m
        self.dsdt = -self.s/self.tau_s


    def Step(self, dt):
        '''
        LIFNeuron.Step(dt)

        Updates the LIF neuron state by taking an Euler step in v and s.
        The length of the step is dt seconds.
        If v reaches the threshold of 1, the neuron fires an action potential
        (spike). Linear interpolation should be used to estimate the time that␣
↪v=1.
        The spike time is appended to the list self.spikes, and v
        is set to zero. After a spike, the neuron is dormant for self.tau_ref
        seconds.

        '''
```

```
        #==== PLACE YOUR CODE HERE ====

        # check Value of V, if V==1: reset
        if self.Get_v() == 1:
            self.Set_v(0)

        self.Slope()
        self.Set_s(self.Get_s() + dt * self.dsdt + self.
 ↪weighted_incoming_spikes / self.tau_s)


        dt_real = dt - self.Get_ref_remaining()
        if dt_real < 0:
            self.Set_ref_remaining(self.Get_ref_remaining() - dt)

#          activated but not fire a spike
        elif self.Get_v() + dt_real * self.dvdt < 1:
            self.Set_v(self.Get_v() + dt_real * self.dvdt)
            self.Set_ref_remaining(0)

#          fire spike
        else:
            dt_to_1 = (1-self.Get_v()) /self.dvdt
            self.spikes.append(self.t + self.Get_ref_remaining() + dt_to_1)
            self.Set_v(1)
            self.Set_ref_remaining(self.tau_ref-(dt - self.Get_ref_remaining()␣
 ↪- dt_to_1))

        self.t = self.t + dt
        # Store v (for plotting), and reset incoming spike accumulator
        self.v_history.append(self.v)
        self.weighted_incoming_spikes = 0.




    def ReceiveSpike(self, w):
        '''
        LIFNeuron.ReceiveSpike(w)

        Registers the arrival of a spike from a presynaptic neuron. The
        member variable self.weighted_incoming_spikes keeps track of all
        the incoming spikes, each weighted by their respective connection
        weights. It is sufficient to add them all together to tabulate the
        total incoming weighted spikes (from all presynaptic neurons).

        Input:
```

```
         w    is the connection weight from the presynaptic neuron.
        '''
        self.weighted_incoming_spikes += w


    def ShowState(self):
        '''
         LIFNeuron.ShowState()

         Prints the current time, membrane potential, input current, and
         remaining refractory time.
        '''
        print('{0:6.4f}s: s={1:5.3f}, v={2:6.4f}, ref remaining={3:7.5f}'.
↪format(self.t, self.s, self.v, self.ref_remaining))
```

2.2 Tests

The test below should yield,

```
0.0000s: s=0.000, v=0.0000, ref remaining=0.00000
0.0010s: s=0.600, v=0.0000, ref remaining=0.00000
0.0020s: s=0.588, v=0.0300, ref remaining=0.00000
```

[8]:
```
lif_a = LIFNeuron(Tau_m=0.02, Tau_ref=0.002, Tau_s=0.05)

# If you want to run the version from the supplied solutions, use
# the line below instead.
#lif_a = solutions.LIFNeuron(Tau_m=0.02, Tau_ref=0.002, Tau_s=0.05)

lif_a.ReceiveSpike(0.03)
lif_a.ShowState()
lif_a.Slope()
lif_a.Step(0.001)
lif_a.ShowState()
lif_a.Slope()
lif_a.Step(0.001)
lif_a.ShowState()
```

```
0.0000s: s=0.000, v=0.0000, ref remaining=0.00000
0.0010s: s=0.600, v=0.0000, ref remaining=0.00000
0.0020s: s=0.588, v=0.0300, ref remaining=0.00000
```

The test below should yield,

```
0.0000s: s=0.500, v=0.0000, ref remaining=0.00150
0.0010s: s=0.490, v=0.0000, ref remaining=0.00050
0.0020s: s=0.480, v=0.0123, ref remaining=0.00000
```

[9]:
```
lif_a = LIFNeuron(Tau_m=0.02, Tau_ref=0.002, Tau_s=0.05)
lif_a.Set_v(0.)
lif_a.Set_s(0.5)
lif_a.Set_ref_remaining(0.0015)
lif_a.ShowState()
lif_a.Slope(); lif_a.Step(0.001)
lif_a.ShowState()
lif_a.Slope(); lif_a.Step(0.001)
lif_a.ShowState()
```

```
0.0000s: s=0.500, v=0.0000, ref remaining=0.00150
0.0010s: s=0.490, v=0.0000, ref remaining=0.00050
0.0020s: s=0.480, v=0.0123, ref remaining=0.00000
```

The test below should yield,

```
0.3000s: s=1.300, v=0.9800, ref remaining=0.00000
0.3010s: s=1.274, v=0.9960, ref remaining=0.00000
0.3020s: s=1.249, v=1.0000, ref remaining=0.00129
0.3030s: s=1.224, v=0.0000, ref remaining=0.00029
0.3040s: s=1.199, v=0.0436, ref remaining=0.00000
Spike occurred at 0.30129 seconds
```

[10]:
```
lif_a = LIFNeuron(Tau_m=0.02, Tau_ref=0.002, Tau_s=0.05)
lif_a.Set_t(0.3)
lif_a.Set_v(0.98)
lif_a.Set_s(1.3)
dt = 0.001
lif_a.ShowState()
lif_a.Slope(); lif_a.Step(dt); lif_a.ShowState()
lif_a.Slope(); lif_a.Step(dt); lif_a.ShowState()
lif_a.Slope(); lif_a.Step(dt); lif_a.ShowState()
lif_a.Slope(); lif_a.Step(dt); lif_a.ShowState()
if len(lif_a.Get_spikes())>0:
    print('Spike occurred at {0:.5g} seconds'.format(lif_a.Get_spikes()[-1]))
```


```
0.3000s: s=1.300, v=0.9800, ref remaining=0.00000
0.3010s: s=1.274, v=0.9960, ref remaining=0.00000
0.3020s: s=1.249, v=1.0000, ref remaining=0.00129
0.3030s: s=1.224, v=0.0000, ref remaining=0.00029
0.3040s: s=1.199, v=0.0436, ref remaining=0.00000
Spike occurred at 0.30129 seconds
```

3 Q2: Implementing `SpikingNetwork`

3.1 `SpikingNetwork` Class

[11]:
```
class SpikingNetwork(object):

    def __init__(self):
        '''
        SpikingNetwork()

        Constructor for SpikingNetwork class.

        The SpikingNetwork class contains a collection of neurons,
        and the connections between those neurons.
        '''
        self.neurons = []       # List of neurons (of various kinds)
        self.connections = []   # List of connections
        self.t_history = []     # List of time stamps for the Euler steps
                                # (Useful for plotting)

    def GetNeuron(self, idx):
        '''
         neur = SpikingNetwork.GetNeuron(idx)

         Returns the Neuron object at index idx.
        '''
        return self.neurons[idx]

    def GetConnection(self, c):
        '''
         con = SpikingNetwork.GetConnection(c)
        '''
        return self.connections[c]

    def AddNeuron(self, neur):
        '''
        SpikingNetwork.AddNeuron(neuron)

        Adds a neuron to the network.

        Input:
         neuron is an object of type LIFNeuron or InputNeuron
        '''
        self.neurons.append(neur)
        neur.idx = len(self.neurons)-1

    def Connect(self, pre, post, w):
        '''
```

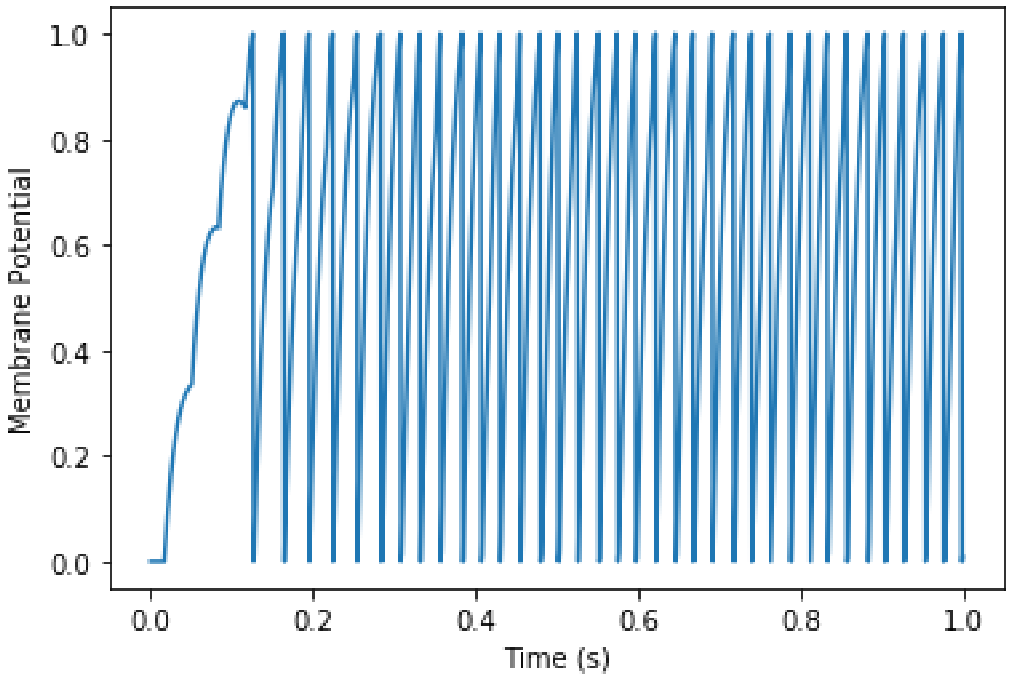
```
        SpikingNetwork.Connect(pre, post, w)

        Connects neuron 'pre' to neuron 'post' with a connection
        weigth of w.

        Each "connection" is stored as a list of 3 numbers of the form:
         [ pre_idx, post_idx, weight ]
        where
         pre_idx is the list index of the pre-synaptic neuron,
         post_idx is the list index of the post-synaptic neuron, and
         weight is the connection weight.
        eg. self.connections = [[0,1,0.05], [1,2,0.04], [1,0,-0.2]]
        '''
        self.connections.append([pre, post, w])


    def Simulate(self, T, dt):
        '''
        SpikingNetwork.Simulate(T, dt)

        Simulates the network for T seconds by taking Euler steps
        of size dt.

        Inputs:
         T    how long to integrate for
         dt   time step for Euler's method
        '''

        # This code takes care of recording time samples in a way
        # that allows continuation. You needn't fuss over this code.
        current = 0 if len(self.t_history)==0 else self.t_history[-1]
        t_segment = np.arange(current, current+T, dt)


        #==== HERE IS WHERE YOUR CODE STARTS ====

        # Loop over time steps (I've set that up for you)
        for tt in t_segment:
            self.t_history.append(tt)  # Record time stamp

            # get connections points to each neuros, receive the spikes
            for post_neuron in self.neurons:
#                 conn = [c for c in self.connections if c[1] == post_neuron.
 ↪idx]
                # reset the incoming_spike_weight
                incoming_weight = 0.
                for connection in self.connections:
```

```
                    # get incoming spike weight from each prev_neurons
                    if connection[1] == post_neuron.idx:
                        prev_neuron = self.neurons[connection[0]]
                        incoming_weight += connection[2] * prev_neuron.
↪SpikesBetween(tt - dt, tt)
                        post_neuron.weighted_incoming_spikes = incoming_weight

                # call slope and spike
                post_neuron.Slope()
                post_neuron.Step(dt)

        #==== PLACE YOUR CODE HERE ====



    def AllSpikeTimes(self):
        '''
        SpikingNetwork.AllSpikeTimes()

        Returns all the spikes of all the neurons in the network.
        Useful for making spike-raster plots of network activity.

        Output:
         all_spikes  a list of sublists, where each sublist holds
                     the spike times of one of the neurons
        '''
        blah = []
        for neur in self.neurons:
            blah.append(np.array(neur.Get_spikes()))
        return blah
```

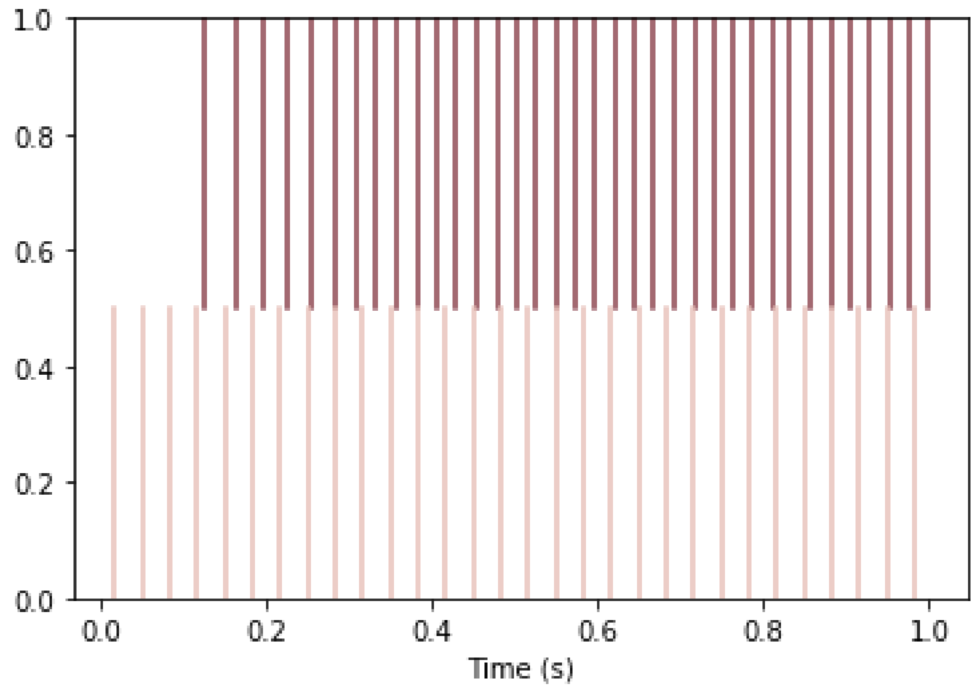
3.1.1 Test

[12]:
```
# This will create a small network to test on. This network is shown in Fig.␣
↪1(a).
net = SpikingNetwork()
A = LIFNeuron(Tau_s=0.1)  # Create a LIF neuron
in1 = InputNeuron( GenerateSpikeTrain([30], [1]) )  # Create an input neuron

net.AddNeuron(A)  # Neuron 0 (uses default values for tau_m and tau_ref)
net.AddNeuron(in1)  # Input neuron generates 30 Hz spike train
net.Connect(in1.Get_idx(), A.Get_idx(), 0.05)  # Connection from neuron 1 to␣
↪neuron 0
net.Simulate(1, 0.001) # Simulate for 1 second
```

[13]:
```
# Plot the membrane potential of net.neuron[0]
plt.plot(net.t_history, net.GetNeuron(0).Get_v_history());
plt.xlabel('Time (s)')
plt.ylabel('Membrane Potential');
```



[14]:
```
# Plot the spike rasters for all the neurons in the network.
PlotSpikeRaster( net.AllSpikeTimes() )
```



4 Q3: Experiments with Spiking Networks

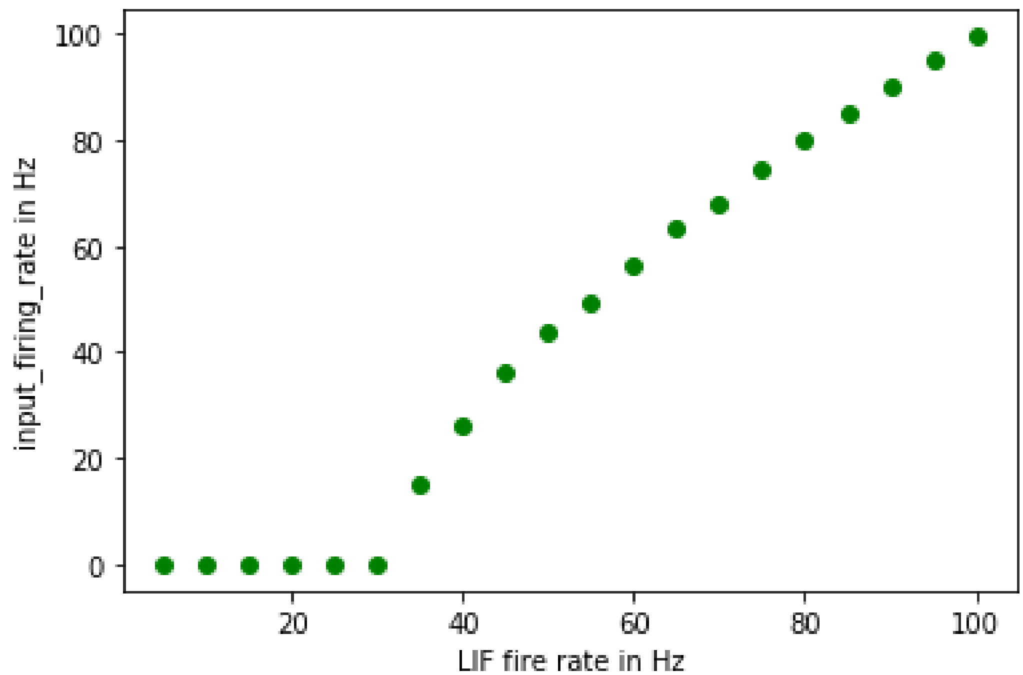
4.1 (a) LIF Firing Rate Curve

[15]:
```
import numpy as np
# i.
# This will create a small network to test on. This network is shown in Fig.␣
↪1(a).
net = SpikingNetwork()
A = LIFNeuron(Tau_s=0.1)  # Create a LIF neuron

# parameters to generate spiketrains 5,10,15...100 Hz, each for 2 seconds
input_firing_rate = np.arange(5,101,5)
time_period = np.arange(2,len(input_firing_rate)*2+1,2)


# Create an input neuron
in1 = InputNeuron( GenerateSpikeTrain(input_firing_rate, time_period))
# Neuron 0 (uses default values for tau_m and tau_ref)
net.AddNeuron(A)
# Input neuron generates 30 Hz spike train
net.AddNeuron(in1)
# Connection from neuron 1 to neuron 0
net.Connect(in1.Get_idx(), A.Get_idx(), 0.03)
```

```
# Simulate for 40 second
net.Simulate(40, 0.001)
```

[16]:
```
# ii.
lif_fire_rate = []
for t in range(0,40,2):
    fireRate = A.SpikesBetween(t, t + 2)/2
    lif_fire_rate.append(fireRate)
#     print('{:d}\t{:d}\t{:.2f}'.format(t,t+2,fireRate))
```

[17]:
```
# Make a plot of input firing rate vs LIF firing rate

plt.plot(input_firing_rate, lif_fire_rate,'og');
plt.xlabel('LIF fire rate in Hz')
plt.ylabel('input_firing_rate in Hz');
```



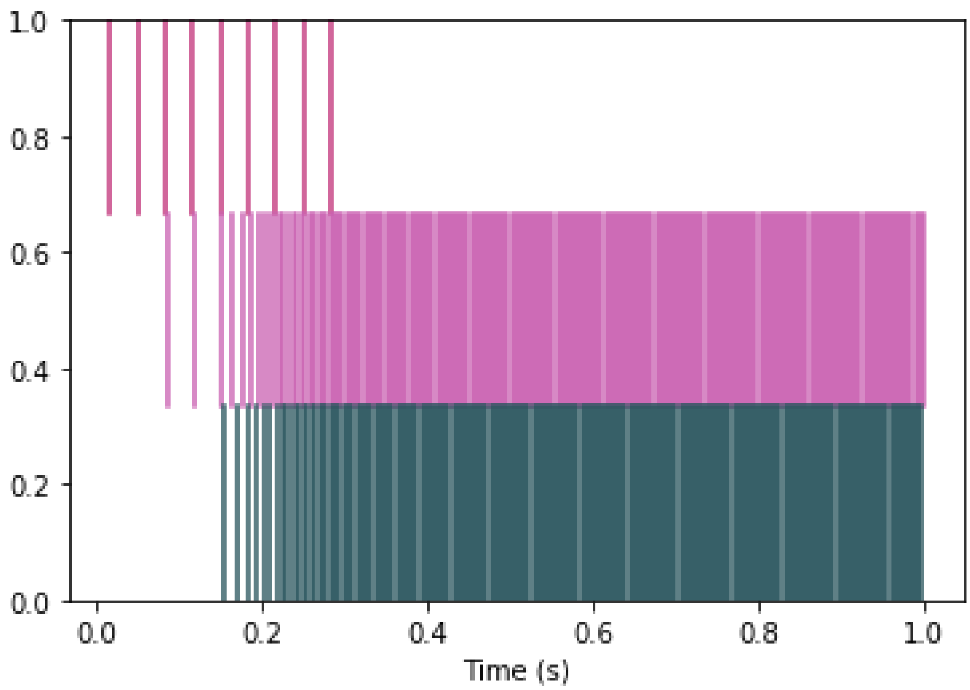
4.2 (b) Two LIF Neurons

[18]:
```
# i.
# This will create a small network to test on. This network is shown in Fig.
↪1(b).
net_1b = SpikingNetwork()
# Create LIF neuron A and B
# default: Tau_m=0.02, Tau_s=0.05, Tau_ref=0.002
```

```
A = LIFNeuron()
B = LIFNeuron()
# create inputNeuron
in_1b = InputNeuron(GenerateSpikeTrain([30],[0.3]))
# add input_neuron, A, and B to the neuron network
net_1b.AddNeuron(in_1b)
net_1b.AddNeuron(A)
net_1b.AddNeuron(B)

# connect input->A, A->B, and B->a with weight=0.5
net_1b.Connect(in_1b.Get_idx(), A.Get_idx(), 0.05)
net_1b.Connect(A.Get_idx(), B.Get_idx(), 0.05)
net_1b.Connect(B.Get_idx(), A.Get_idx(), 0.05)
net_1b.Simulate(1, 0.001)
```

[19]:

```
# ii.
PlotSpikeRaster(net_1b.AllSpikeTimes())
```



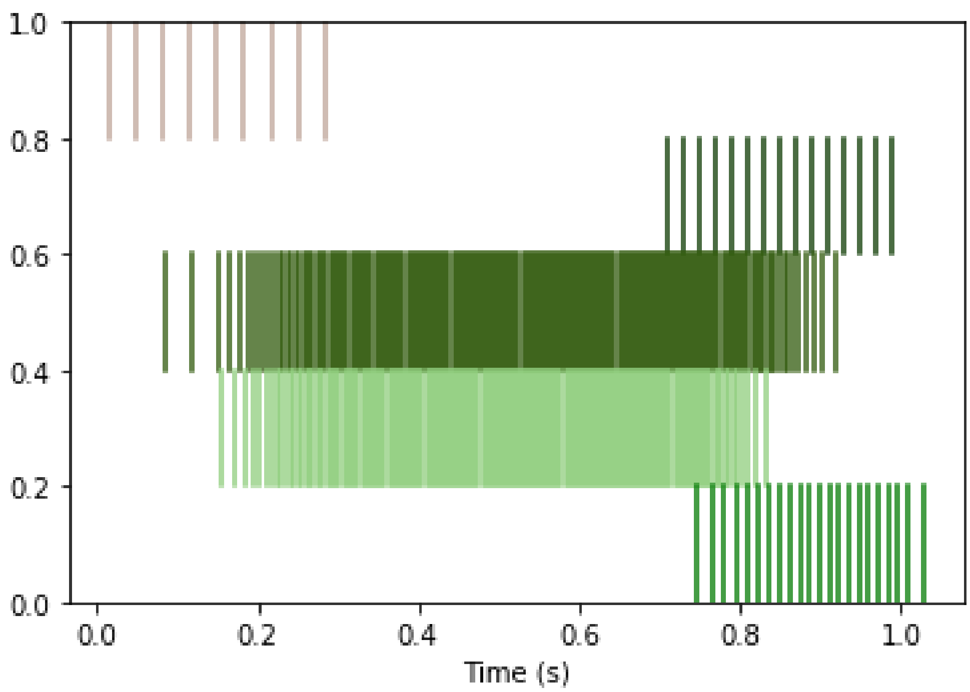
4.3 (c) Three LIF Neurons: Inhibition

[20]:

```
# i.
# This will create a small network to test on. This network is shown in Fig.␣
↪1(b).
net_1c = SpikingNetwork()
```

```
# Create LIF neuron A and B and C
# default: Tau_m=0.02, Tau_s=0.05, Tau_ref=0.002
A = LIFNeuron()
B = LIFNeuron()
C = LIFNeuron()
# create inputNeurons connected to A and C
in_1c_A = InputNeuron(GenerateSpikeTrain([30],[0.3]))
in_1c_C = InputNeuron(GenerateSpikeTrain([0, 50],[0.7, 1.]))
# add input_neurons and LIF neurons
net_1c.AddNeuron(in_1c_A)
net_1c.AddNeuron(in_1c_C)
net_1c.AddNeuron(A)
net_1c.AddNeuron(B)
net_1c.AddNeuron(C)



# connect input->A, A->B, and B->A with weight=0.05
net_1c.Connect(in_1c_A.Get_idx(), A.Get_idx(), 0.05)
net_1c.Connect(A.Get_idx(), B.Get_idx(), 0.05)
net_1c.Connect(B.Get_idx(), A.Get_idx(), 0.05)
net_1c.Connect(in_1c_C.Get_idx(), C.Get_idx(), 0.05)
net_1c.Connect(C.Get_idx(), B.Get_idx(), -0.2)
net_1c.Simulate(1.5, 0.001)
```

[21]:
```
# ii.
PlotSpikeRaster(net_1c.AllSpikeTimes())
```



4.3.1 iii.

After 0.7 second, the second inputneuro fired spikes in frequency of 70 Hz. It soonly activated the neuron C and make it fire spikes to B. Because of the negative weight of connection from C to B, neuron B was inhibited first. After 0.7s, the neuron B fires less so that A receive spike weight from B less frequently. Finally both A and B were inhibited by second input neuron and C. It explains why we see less spikes for neuron A and B in the plot.

4.4 (d) Ring Oscillator

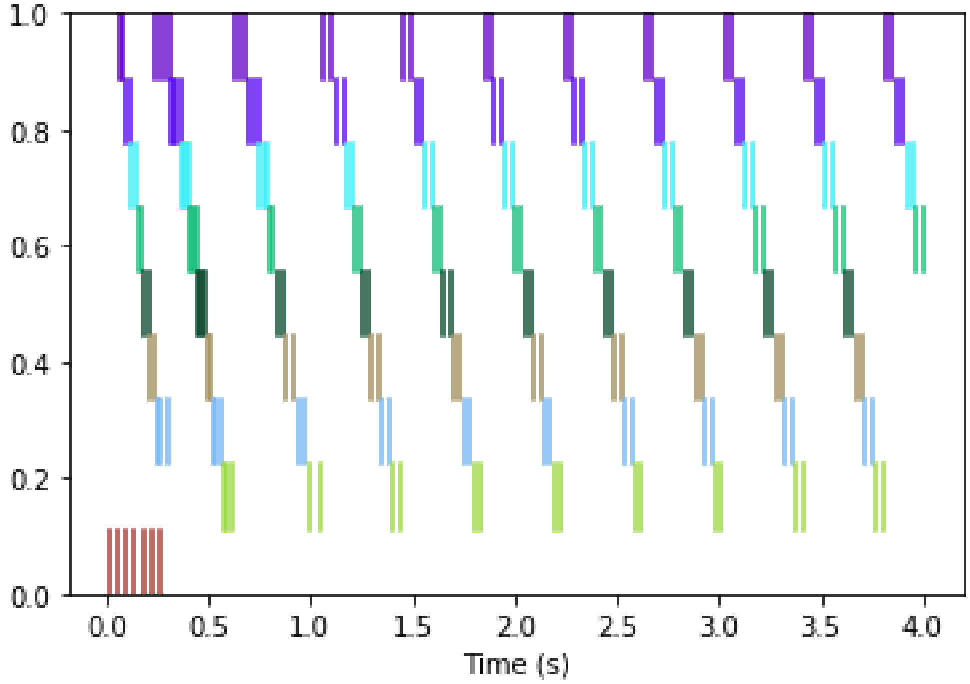
[22]:
```
# i.
# This will create a small network to test on. This network is shown in Fig.␣
↪1(b).
net_1d = SpikingNetwork()

# default: Tau_m=0.02, Tau_s=0.05, Tau_ref=0.002
# Create LIF neurons A-H, with Tau_m = 50 ms, and Tau_s = 100 ms
A = LIFNeuron(Tau_m=0.05, Tau_s=0.1)
B = LIFNeuron(Tau_m=0.05, Tau_s=0.1)
C = LIFNeuron(Tau_m=0.05, Tau_s=0.1)
D = LIFNeuron(Tau_m=0.05, Tau_s=0.1)
E = LIFNeuron(Tau_m=0.05, Tau_s=0.1)
F = LIFNeuron(Tau_m=0.05, Tau_s=0.1)
G = LIFNeuron(Tau_m=0.05, Tau_s=0.1)
H = LIFNeuron(Tau_m=0.05, Tau_s=0.1)


# add LIF neurons
net_1d.AddNeuron(A)
net_1d.AddNeuron(B)
net_1d.AddNeuron(C)
net_1d.AddNeuron(D)
net_1d.AddNeuron(E)
net_1d.AddNeuron(F)
net_1d.AddNeuron(G)
net_1d.AddNeuron(H)

# connect A->B...->H->A with weight 0.2(excitatory ring)
net_1d.Connect(A.Get_idx(), B.Get_idx(), 0.2)
net_1d.Connect(B.Get_idx(), C.Get_idx(), 0.2)
net_1d.Connect(C.Get_idx(), D.Get_idx(), 0.2)
net_1d.Connect(D.Get_idx(), E.Get_idx(), 0.2)
net_1d.Connect(E.Get_idx(), F.Get_idx(), 0.2)
net_1d.Connect(F.Get_idx(), G.Get_idx(), 0.2)
net_1d.Connect(G.Get_idx(), H.Get_idx(), 0.2)
```

```
net_1d.Connect(H.Get_idx(), A.Get_idx(), 0.2)
```

[23]:
```
# ii.
# connect A->B...->H->A with weight -0.4(inhibitory ring)
net_1d.Connect(A.Get_idx(), H.Get_idx(), -0.4)
net_1d.Connect(B.Get_idx(), A.Get_idx(), -0.4)
net_1d.Connect(C.Get_idx(), B.Get_idx(), -0.4)
net_1d.Connect(D.Get_idx(), C.Get_idx(), -0.4)
net_1d.Connect(E.Get_idx(), D.Get_idx(), -0.4)
net_1d.Connect(F.Get_idx(), E.Get_idx(), -0.4)
net_1d.Connect(G.Get_idx(), F.Get_idx(), -0.4)
net_1d.Connect(H.Get_idx(), G.Get_idx(), -0.4)
```

[24]:
```
# iii.

# create inputNeurons ( fire @25 Hz for the first 0.3 seconds,then go dormant.)
in_1d = InputNeuron(GenerateSpikeTrain([25],[0.3]))

# add inputNeuron to network
net_1d.AddNeuron(in_1d)

# connect input to A with weight 0.2
net_1d.Connect(in_1d.Get_idx(), A.Get_idx(), 0.2)
```

[25]:
```
# iv.

# net_1d.Simulate(4, 0.001)
# PlotSpikeRaster(net_1d.AllSpikeTimes())
net_1d.Simulate(4, 0.001)
PlotSpikeRaster(net_1d.AllSpikeTimes())
```



[26]:
```
for s in A.spikes:
    print('{:.5f}'.format(s))
```

```
0.06424
0.08741
0.23990
0.26609
0.28546
0.31040
0.63412
0.65624
0.68671
1.06609
1.10637
1.45990
1.49526
1.85305
1.88551
2.24637
2.27789
2.64000
2.67119
3.03296
3.06413
3.42596
```

3.45712
3.81896
3.85012

=== YOUR ANSWER HERE === It takes around 0.4 s

5 Q4: Neural Activation Functions

5.1 (a) ReLU derivative

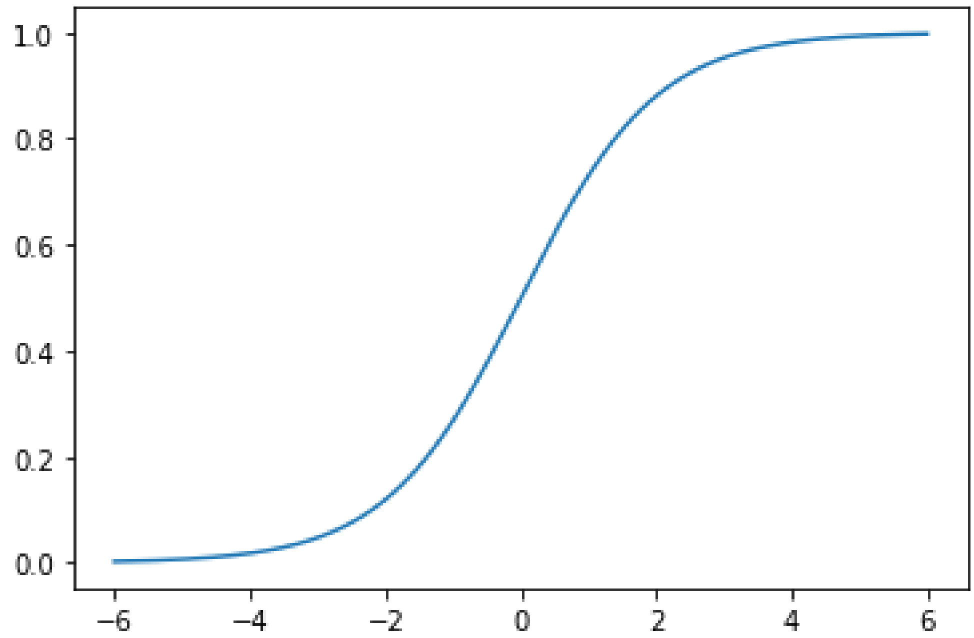
$$f(x) = ReLU(x) = max(x, 0) = \begin{cases} 0 & if\ x <= 0 \\ x & if\ x > 0 \end{cases} \tag{1}$$

$$f'(x) = \begin{cases} 0 & if\ x <= 0 \\ 1 & if\ x > 0 \end{cases} \tag{2}$$

5.2 (b.i) Logistic

[27]:
```
# define z
z = np.arange(-6, 6, 0.01)

# standard_logistic
def standard_logistic(z):
    return 1/(1+ np.exp(-z))

# plot
plt.figure()
plt.plot(z, standard_logistic(z))
```

[27]:
```
[<matplotlib.lines.Line2D at 0x10e8e3c90>]
```



$$f(z) = \frac{1}{1+e^{-z}} \tag{3}$$

$$f'(z) = \frac{e^{-z}}{(e^{-z}+1)^2} \tag{4}$$

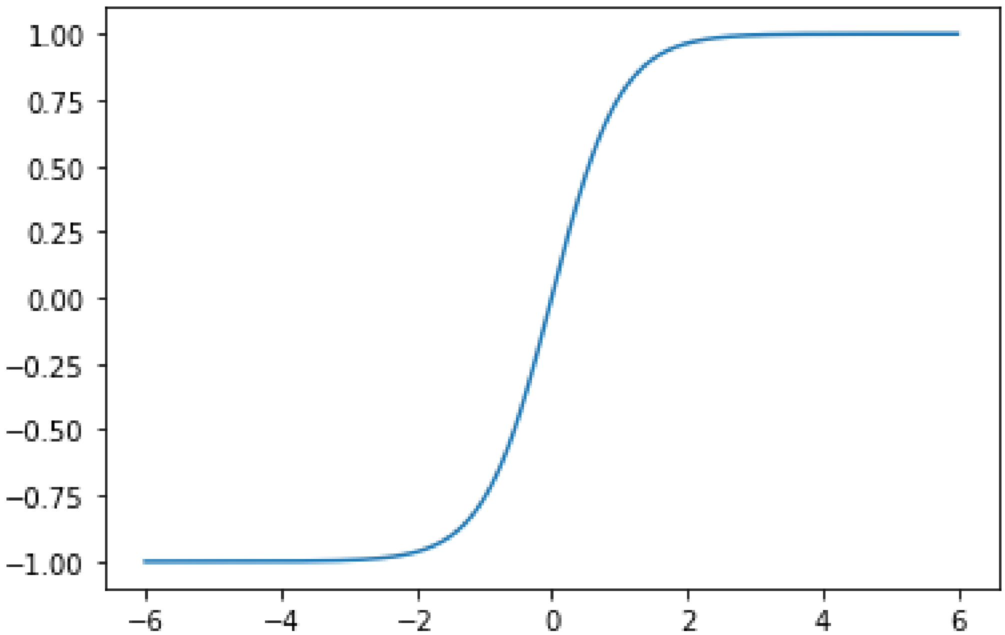
$$f'(0) = \frac{1}{(1+1)^2} = \frac{1}{4} \tag{5}$$

The derivative reaches 0 when z approaches $-\infty$ or ∞. The derivative reaches its maximum at z=0. The maximum of the derivative is $\frac{1}{4}$

5.3 (b.ii) tanh

[28]:
```
# tanh function
def tanh(z):
    return np.tanh(z)

# plot
plt.figure()
plt.plot(z, tanh(z))
```

[28]: `[<matplotlib.lines.Line2D at 0x10e955450>]`



$$f(z) = tanh(z) = \frac{sinh(z)}{cosh(z)} \tag{6}$$

$$sinh(z) = \frac{e^z - e^{\text{-}z}}{2} \qquad cosh(z) = \frac{e^z + e^{\text{-}z}}{2} \tag{7}$$

$$f'(z) = sech^2(z) = \frac{4}{(e^-z + e^z)^2} \tag{8}$$

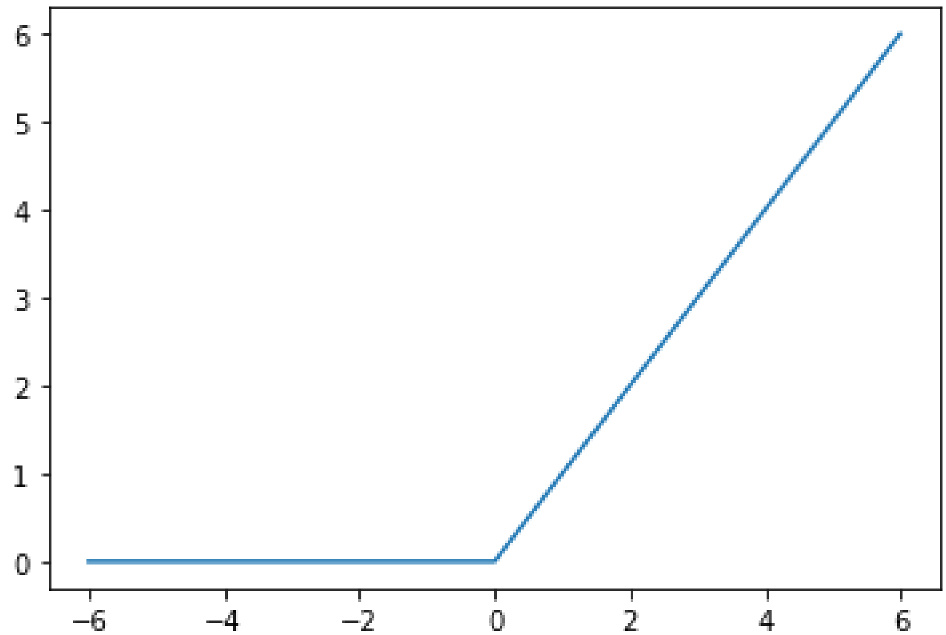
$$f'(0) = \frac{4}{(1+1)^2} = 1 \tag{9}$$

YOUR ANSWER HERE The derivative reaches 0 when z approaches $-\infty$ or ∞. The derivative reaches its maximum at z=0. The maximum of the derivative is 1.

5.4 (b.iii) ReLU

[29]:
```
# ReLU function
def ReLU(z):
    return np.maximum(np.zeros(len(z)), z)

# Plot
plt.figure()
plt.plot(z, ReLU(z))
```

[29]: `[<matplotlib.lines.Line2D at 0x10eb00d50>]`



YOUR ANSWER HERE The derivative reaches 0 when z <= 0. The derivative reaches its maximum at z>0. The maximum of the derivative is 1.

$$f(wx+b) \tag{10}$$

5.5 (c.i) Derivative of f with respect to x

$$\frac{d}{dx}f(wx+b) = \frac{df(wx+b)}{d(wx)} \cdot \frac{d(wx)}{dx} = w \cdot f'(wx+b) \tag{11}$$

5.6 (c.ii) Derivative of f with respect to w

$$\frac{d}{dw}f(wx+b) = \frac{df(wx+b)}{d(wx)} \cdot \frac{d(wx)}{dw} = x \cdot f'(wx+b) \tag{12}$$

5.7 (c.ii) Derivative of f with respect to b

YOUR ANSWER HERE

$$\frac{d}{db}f(wx+b) = \frac{df(wx+b)}{d(wx+b)} \cdot \frac{d(wx+b)}{db} = f'(wx+b) \tag{13}$$

```
[ ]:
```